DNS Resolution with Renewal Using Piggyback

Beakcheol Jang, Dongman Lee, Kilnam Chon, and Hyunchul Kim

Abstract: **Domain name system (DNS) is a primary identification mechanism for Internet applications. However, DNS resolutions often take an unbearably long time, and this could seriously impair the consistency of the service quality of Internet applications based on DNS such as World Wide Web. Several approaches reduce DNS resolution time by proactively refreshing expired cached records or prefetching available records beforehand, but these approaches have an inherent problem in that they cause additional DNS traffic. In this paper, we propose a DNS resolution time reduction scheme, named renewal using piggyback (RUP), which refreshes expired cached records by piggybacking them onto solicited DNS queries instead of by issuing additional DNS queries. This method decreases both DNS resolution time and DNS traffic since it reduces the number of queries generated to handle a given DNS resolution without generating additional DNS messages. Simulation results based on two large independent DNS traces show that our proposed approach much reduces not only the DNS resolution time but also the DNS traffic.**

Index Terms: **Domain name system (DNS), piggyback, renewal, renewal using piggyback (RUP).**

I. INTRODUCTION

Domain name system (DNS) is a primary identification mechanism for looking up hosts and servers for Internet applications such as the Web and electronic mail. Name resolution, one of the key DNS functions, maps a domain name to an IP address and vice versa. However, recent studies [1]–[3] warn that DNS resolutions often take unbearably long and this excessively long DNS lookup time can seriously impair the consistency of the service quality of DNS-based Internet applications, especially the Web. Wills and Shang [4] report that as many as approximately 30% of DNS resolutions for servers randomly chosen from a log at the National Laboratory for Applied Network Research (NLANR) exceed two seconds, and Cohen and Kaplan [1] report that as many as about 10% of DNS resolutions for servers drawn from a log of AT&T proxy servers take more than four seconds. Moreover, a noticeable number of DNS lookups are unanswered. Jung and Sit [5] report that 23% of all client lookups in an MIT trace fail to elicit any answer, even a failure indication. The query packets for these unanswered lookups, including retransmissions, account for more than half of all DNS query packets in the trace.

Why does the DNS resolution often take so much time? First, since DNS databases are hierarchically distributed, a DNS resolution may involve several DNS queries to multiple remote DNS servers. The second reason is that DNS uses static timeout value and retransmissions for reliable delivery because standard DNS query messages are delivered using user datagram protocol (UDP). As a result, DNS resolutions take in the order of seconds in the presence of packet losses. To alleviate these problems, DNS uses a caching scheme. Local DNS servers (LDNSs) cache all the records that they receive as responses to DNS queries, which are available until their time-to-live (TTL) values expire [6]–[9]. Several studies report that about 80% of DNS lookups are cache-hits [4], [10], but the DNS resolution time still tends to be unpredictable. It has been reported that about 50% of DNS resolutions take more than 100 ms, and that about 10% of DNS resolutions take more than one second [5]. However, there remains room for improvement in the DNS cache. 67% to 90.3% of DNS cache misses are previously seen misses [2], a problem which occurs because the cached records are expired even though they are in the DNS cache. If the expired records had been refreshed beforehand, the misses could have been removed.

In spite of such critical performance degradation, only a few papers have been published in the last several years on how to improve the DNS resolution time. Cohen and Kaplan propose a method called *pre-resolving*, which is based on the local availability of a Web page [1]. They also propose *renewal*, which reduces the cache-miss rate by proactively refreshing cached records when they expire [2]. Although these schemes do reduce DNS resolution time somewhat, this improvement comes at the cost of additional DNS traffic.

Although the existing approaches reduce DNS resolution time, they necessarily incur additional DNS traffic and do not fully exploit the unused potential of the DNS protocol. In this paper, we propose a DNS resolution time reduction scheme, named *renewal using piggyback* (RUP). It refreshes expired cached records by simply piggybacking them onto DNS query messages instead of refreshing them by issuing additional DNS query messages, like the renewal scheme requires. RUP decreases both the DNS resolution time and traffic since it reduces the number of queries generated to handle DNS resolutions without additional DNS query messages. Simulation results based on two large independent DNS traces show that the proposed scheme reduces the DNS resolution time and the number of DNS messages as much as 45.5% and maximum 45.8%, respectively. They also show that such improvements can be en-

Manuscript received April 29, 2007; approved for publication by Song Chong, Division III Editor, April 19, 2009.

This work was supported by the IT R&D program of MKE/IITA [2008-F-047-02, Development of Urban Computing Middleware] and the Ministry of Knowledge Economy, Korea, under the Information Technology Research Center support program supervised by the Institute of Information Technology Advancement [IITA-2009-C1090-0902-0006].

B. Jang is with the Department of Computer Science at North Carolina State University, Raleigh, USA, email: bcjang@gmail.com.

D. Lee is with the Department of Computer Science at Korea Advanced Institute of Science and Technology IT Convergence Campus (KAIST ICC), Daejeon, South Korea, email: dlee@icu.ac.kr.

K. Chon is with the Department of Computer Science at Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, email: chonkn@gmail.com.

H. Kim is with the School of Computer Science and Engineering at Seoul National University, Seoul, South Korea, email: hyunchulk@gmail.com.

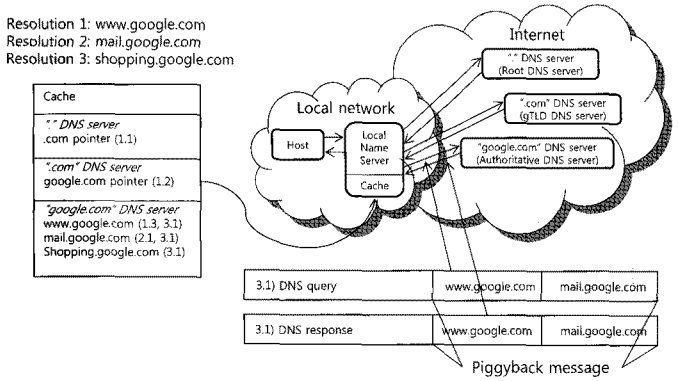


Fig. 1. Example of renewal using piggyback (RUP) mechanism; in which two expired cached records, www.google.com and mail.google.com are piggybacked and refreshed in the resolution 3.

sured regardless of the variation of TTL value. In the evaluation between the best RUP policy and the best renewal policy [2], the best RUP policy reduces the DNS resolution time more efficiently than the best renewal policy does. It also remarkably decreases the DNS traffic, while the best renewal policy increases it in proportion to the number of refreshed records, that is, it is not scalable.

The remainder of this paper is organized as follows. In Section II, we describe the existing DNS resolution time reduction schemes. Section III presents the proposed scheme in detail. In Section IV, we evaluate the proposed scheme. Section V provides an evaluation between the proposed scheme and the existing DNS resolution time reduction scheme. Finally, we conclude in Section VI.

II. RELATED WORKS

In this section, we discuss existing DNS resolution time reduction schemes. Cohen and Kaplan [1] conduct measurement studies based on a log of AT&T research proxy servers and verify that DNS resolution time is a major cause of long Web latencies. They propose *pre-resolving*, in which browsers or proxies perform DNS resolutions on all links or their subsets that appear in a Web page before a request to the server is issued, thereby eliminating DNS query time from user-perceived latency. However, though it achieves a reduction of Web latency through the improvement of DNS cache performance, it also incurs additional DNS traffic.

Another measurement is taken, showing that most DNS cache-misses, from about 67.0% to 90.3%, are expired misses. In response, Cohen and Kaplan propose a proactive caching scheme named *renewal* [2]. In the scheme, when a cached record is expired, a local DNS server (LDNS) automatically sends an unsolicited query message to the authoritative DNS server (ADNS) of the record. With a response from the ADNS, the expired cached record is refreshed and available for its TTL duration. The simulation results show that the proposed approach reduces the DNS cache-miss rate by as much as about 40% to 80%. However, this scheme increases the DNS traffic in direct proportion to the number of the records refreshed by the scheme. Moreover, it can impose a heavy burden on the DNS server because multiple DNS servers may be involved in refreshing an expired cached record. Specifically, when an LDNS does not know the ADNS of an expired cached record, it cannot send a DNS query directly to the ADNS and must send several DNS queries to upper-level ADNSs such as root or gTLD DNS servers to find the ADNS of the record. Even if the LDNS knows the ADNS of an expired cached record and directly sends just a DNS query to the ADNS in renewal, it is an excessive effort to use a DNS query to refresh an expired cached record. The ADNS will serve to put needless records in the response message of the query. For examples, let's assume that www.kaist.edu has three type A resource records (RRs) and three ADNSs, which are cached in a LDNS. Only one of type A RRs of www.kaist.edu is expired. In renewal, the LDNS sends a query message to an ADNS of kaist.edu to refresh the RR. The ADNS will send a DNS response including three type A RRs of www.kaist.edu in the answer section, three type NS RRs of kaist.edu in the authority section, and three type A RRs of the NS RRs in the additional section though the LDNS needs to refresh only one of type A RRs of www.kaist.edu. Like this, it is a waste of resource to use DNS queries to refresh an expired cached record.

III. RENEWAL USING PIGGYBACK

In this section, we describe our proposed scheme, RUP. RUP piggybacks expired cached records, refresh requests, onto a solicited DNS query message to refresh them. Piggybacking techniques incur low overhead and can be incrementally deployed because they exploit existing protocols [3]. Because of such benefits, piggybacking has been exploited for such processes as cache validations in proxy cache [11], sending hints [12], and reducing the connection setup time between clients and Web servers [3], among other uses.

A. Renewal Using Piggyback Mechanism

RUP only exploits solicited DNS queries for renewals of expired cached records instead of generating additional DNS queries. Whenever a LDNS should send a DNS query to an ADNS to answer a client's resolution, the expired cached records, the refresh questions, authorized by the ADNS are piggybacked onto the query as the format of the resource record (RR). We do not recommend using the format of DNS question for the refresh question because we need to refresh not the answer of a question but an expired cached record in itself. The answer of a question must include other useless records with the record that we need to refresh. Moreover current DNS does not theoretically and practically support multiple DNS questions in a DNS query [6], [7], [9]. The ADNS sends a response, and the refreshed records are piggybacked onto the response. In RUP, the results of each query are cached and indexed per each ADNS. Once the records are refreshed, their TTL values get refreshed and they can be reused for their respective TTL value durations.

As shown in Fig. 1, the proposed scheme works as follows.

1) A host requests a resolution for www.google.com to its LDNS. If it has no related record in its cache, the LDNS

performs the resolution as follows: 1.1) The LDNS redirects the resolution to a root DNS server (RDNS), and the RDNS responds with an *NS* record for .com. The LDNS caches the result and it is indexed by the RDNS. 1.2) The LDNS redirects the resolution to the .com ADNS, and the ADNS responds with an *NS* record for google.com. The LDNS caches the result, which is indexed by the .com ADNS. 1.3) Finally, the resolution is redirected to the google.com ADNS, and the ADNS responds with the name to IP address mapping information (an *A* record) for www.google.com. The result is cached in the LDNS and indexed by the google.com ADNS and is sent to the host.

2) A host requests a resolution for mail.google.com to the LDNS. If the corresponding record is not in its cache but the *NS* record for google.com is cached and fresh at the LDNS, the following step is performed for the resolution: 2.1) The LDNS immediately redirects the resolution to the google.com ADNS. The ADNS responds with an *A* record for mail.google.com, the LDNS caches the result indexed by the google.com ADNS, and sends it to the host.
3) A host requests the resolution for shopping.google.com to the LDNS. If the *A* records for www.google.com and mail.google.com in the LDNS cache are expired but the NS record for google.com is cached and fresh, the resolution is performed as follows: 3.1) The LDNS immediately redirects the resolution to the google.com ADNS. At the same time, it piggybacks the expired cached records, the *A* records for www.google.com and mail.google.com, onto the query message. The ADNS responds with an *A* record of shopping.google.com and simultaneously piggybacks *A* records for www.google.com and mail.google.com. The *A* record for shopping.google.com is cached and indexed by the google.com ADNS and is sent to the host and, simultaneously, *A* records for www.google.com and mail.google.com are refreshed and made available for their TTL duration.

B. Renewal Using Piggyback Message Format

RUP message formats are designed to exploit the unused fields of the existing DNS protocol. We use the ARCOUNT field and the additional record section of the DNS message format [6], [7]. The ARCOUNT field specifies the number of RRs of the additional record section in a DNS response message, and although it exists, it is not used in a DNS query message. The additional record section carries RRs which may be helpful in using the RRs in other sections of a DNS response message; it exists but is empty in a DNS query message. For RUP queries, expired cached records are piggybacked onto the additional record section to the form of RR and the number of records is specified in the ARCOUNT field of a DNS query message. We say that the expired cached record is piggybacked in an RUP query message to an RUP question. In an RUP response message, the refreshed records are piggybacked onto the additional record section to the form of RR with existing additional records. The ARCOUNT field specifies the number of records in the additional record section. As above, the expired cached record is piggybacked in an RUP response message to an RUP answer.

DNS messages are limited to 512 bytes, and longer messages are truncated when UDP is used as a transmission mechanism [7]. BIND, the most popular DNS server software, transfers messages exceeding 512 bytes after truncation using TCP [9], but DNS extension mechanism [16] and the latest 9.0 version of BIND [9] extend the limit to 1280 bytes. We limit the size of an RUP message to the limits of DNS message. That is, an RUP query message piggybacks RUP questions within the 512 byte or 1280 byte limit by DNS server softwares rather than piggybacking all expired cached records indexed by an ADNS, and an RUP response message does the same for RUP answers. This may lead to a situation that the RUP response message cannot accommodate all the answers to the RUP questions in an RUP query message. In this case, an RUP response message piggybacks RUP answers up to the 512 byte or 1280 byte limit, starting from an answer to the first RUP question. For instance, if the LDNS sends 10 RUP questions in an RUP query, but the RUP response only has room for 6 answers, the ADNS answers first 6 RUP questions, and drops the rest of the RUP questions. The proposed RUP message format only exploits the unused fields of the existing DNS message format, and is interoperable with the current DNS.

C. Renewal Using Piggyback Polices

We naturally restrict the number of RUP questions in a DNS query by the limitations on the RUP message length (i.e., 512 bytes or 1280 bytes), and useful records should be piggybacked as much as possible. Here we present several RUP policies that determine which expired cached records should be piggybacked. In the policies, each cached record is stored and indexed per its ADNS. When a cached record is used, the cached records indexed by the same ADNS are sorted according to each RUP policy. Whenever an LDNS should send a query message to an ADNS to answer a host's lookup, the expired cached records indexed by the ADNS are piggybacked according to their order. Each RUP policy has a variable (m), the maximum number of RUP questions per RUP message, which limits the maximum number of RUP questions that can be piggybacked in a RUP message as many as its value. The LDNS can control the variable within the 512 byte or 1280 byte limit.

We describe our RUP policies by specifying how each of them sorts and piggybacks cached records. Our policy-design principles are to use only the information available locally at the DNS server cache, be simple enough to implement, and incur a minimum of overhead. We name our RUP policies to be analogous to the cache replacement policies, with analogies made based on the property of the request sequence exploited.

- RUP-FIFO (m): Each cached record is stored in a stack indexed by its ADNS. Once the records are stored, their sequence never changes. The expired cached records are piggybacked in the order of first-stored-first-piggybacked, which is similar to the first-in-first-out (FIFO) cache replacement policy [13], [14]. Unfortunately, it suffers from Belady's anomaly like the FIFO policy [13], [14].
- RUP-LRU (m): Each cached record is stored in a stack indexed by its ADNS. Whenever a cached record is used, it is

moved to the top of the stack. The expired cached records are piggybacked in the order of the stack. That is, the expired cached records are piggybacked in the order of most-recently-used. This policy is similar to the least-recently-used (LRU) cache replacement policy [13], [14] that removes cached items in the order of least-recently cached hits. It inherits the replacement overhead from the LRU policy.

- RUP-LFU (m): Each cached record is stored in an array indexed by its ADNS. Whenever a cached record is used, the priority of the cached record is by incremented as much as its fixed value, and the cached records indexed by the same ADNS as the cached record are sorted in priority sequence. The expired cached records are piggybacked according to the order of the array. This policy is similar to the least-frequently-used (LFU) cache replacement policy [14] that removes cached items in the order of the smallest number of hits, and it inherits the sorting overhead from the LFU policy.

The performance of RUP relies on the number of DNS queries sent by the LDNS to the ADNS. If there are a few such queries, the cached record barely gets refreshed by piggybacking. However, the more the traffic between the LDNS and ADNS increases, the more frequently the cached records are refreshed. The added cost of our mechanism is mainly the increased size of the regular DNS messages and additionally generated RUP questions due to piggybacking. However, our simulation result shows that the increased byte consumption by piggybacking is trivial or often reduces overall bytes consumed by the DNS cache policy [6], [7] (see Fig. 4). The number of RUP questions piggybacked onto a single query can be also controlled by changing the value of the maximum number of RUP questions per RUP message. Moreover, our simulation result reports that even when the maximum number of RUP questions per RUP message is one, the best RUP policy reduces the resolution time of the DNS cache policy as much as 23.8% (see Fig. 2). The cost for the LDNS cache is slightly increased as it must maintain a list of cached records on a per server basis. The additional cost for the ADNS is that it must refresh the piggybacked records in addition to processing the regular DNS question. However, in the absence of piggybacking, such requests may have to be done, in the future, by the LDNS in separate DNS queries.

IV. EVALUATION OF RENEWAL USING PIGGYBACK POLICIES

In this section, we evaluate the performance of RUP policies and suggest the best RUP policy as a solution to the problem of renewal. We compare the performance of the best RUP policy with that of the best renewal policy in the Section V.

A. *Evaluation Model*

A.1 Collected Traces

We construct trace-driven simulations based on two DNS traces, both of which were collected using tcpdump [17]. Table 1 presents the basic characteristics of our DNS traces. For our simulation, we exploit the resolutions that are successfully answered. The unanswered resolutions can not be simulated because they do not receive any answer message.

Table 1. Basic trace statistics.

	KAIST trace	Yonsei trace
Date	Nov 23-27, 2002	Feb 5-9, 2002
Place	KAIST, Daejeon, South Korea	Yonsei univ., Seoul, South Korea
Total lookup	16,966,061	8,996,167
Total query	6,533,016	2,854,049
Distinct host	9,268	6,865
Distinct DNS server	64,327	76,106
Distinct domain name	692,504	575,639

A.2 Parameter Used

We evaluate RUP policies by varying two parameters: The maximum number of RUP questions per RUP message and the TTL value.

- Maximum number of RUP questions per RUP message: This assessment is important since as the parameter increases, the LDNS with RUP policies sends more frequently a burst of RUP question onto a DNS query to an ADNS. The burst of requests onto a DNS query can make the ADNS instantaneously busy. Therefore, RUP policies should provide proper performance by keeping the value of this parameter as small as possible. We set the range of this parameter from 0 to 17 because the average of maximum values of this parameter is about 17 in our traces. We limit the size of RUP message to 512 bytes.
- Time-to-live value: The evaluation of this parameter is significant. The first reason for this is that the TTL value directly affects the performance of DNS cache. The second reason is that the trend of TTL value allocation is variable according to the times. For instance, the initial DNS recommends that TTL values should be on the order of days for the typical hosts [6]. However, the current abnormal uses of DNS prefer exceptionally low TTL values, usually only a few seconds or minutes, which deteriorate the DNS resolution time and increase the DNS traffic. Specifically, content distribution network (CDN) and popular Web sites with multiple servers allocate low TTL values to their RRs to help balance the load across servers, for fault tolerance, or to direct each client request to a geographically close server [15]. In mobile networking, dynamic DNS allocates low TTL values to provide the basis for host mobility support in the Internet as well [10]. Therefore, RUP policies must be able to consistently improve the performance of the DNS cache policy at all the TTL value as much as possible so that the performance of them is not dependent on the trend of TTL value allocation. We set the range of TTL values from one to 604,800 seconds (seven days) because the maximum TTL value for RRs is restricted to 604,800 seconds [9]. We use the scale of the log in the presentation, and the range is shown from one to 1,000,000 seconds.

We do not vary the cache space of the DNS server because it is not a significant issue in DNS. DNS caching differs in some

basic respects from content-caching. RRs have a considerably smaller size, and the storage space is ample with respect to the amount of data. In fact, the cache space of DNS servers is theoretically infinite in RFCs [6], [7], and there is not any practical limit in BIND [9].

A.3 Evaluation Metrics

We take the following as evaluation metrics: the average resolution time, the average number of queries per resolution, and the average bytes consumed per resolution, which have been traditionally considered when evaluating the performance of resource retrieval [11]. We also present the average number of RUP questions per RUP message as a measure of the evaluation. We do not consider the memory overhead as our evaluation metric because the memory usage is not important in DNS cache. The memory usage of DNS cache is trivial (at most a few mega bytes) because sizes of resource records are small (tens of bytes). In reality, DNS cache size is not limited, and expired cached records are not deleted in both the standards [6], [7] and the implementation [9].

- Average resolution time: This describes how long it takes on average for DNS lookups to receive responses. The primary purpose of our proposed scheme is to reduce the DNS resolution time.
- Average number of queries per resolution: This describes the average number of queries issued to handle a DNS resolution. It shows the network overheads of RUP policies in terms of the number of messages
- Average number of bytes consumed per resolution: This describes the average number of bytes consumed for a resolution by both requesting and requested DNS servers. It shows the network overheads of RUP policies in terms of the number of bytes.
- Average number of RUP questions per RUP message: This describes the average number of RUP questions piggybacked in an RUP message.

To avoid a duplication of the presentation, we do not provide the performance of the DNS cache as an evaluation metric. In fact, the average number of queries per resolution directly reflects the performance of the DNS cache.

A.4 Methodology

We evaluate the performance of the RUP policies, by considering the performance of the DNS cache policy as the base of the evaluation. For these simulations, we build our own simulators, which support the DNS cache policy [6], [7], [9] and the RUP policies. We assume that there is no packet loss in a network and no DNS server failure, and we apply name compression [6], [7]. All values applied for these simulations such as query times and TTL values of RRs are actual values in our traces. As in the experiment by Krishnamurthy and Wills [11], we add 0.1 ms to the response time for each RUP question to reflect the additional time needed to process a RUP question at both the client and the server points.

B. Evaluation Results

Here we vary the parameters to present various outcomes for our evaluation metrics. Our presentation and explanation are focused on the performance of the best RUP policy.

Fig. 2 through 5 present the distributions of the average resolution time, the average number of queries per resolution, the average bytes consumed per resolution, and the average number of RUP questions per RUP message, varying the maximum number of RUP questions per RUP message for our traces. Among RUP policies, RUP-LFU consistently shows the best performance for all our evaluation metrics. The performance of RUP-LRU is nearly similar to that of RUP-LFU. RUP-FIFO is less effective than RUP-LFU and RUP-LRU. In this evaluation, the DNS cache policy does not piggyback a RUP question, and its value for this parameter is always zero. However, for ease of comparison, we represent the performance of the DNS cache policy as lines in the following figures.

In Fig. 2, the best RUP policy, RUP-LFU always shows lower average resolution time than the DNS cache policy. The resolution time in RUP-LFU is sharply reduced as the maximum number of RUP questions per RUP message increase from one to five, and the reduction becomes dull when the maximum number of RUP questions per RUP message is six to ten. There shows little variation when the number becomes eleven or more in both our traces. Specifically, it decreases the resolution time of the DNS cache policy as much as 45.5% for the KAIST trace and 41.0% for the Yonsei trace. This is because RUP-LFU reduces the number of queries for resolutions on the DNS cache policy by increasing the performance of the DNS cache. Actually, users do not experience that each latency reduces by tens of milliseconds but feel that about 50% of DNS resolutions having taken from hundreds of millisecond to tens of second with the DNS cache policy [5] take nearly zero second with RUP-LFU because the cache miss-rate reduces as much as about 50%. Fig. 3 shows that RUP-LFU reduces the average number of queries per resolution of the DNS cache policy maximum by as much as 45.8% and by 40.9% for our traces at KAIST and Yonsei, respectively. In Fig. 4, although RUP-LFU increases the average bytes consumed per resolution of the DNS cache policy maximum by 3.2% and 4.9% for our traces, respectively, the increases are obviously trivial when we consider the benefits for the resolution time and the number of queries per resolution of RUP-LFU. Moreover, RUP-LFU reduces, rather than increases, the average bytes consumed per resolution of the DNS cache policy at most of the instances, the maximum numbers of RUP questions per RUP message of between one to fourteen for the KAIST trace and between one to thirteen for the Yonsei trace. In Fig. 5, RUP-LFU piggybacks averages of 5.0 and 4.1 RUP questions per RUP message at the worst case for each trace. One may argue that the average numbers of 5.0 and 4.1 RUP questions per RUP message imposes a heavy burden on the DNS server, but in the absence of piggybacking, such requests may have to be done, in the future, by the LDNS in separate DNS queries. Moreover, he can control the average number of RUP questions per RUP message by changing the value of the maximum number of RUP questions per resolution. Even when the average number of RUP questions per RUP message of about 0.70 for the KAIST trace and about 0.65 for the Yonsei trace, when the maximum number

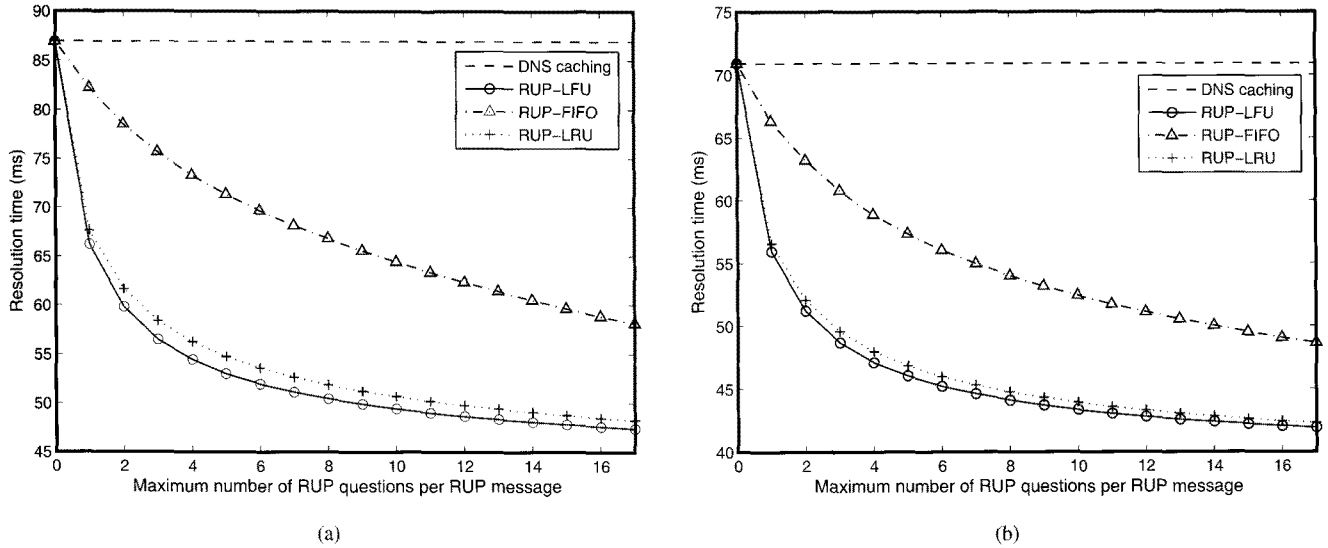


Fig. 2. Average resolution time as a function of the maximum number of RUP questions per RUP message: (a) KAIST DNS trace and (b) Yonsei university DNS trace.

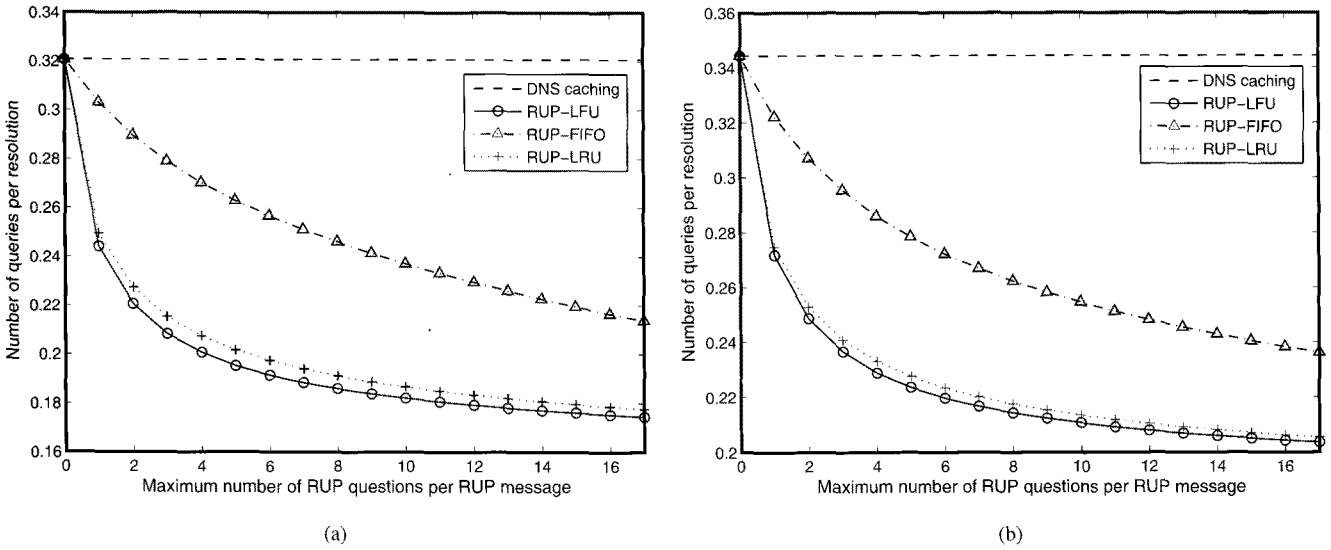


Fig. 3. Average number of queries as a function of the maximum number of RUP questions per RUP message: (a) KAIST DNS trace and (b) Yonsei university DNS trace.

of RUP questions per RUP message is one, RUP-LFU reduces the resolution time of the DNS cache policy as much as 23.8% and 20.1% for our traces at KAIST and Yonsei, respectively.

Figs. 6–9 present the distributions of the average resolution time, the average number of queries per resolution, the average bytes consumed per resolution, and the average number of RUP questions per RUP message, varying TTL value for our traces. As the results for the variation of the maximum number of RUP questions per RUP message, RUP-LFU shows the best performance among RUP policies for all our evaluation metrics. The performance of RUP-LRU is nearly similar to that of RUP-LFU. RUP-FIFO is less efficient than RUP-LFU and RUP-LRU.

In Fig. 6, our best policy, RUP-LFU decreases the average resolution time of the DNS cache policy at all the TTL values. Specifically, it reduces the average resolution time of the DNS cache policy as much as 46.7% for the KAIST trace at 700 seconds TTL value, and 42.6% at 1,000 seconds TTL values for the Yonsei trace. Fig. 7 shows that RUP-LFU decreases the average number of queries per resolution of the DNS cache policy regardless of TTL values. Specifically, it decreases the average number of queries per resolution of the DNS cache policy maximum by as much as 46.8% for the KAIST trace and by 42.7% for the Yonsei trace. Moreover, it consistently reduces both the resolution time and the number of queries per resolution of the DNS cache policy regardless of TTL values in Figs. 8 and 9. These facts prove that RUP-LFU can ensure the performance improvement regardless of the variation of TTL values. In terms of TTL value, the resolution time and the number of queries per resolution of RUP-LFU are improved as much as about ten times of those of the DNS cache policy. For the KAIST trace, the resolution times of RUP-LFU are 288.5 ms at 10 seconds TTL value and 152.2 ms at the TTL value of 100 seconds, while those of the DNS cache policy are 272.8 ms and 137.1 ms at the TTL value of 100 seconds and of 1000 seconds, respectively. In Fig. 8,

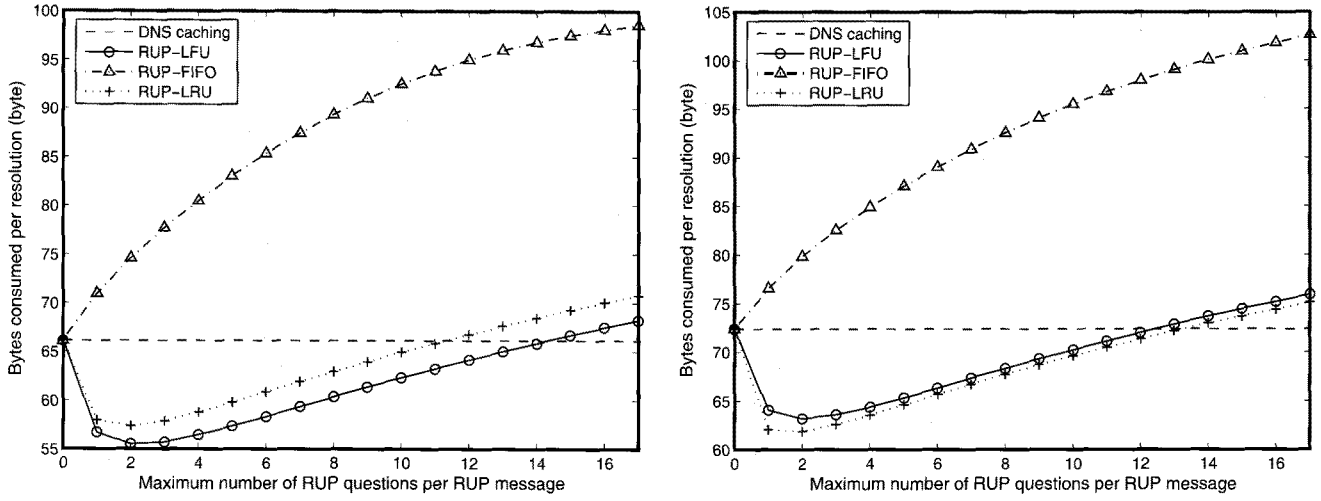


Fig. 4. Average number of bytes consumed as a function of the maximum number of RUP questions per RUP message: (a) KAIST DNS trace and (b) Yonsei university DNS trace.

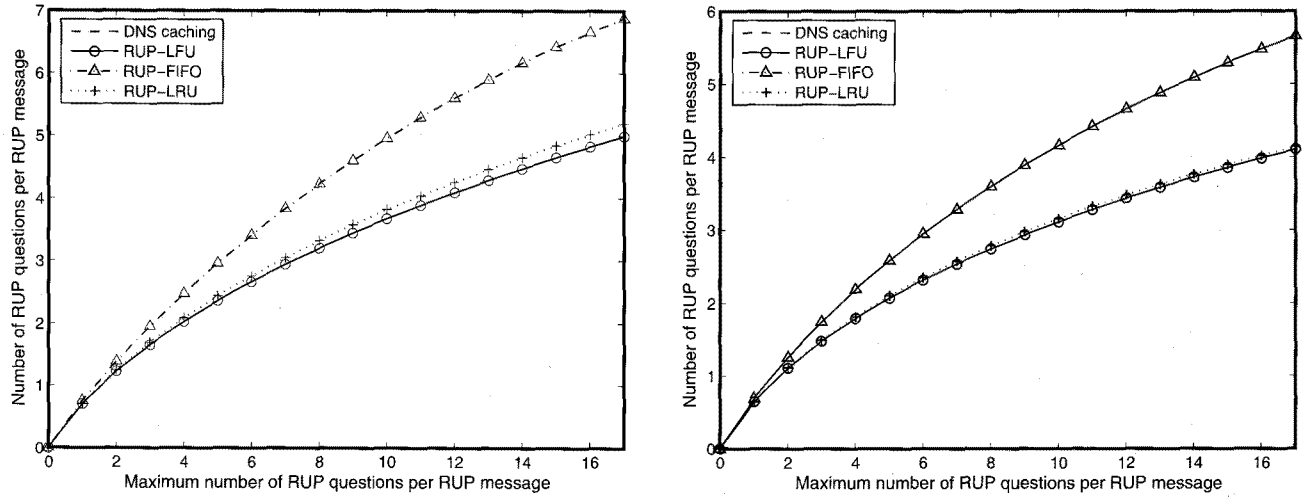


Fig. 5. Average number of RUP questions per RUP message as a function of the maximum number of RUP questions per RUP message: (a) KAIST DNS trace and (b) Yonsei university DNS trace.

RUP-LFU often requires more bytes for message transmission than the DNS cache policy. However, its byte consumptions are not more than two times of the DNS cache policy's at most of the TTL values. Moreover, they are smaller than the DNS cache policy's at 10,000 seconds or more TTL values. RUP-LFU piggybacks on average the less than ten RUP questions per RUP message at most of the TTL values.

In summary, we believe the similarity of results for the two large, independent traces proves the effectiveness of our proposed scheme. Among RUP policies, RUP-LFU shows the best performance for all evaluation metrics. As a result, we suggest RUP-LFU as a solution to renewal.

V. RENEWAL USING PIGGYBACK VS. RENEWAL

In this section, we compare the best RUP policy, RUP-LFU, with the best renewal policy, renewal-LFU [2]. We perform a trace-driven simulation using the traces in Table 1.

A. *Evaluation Model*

Here we describe the parameters used and the evaluation metrics.

A.1 Parameter Used

To our evaluation parameter, we exploit the relative increase of queries used as the parameter in the evaluation of renewal policies [2]. However, we modify it to accommodate both RUP-LFU and renewal-LFU. This is because RUP differs from renewal in terms of cache update operations; while renewal refreshes expired cache records by issuing additional DNS queries, RUP does them by piggybacking additional RUP questions onto solicited DNS queries. Here we call by a request, a regular DNS query issued to answer to the DNS resolution, an

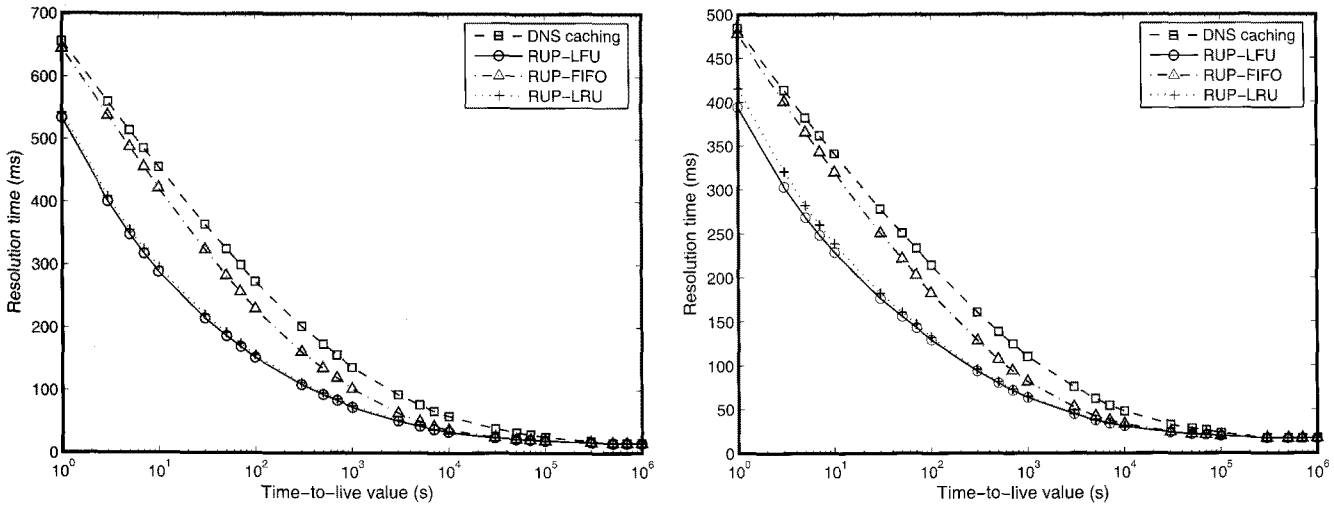


Fig. 6. Average resolution time as a function of TTL value: (a) KAIST DNS trace and (b) Yonsei university DNS trace.

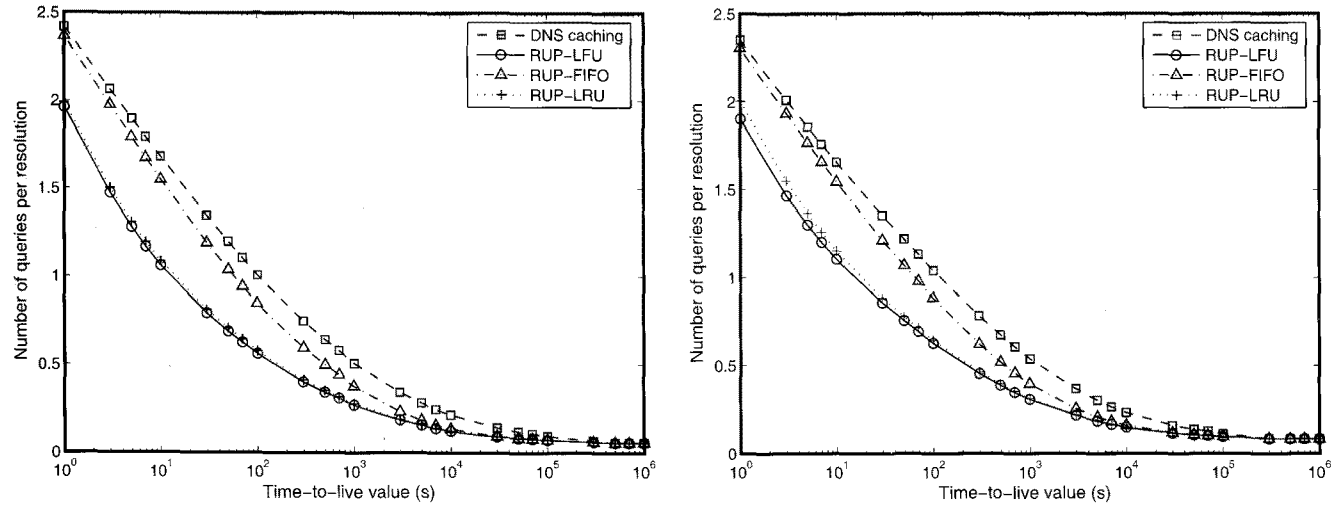


Fig. 7. Average number of queries as a function of TTL value: (a) KAIST DNS trace and (b) Yonsei university DNS trace.

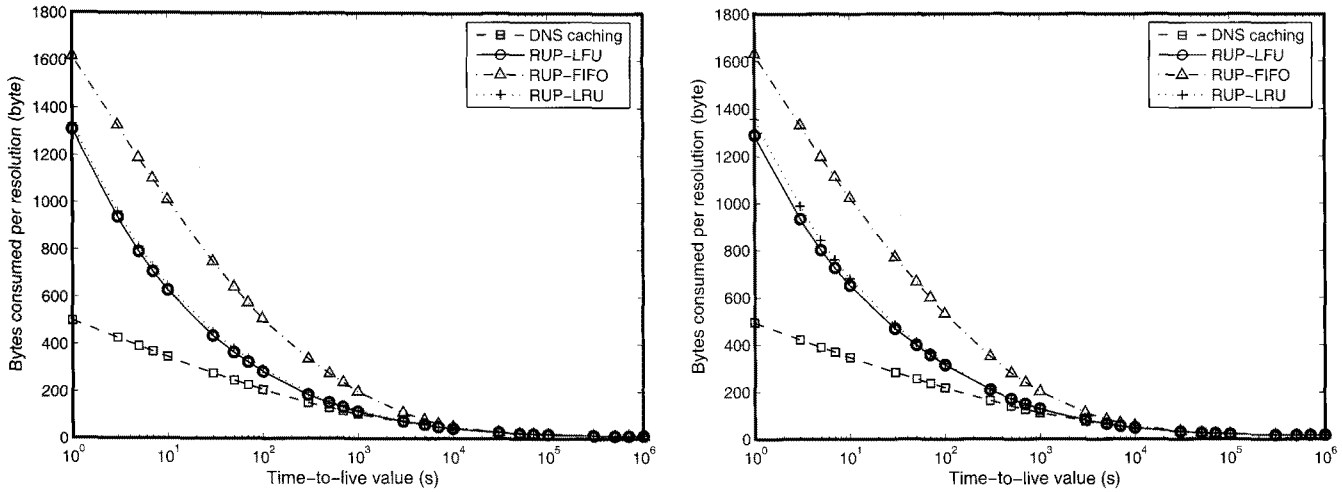


Fig. 8. Average number of bytes consumed as a function of TTL value: (a) KAIST DNS trace and (b) Yonsei university DNS trace.

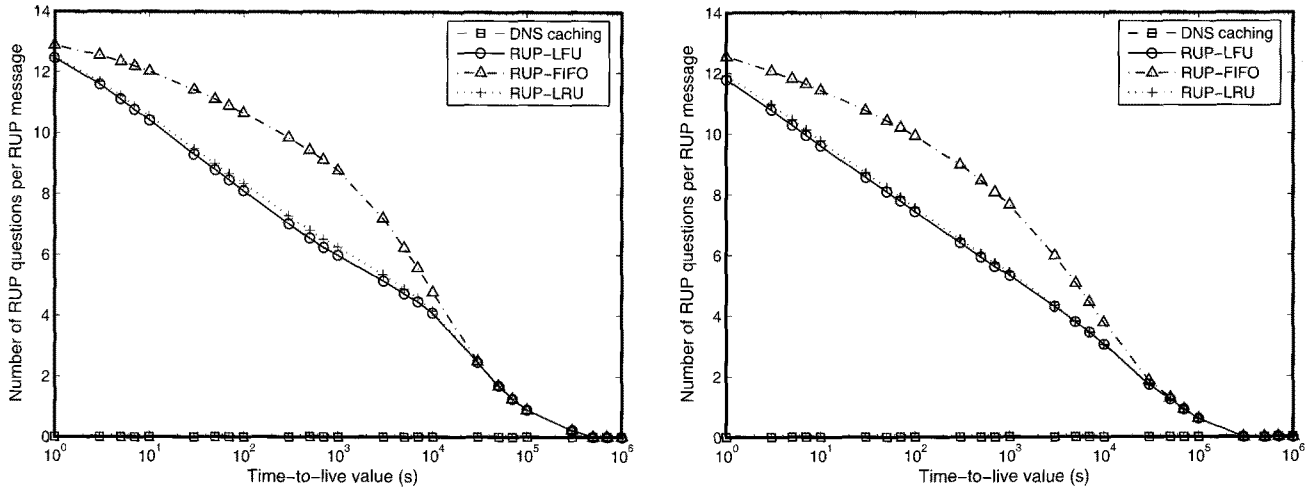


Fig. 9. Average number of RUP questions per RUP message as a function of TTL value: (a) KAIST DNS trace and (b) Yonsei university DNS trace.

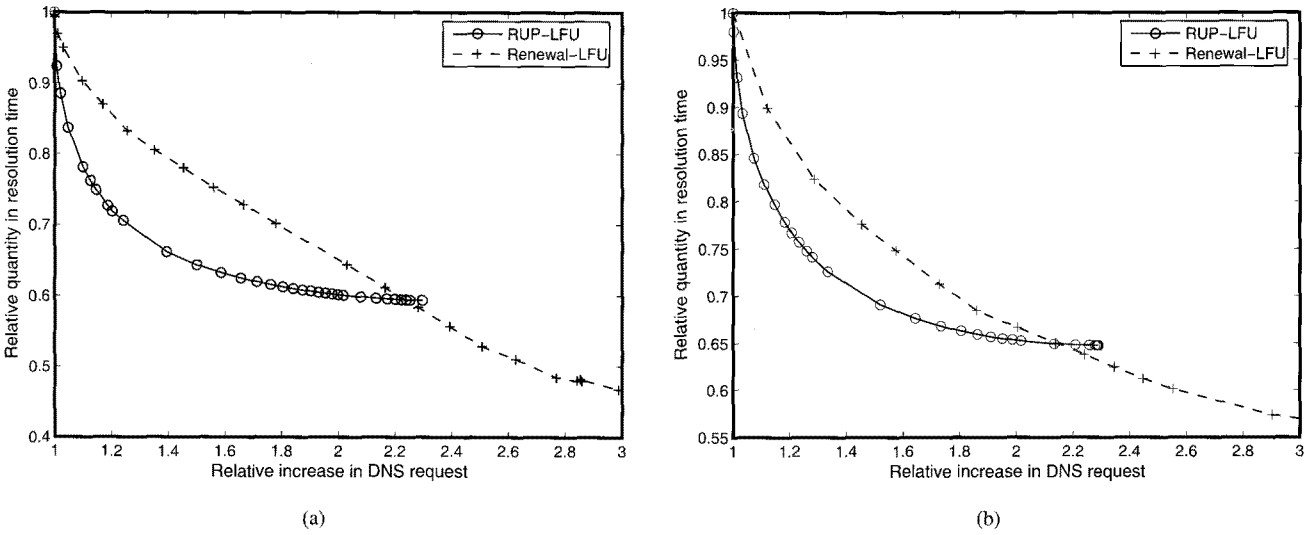


Fig. 10. Relative decreases in resolution times as a function of relative increase in DNS requests: (a) KAIST DNS trace and (b) Yonsei university DNS trace.

additional DNS query generated to refresh the expired cached record in renewal, and an RUP question piggybacked onto the DNS query to refresh the expired cached record in RUP. As a result, we use the relative increase in requests instead of the relative increase in queries as our evaluation parameter. We normalize the number of requests for the DNS cache policy to one. The number of requests for RUP-LFU and renewal-LFU is measured relative to the normalized one for the DNS cache policy.

A.2 Evaluation Metrics

We use the relative quantity in resolution time, the relative quantity in DNS messages (DNS queries and responses), and the relative quantity in bytes consumed for message transmissions as our evaluation metrics. We do not consider the processing overhead at the DNS server as our evaluation metric because we assume that the processing overhead of a RUP question by RUP at the DNS server is similar to that of an additional DNS query by renewal. As above, we normalize the performance of the DNS cache policy on each evaluation metric to one and that of the other two is measured relative to it. We also compare RUP with renewal in terms of deployment overheads.

B. Evaluation Results

Fig. 10 presents the distributions of relative quantities in resolution times, varying relative increases in requests for our traces. Relative quantities in resolution times of RUP-LFU are smaller than those of renewal-LFU, with request overheads of 1.0 to about 2.2 for both our traces. This is because it may require several requests, that is, DNS queries to multiple ADNS in renewal, to refresh an expired cached record, while it needs only a request, that is, an RUP question to an ADNS in RUP. For our traces, renewal-LFU issues on average 1.33 DNS queries to refresh an expired cached record. For the KAIST trace, RUP-LFU and renewal-LFU reduce the resolution time of the DNS cache

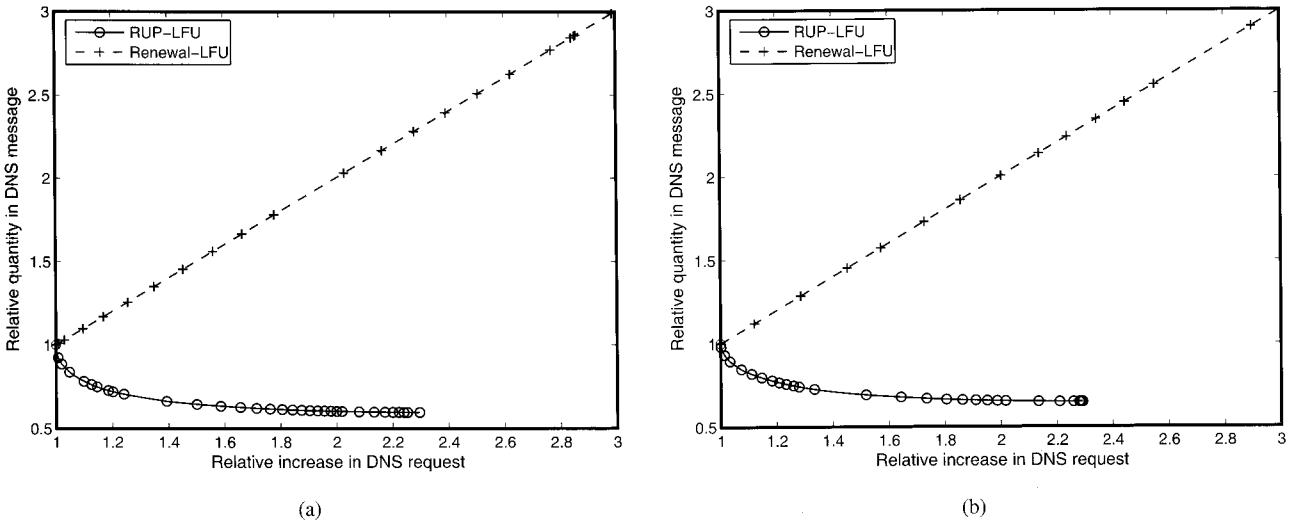


Fig. 11. Relative decreases in DNS messages as a function of relative increase in DNS requests: (a) KAIST DNS trace and (b) Yonsei university DNS trace.

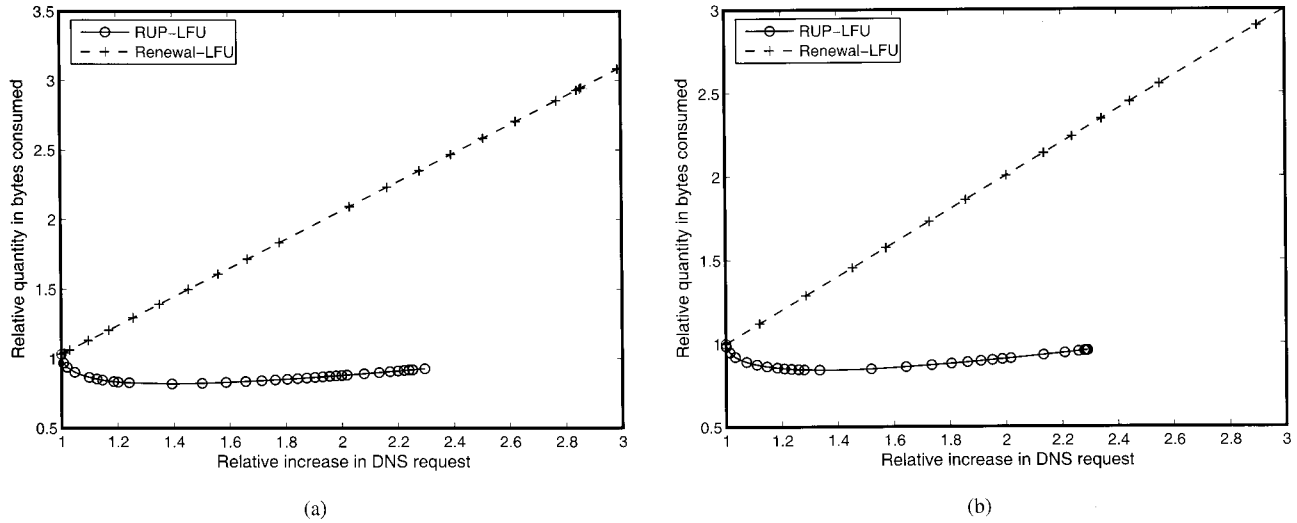


Fig. 12. Relative decreases in bytes transmitted for resolution as a function of relative increase in DNS requests: (a) KAIST DNS trace and (b) Yonsei university DNS trace.

policy by 21.9% and 9.6%, respectively, with a request overhead of 1.1. That is, RUP-LFU decreases the resolution time of the DNS cache policy as much as 2.3 times (21.9%/9.6%) with a request overhead of 1.1, compared with renewal-LFU. It reduces the resolution time of the DNS cache policy as much as 1.8 and 1.4 times, with the respective request overheads of 1.3 and 1.7, compared with renewal-LFU. For the Yonsei trace, RUP-LFU decreases them as much as 1.8, 1.5, and 1.2 times with each request overhead, respectively, compared with renewal-LFU. However, the performances of RUP-LFU and renewal-LFU are converged at the request overhead of about 2.2 for both our traces. This is because RUP-LFU chooses the records to be refreshed among cached records indexed by an ADNS, while renewal-LFU chooses them among overall cached records, and the usability of cached records refreshed by RUP-LFU deteriorates more than that of cached records refreshed by renewal-LFU as the request overhead increases. For our traces, RUP-LFU cannot issue requests more than the request overhead of about 2.3, though renewal-LFU can. The first reason for this is that RUP piggybacks RUP questions onto only solicited DNS query messages. The second reason is that the size of an RUP message is limited to 512 bytes.

Fig. 11 presents the relative quantities in DNS messages, varying relative increases in requests for our traces. RUP-LFU reduces overall DNS messages by increasing the performance of the DNS cache. However, renewal-LFU increases them because it issues many additional DNS messages to refresh expired cached records, though it improves the performances of DNS caches like RUP-LFU. Specifically, RUP-LFU reduces DNS messages maximum as much as 40.7% for the KAIST trace and by 35.1% for the Yonsei trace, with a request overhead of about 2.2. However, renewal-LFU increases the number of DNS messages in direct proportion to request overheads for both our traces, and such increases of DNS messages must significantly enlarge the overall Internet traffic in terms of flow because DNS messages already takes huge portion of the whole Internet traf-

fic. Today, DNS messages take as much as 10% to 40% of the overall Internet traffic in terms of flow.

Fig. 12 presents the relative quantities in bytes consumed, varying the relative increase in requests for our traces. RUP-LFU also reduces the bytes consumed for message transmissions by the DNS cache policy, while renewal-LFU increases them. The first reason is that an RUP question piggybacked by RUP has a much lighter weight than an additional DNS query issued by renewal in terms of byte consumption. Specifically, each RUP question and answer require about 18.5 bytes, while a DNS query does about 33.2 bytes, and a DNS response does about 173.0 bytes for our traces. That is, the weight of an RUP question needs only 18.0% ((18.5+18.5) / (33.2+173.0)) of that of an additional DNS query in terms of byte consumption. The second reason is that, as we mentioned above, renewal requires more requests to refresh an expired cached record than RUP. As a result, RUP-LFU decreases the bytes consumed for message transmissions by the DNS cache policy as much as about 20.7% at the request overhead of 1.39 for the KAIST trace and about 16.5% at the request overhead of 1.32 for the Yonsei trace, while renewal-LFU increases in direct proportion to request overheads for both our traces.

For deployment, RUP requires changes on requesting and requested DNS servers because DNS servers requesting an RUP message should be able to piggyback RUP questions onto DNS queries and requested DNS servers must be able to recognize and respond RUP questions, while renewal needs only a change on requesting DNS servers because it uses normal DNS queries and response messages to refresh cached records at their expiration times. However, RUP is interoperable with the current DNS because it exploits unused fields of the existing DNS message format to piggyback expired cached records.

VI. CONCLUSION

With domain name server (DNS) resolutions being recognized as a critical performance bottleneck of Internet name-based applications such as the World Wide Web, several approaches to addressing this problem have been explored. Although contributing to reducing DNS resolution time, they incur additional DNS messages. In this paper, we propose a scheme named renewal using piggyback (RUP) to reduce DNS resolution time without incurring additional DNS messages. RUP refreshes expired cached records by simply piggybacking them onto solicited DNS queries instead of issuing additional DNS queries. Simulation results based on two large independent DNS traces show that our best RUP policy much reduces not only the DNS resolution time but also the DNS traffic. Moreover, such improvements can be ensured regardless of the variation of time-to-live (TTL) value. In comparison with renewal, RUP reduces the DNS resolution time more efficiently. It also remarkably decreases the DNS traffic, while renewal increases in proportion to the number of refreshed records.

ACKNOWLEDGMENTS

We thank Taejin Yun and Kwangseok Kim, who work on the network infrastructure at KAIST, for helping to collect data on DNS traffic on KAIST's DNS server. Likewise, we also really appreciate Minhee Jun, who runs the DNS server at Yonsei university, for helping to collect data on DNS traffic on Yonsei's DNS server.

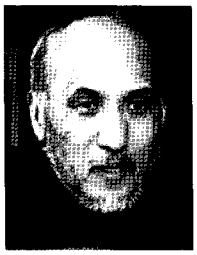
REFERENCES

[1] E. Cohen and H. Kaplan, "Prefetching the means for document transfer: A new approach for reducing web latency," *Comput. Netw.*, vol. 39, pp. 437–455, 2002.
[2] E. Cohen and H. Kaplan, "Proactive caching of DNS records: Addressing a performance bottleneck," *Comput. Netw.*, vol. 41, pp. 707–726, 2003.
[3] B. Krishnamurthy, R. Liston, and M. Rabinovich, "DEW: DNS-enhanced web for faster content delivery," in *Proc. World Wide Web*, May 2003.
[4] C. Wills and H. Shang, "The contribution of DNS lookup costs to web object retrieval," Worcester Polytechnic Institute Tech. Rep. TR-00-12, July 2000.
[5] J. Jung, H. B. E. Sit, and R. Morris, "DNS performance and the effectiveness of caching," *IEEE/ACM Trans. Netw.*, Oct. 2002.
[6] P. Mockapetris, Domain names-concepts and facilities, *RFC 1034*, Nov. 1987.
[7] P. Mockapetris, Domain names-implementation and specification, *RFC 1035*, Nov. 1987.
[8] P. Mockapetris and K. Dunlap, "Development of the domain name system," in *Proc. ACM SIGCOMM*, 1988, pp. 123–133.
[9] P. Albitz and C. Liu, *DNS and BIND*, 4th ed., O'Reilly , 2001.
[10] A. Snoeren and H. Balakrishnan, An end-to-end approach to host mobility. in *Proc. ACM MOBICOM*, Aug. 2000.
[11] B. Krishnamurthy and C. Wills, "Study of piggyback cache validation for proxy caches in the world wide web," in *Proc. USENIX Symp. Internet Technologies and Systems*, Dec. 1997.
[12] E. Cohen, B. Krishmurthy, and J. Rexford, "Improving end-to-end performance of the web using server volumes and proxy filters,". *Comput. Commun. Rev.*, vol. 28, pp. 241–253, 1998.
[13] A. Dan and D. Twosley, "An approximate analysis of the LRU and FIFO buffer replacement schemes," in *Proc. ACM SIGMETRICS*, 1990.
[14] M. Milenkovic, *Operating systems concepts and design*, 2nd ed., McGraw-Hill, 1992.
[15] A. Shaikh, R. Tewari, and M. Agrawal, "On the effectiveness of DNS-based server selection," in *Proc. IEEE INFOCOM*, Aug. 2001.
[16] P. Vixie, Extension mechanisms for DNS, *RFC 2671*, Aug. 1999.
[17] V.Jacobson, C.Leres, and S. McCanne. *Tcpdump*. ver. 3.6.1. [Online]. Available: http://www.tcpdump.org/

Beakcheol Jang is a Ph.D. student in Department of Computer Science at North Carolina State University, Raleigh, USA. His research interests include Internet Application such as Web and DNS, Traffic analysis emphasis on application performance, and Wireless and Mobile Systems. He received his B.S. in Computer Science from Yonsei University, Seoul, South Korea in 2001, and his M.S. in Computer Science from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea in 2002.

Dongman Lee is a Professor in Department of Computer Science at Korea Advanced Institute of Science and Technology IT Convergence Campus (KAIST ICC), Daejeon, South Korea. From 1988 to 1997, he worked as Technical Contributor at Hewlett-Packard. He has been actively participating in Korean and international Internet number and name committees since 1998. He received a Prime Minister Award as the recognition on the advancement of the Korean Internet in 2000. His laboratory, Collaborative Distributed Systems and Networks Lab, was appointed as National Research Laboratory in 2001 and granted 0.9 M US dollars. He has published more than 50 papers in int'l journals and conference proceedings. He has served as a member of the program committee of several international conferences such as IEEE Multimedia, PDCS, PRDC, VSMM, etc. His research interests include distributed systems, computer networks, mobile computing, and

pervasive computing. He is a member of IEEE and ACM. He received the BS in Computer Engineering from Seoul National University, South Korea in 1982, and the MS and Ph.D. in Computer Science from KAIST, Daejeon, South Korea in 1984 and 1987, respectively.



Kilnam Chon is a Professor in Department of Computer Science at KAIST, Daejeon, South Korea. He has special interests in system architecture, including computer networking, distributed processing, and information systems. He has worked as a principal investigator of a national project on workstation development, intelligent processing computer development. He received his MS in computer science and Ph.D. in Systems Engineering from UCLA in 1967 and 1974, respectively.

Hyunchul Kim is a BK Assistant Professor in School of Computer Science and Engineering at Seoul National University, Seoul, South Korea. He worked as a visiting scholar in the Cooperative Association for Internet Data Analysis (CAIDA) and as a Research Professor in School of Engineering at ICU, Daejeon, South Korea 2005 to 2006. His research interests include web caching and replication, scalable distributed storage infrastructure, network traffic measurement, and peer-to-peer networking. He received his B.S., M.S., and Ph.D. in Department of Computer Science from KAIST in 1995, 1997, and 2005, respectively.