A Hybrid Web Browser Architecture for Mobile Devices

Junguk CHO[1], Euiseong SEO[2], Jinkyu JEONG[2]
[1]*School of Computing, University of Utah, UT 84112, USA*
[2]*Sungkyunkwan University, Gyeonggi-do 440-746, Republic of Korea*
** Corresponding author: jinkyu@skku.edu*

***Abstract*—Web browsing on mobile networks is slow in comparison to wired or Wi-Fi networks. Particularly, the connection establishment phase including DNS lookups and TCP handshakes takes a long time on mobile networks due to its long round-trip latency. In this paper, we propose a novel web browser architecture that aims to improve mobile web browsing performance. Our approach delegates the connection establishment and HTTP header field delivery tasks to a dedicated proxy server located at the joint point between the WAN and mobile network. Since the traffic for the connection establishment and HTTP header fields delivery passes only through the WAN between the proxy and web servers, our approach significantly reduces both the number and size of packets on the mobile network. Our evaluation showed that the proposed scheme reduces the number of mobile network packets by up to 42% and, consequently, the average page loading time is shortened by up to 52%.**

***Index Terms*—Browsers, Internet, Mobile computing, Web services, World Wide Web.**

I. INTRODUCTION

The data traffic generated by web browsing accounts for the second largest portion of mobile network traffic [1]. Mobile web browser users, however, experience slower web loading time in mobile network environments than in conventional networks such as wired or Wi-Fi networks [2].

In mobile network environments, low bandwidth, long round-trip time (RTT) and radio signal variation may aggravate the web loading delay. Since many web content providers are changing their web pages to become mobile-friendly, with sizes between 100 to 200 kbytes, the significance of the low bandwidth, however, is gradually shrinking. In addition, the signal strength is considered to be negligible to TCP downlink performance as long as it is above a threshold [4]. Consequently, lots of researchers have claimed that the long RTT of a mobile network is the most significant inhibitor of fast mobile web browsing [3-4].

A web page usually has many embedded objects that reside at other web sites. Loading a web object involves a domain name system (DNS) lookup operation and a transport control protocol (TCP) handshake during the connection establishment phase between the mobile web browser and the web server. The size of packets for DNS lookups and TCP handshakes is generally small. However, transferring them over a mobile network requires significant time due to the long RTT. As a result, such frequent small-packet communications slow down the overall web page loading time on mobile networks.

Meanwhile, according to the HTTP standard [6], many web browsers embed additional header fields in HTTP requests, such as Accept-Language, Accept-Charset, and User-Agent. These fields are useful for web servers since they can make better decision for responding the requests in accordance with the content of the fields. But, most of the header field data does not change frequently since these fields represent the web browser. These header fields, however, occupy precious mobile network bandwidth and burden the load on the mobile network.

In this paper, we propose a novel mobile web browser architecture that aims to reduce the page loading delay over a mobile network. The proposed architecture consists of a mobile web browser and its corresponding proxy server. The proxy server resides at the joint point between the wide-area network (WAN) and the mobile network. The mobile browser stays connected to the proxy server with a persistent connection, which is defined in the hypertext transfer protocol (HTTP) 1.1 standard.

The mobile browser in our approach delegates the connection establishment tasks to the proxy server. The mobile browser no longer generates TCP handshakes and DNS lookups once the connection between the mobile browser and the proxy server is established, and the traffic for the connection establishment phase passes only through the high-speed WAN link between the proxy and web servers. Consequently, the time delay caused by TCP handshakes and DNS lookups over the mobile network is eliminated from the web page loading time.

In addition, all HTTP requests from the mobile web browser are forwarded to the proxy server and, in turn, the proxy requests the web pages from web servers on behalf of the mobile browser. Therefore, inclusion of the header fields in the requests at the browser end is unnecessary in the proposed architecture because the header fields can later be appended by the proxy server. Appending the header fields at the proxy server greatly reduces the size of the request travelling through the mobile network. In addition, this approach greatly leverages the advantage of HTTP pipelining because the smaller a request is, the more requests a packet delivers.

We evaluated the proposed scheme with the prototype implementation. The prototype mobile web browser is built upon the built-in web browser of Android, which is an open-source mobile operating system and a simple proxy server is implemented to be paired with the prototype mobile web browser. The evaluation was conducted with various access

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2013R1A6A3A01023894) and by the IT R&D program of MKE/KEIT [10041244, SmartTV 2.0 Software Platform].

Digital Object Identifier 10.4316/AECE.2014.03001

patterns over real world web pages from the five most popular web sites [7-8].

The rest of the paper is organized as follows. Section 2 introduces technical background about web browsing on mobile networks and the motivation for this research. Section 3 presents our mobile web browser architecture, and Section 4 analyzes the effectiveness of the prototype implementation of the proposed scheme. After introducing related work in Section 5, we conclude this paper in Section 6.

II. BACKGROUND AND MOTIVATION

In this section, we describe the web browsing process from establishing connections to rendering a complete web page. We also associate network characteristics with the web browsing process from the perspective of web loading performance. Then, we depict the motivation of this work by demonstrating the adverse effect of long RTT in the mobile network on web loading performance. Finally, we introduce previous web browsing architectures and their pros and cons.

A. Background

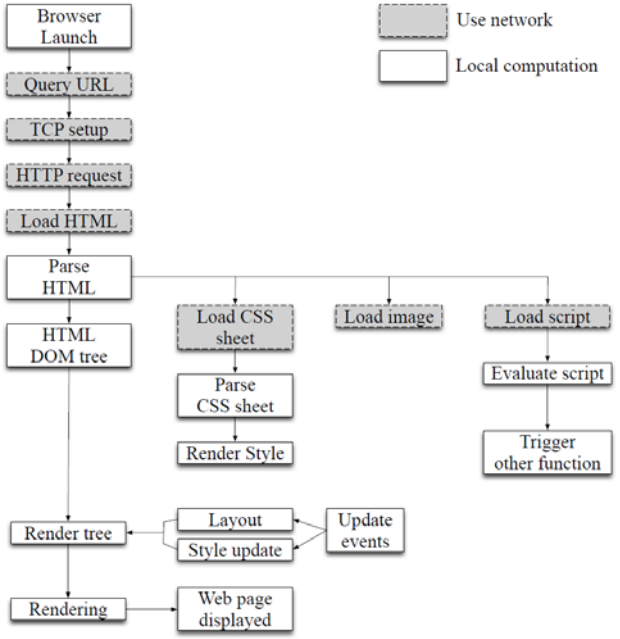


Figure 1. Workflow of a web browser [9].

Fig. 1 shows the internal process of a web browser while it is loading a web page [9]. The work flow begins when a user inputs the uniform resource locator (URL) of a web page in the browser window and pushes the load button of the browser. The browser first sends a DNS query to resolve the IP address of the URL. Then, the browser establishes a TCP connection to the web server through three-way handshaking. Both the DNS query and TCP three-way handshaking are conducted in the connection establish phase.

After the browser establishes a TCP connection with the web server, the browser requests the web page from the server, and in turn receives the web page as the response from the server. The browser parses the web page and generates a document object model (DOM) tree, which stores the information on the structure and data on the web page.

A web page may have multiple embedded web objects including JavaScripts, cascading style sheets (CSSs) and images. Such embedded objects are sometimes located in other remote web sites. The browser makes a request to an appropriate web server when it finds an embedded web object. Each web request for embedded objects is followed by a new connection establishment phase when the browser has no open connection with a corresponding web server. The browser starts rendering of a web page when a sufficient number of its embedded objects are fetched. When all embedded objects are fetched and rendering of the web page is complete, the web page loading instance finishes [9].

The components in the loading process can be classified into two classes: network operations and local computations. The gray boxes with dotted lines in Fig. 1 denote network operations, and the white boxes represent local computations.

Naturally, the network operations become performance bottlenecks when the underlying network has low bandwidth or long RTT whereas the performance of the local computation operations is bounded by the CPU performance. Lots of previous studies claimed that there are two possible sources of performance bottlenecks for web page loading operations over mobile network. One is poor network performance [3-5], and the other is limited CPU capability [10-13]. However, the improvement in web page loading time from reducing the amount of computation is marginal in current smartphone systems [4].

Although the performance of a mobile network is determined by many factors such as its network bandwidth, RTT and radio signal strength, improving the bandwidth of an existing mobile network does not improve the browser delay significantly [3-4]. Moreover, TCP downlink throughput is not affected by the signal strength if it is above some threshold [3]. Accordingly, the RTT of the network is considered the most significant contributor to web browsing performance over a mobile network [3-4].

B. Motivation

TABLE I. AVERAGE NUMBER OF PACKETS/SIZE OF DATA (IN KBYTES) FOR LOADING MOBILE WEBSITES. THE NUMBER OF PACKETS AND THE AMOUNT OF DATA FOR THE HTTP RESPONSE IS OMITTED

Web Sites	Whole Page	DNS Queries	TCP Handshakes
	TCP Closes	HTTP Requests	Avg. Conn. Reuse Counts
http://google.co.kr	528.7/372.8	12.0/2.0	25.0/1.7
	33.3/2.1	15.4/10.0	7.1
http://xhtml.weather.com	80.0/30.2	6.0/1.3	9.5/0.7
	12.7/0.8	9.0/5.9	5.8
http://cnnmobile.com	433.7/209.0	11.1/2.3	57.0/4.0
	75.9/4.9	27.2/18.2	8.2
http://m.facebook.com	157.8/76.2	8.1/1.6	17.7/1.2
	23.4/1.5	8.8/5.1	2.9
http://en.m.wikipedia.org	216.0/95.7	6.0/0.8	27.0/2.1
	36.0/2.3	13.7/7.8	4.7
http://m.espn.go.com	908.7/556.8	30.6/6.3	72.8/5.1
	96.7/6.2	40.5/25.7	16.2

In order to reveal the proportion of connection establishment phases to the total network transmission for web page loading, we analyzed the number of packets and size of data transferred to load each of the six most popular mobile web sites [7-8]. Table I shows the measured data and classifies it into packets and data for DNS lookups, TCP handshakes, TCP closes and HTTP requests. It also includes

the number of connection reuses for an instance.

According to our measurement, the packets for the connection establishment operations (TCP handshakes and DNS lookups) take 12% of the total transferred packets. However, the amount of data needed for the connection establishment operations is negligible. Consequently, we can conclude that the size of a packet for the connection establishment operations is small in comparison to that for web object transmission. Because the RTT of the mobile network is slow, the time to finish the connection establishment phase is determined by the number of transferred packets rather than the amount of transferred data. Thus it is expected that the connection establishment phase significantly affects the overall web page loading time a over mobile network due to its slow RTT. Also, the proportion of packets for the connection establishment phase is expected to increase as the amount of cached data in the local storage increases. When many cached objects are hit, the number of packets for loading a web page decreases and the portion of the connection packets relatively increases.

To lessen the connection establishment overhead, the HTTP 1.1 standard introduced the HTTP persistent connection and HTTP pipelining. These two schemes are currently provided by most commercial web browsers. The HTTP persistent connection is a technique to keep a TCP connection open after finishing an HTTP transaction, and to reuse the open connection for the forthcoming HTTP transactions instead of opening new connections. By default, all connections are regarded as persistent under HTTP 1.1 and as non-persistent under HTTP 1.0. Depending on the value of the connection header field, the web server closes or maintains the connection after servicing the HTTP request.

The built-in web browser of Android supports the HTTP persistent connection like many others. The browser stores an open persistent connection in the connection cache after finishing the HTTP transaction if the used HTTP connection supports the persistent connection (i.e., HTTP 1.1 or HTTP 1.0 with keep-alive). Otherwise, the browser closes the connection. If a connection to a web server is necessary for a new HTTP request, and there is a cached connection to the web server in the connection cache, the browser reuses the cached connection to avoid the connection establishment overhead.

However, the number of reused connections is less than half of the total number of HTTP requests according to our analysis as shown in Table I. This is because a web page usually has multiple embedded objects that are provided by different web sites and new connections to the web sites are necessary to access the embedded objects. In addition, the limited capacity of the connection cache forces the connection cache to discard cached connections that may be reused in the near future. Even if the capacity of the browser connection cache is unlimited, web servers may terminate the open persistent connections when they become idle [14]. Accordingly, the effect of the persistent connection is limited in practice, and newly established connections struggle with the long RTT of the mobile network.

Meanwhile, HTTP pipelining enables web browsers to issue HTTP requests without waiting for responses to the previous requests. With HTTP pipelining, a web browser packs multiple requests in a single TCP packet. This may greatly reduce the number of packets and the number of new connections for requesting web objects in the same web server. HTTP pipelining also benefits from thread-level parallelism at a server side since it overlaps processing of simultaneous multiple requests.

Although HTTP pipelining is effective in improving the web page loading delay, its actual effectiveness is affected by two reasons as shown in Fig. 2.

First, the maximum transmission unit (MTU) size of the underlying network limits the number of HTTP requests packed in a single TCP packet. For example, the average size of an HTTP request in Table I is approximately 650 bytes while the MTU size of a TCP packet is 1,500 bytes. Therefore, only two or three HTTP requests are allowed to be delivered by a TCP packet as depicted in Fig. 2(a).

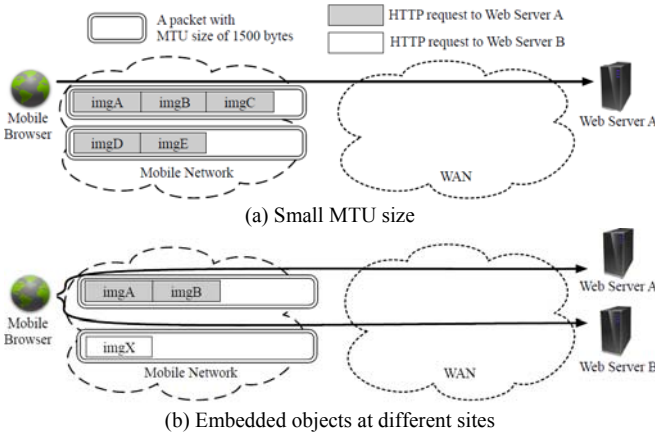


Figure 2. Disturbance factors of HTTP pipelining

Second, HTTP pipelining is not applicable when embedded web objects exist at multiple web sites because HTTP pipelining pipelines multiple requests in a single TCP connection. Accordingly, there are three embedded objects in a web page and if their hosts are different, HTTP requests for them cannot be packed in a single TCP packet as shown in Fig. 2(b).

Considering that the RTT of a mobile network is quite long, reducing the number of packets by exploiting the two optimization techniques, persistent connection and request pipelining, is crucial for improving web page loading time. In this paper, we propose a web browser architecture that minimizes the number of packets to load a web page by maximizing the benefits from these two optimization techniques.

C. Existing Mobile Web Browsers

In order to resolve the slow web page loading issue on a mobile network, a few research results have been reported. Many of them fall into two categories: thin-client architecture and native browser architecture [15].

A thin-client web browser consists of a light-weight web browser and a remote proxy server. A request for a web page is forwarded to the proxy server by the browser, and then the proxy fetches the web objects for the page, renders the web page image, and sends the rendered image back to the web browser. The browser is only responsible for forwarding user inputs to the proxy and for displaying the rendered web page images delivered from the proxy. This

architecture was initially suggested to overcome the limited computing power of mobile devices [10-13]. In addition to this, the thin-client scheme also lessens the web page loading delay from the slow RTT to some degree since the proxy server is in charge of establishing connections and retrieving the embedded web objects.

However, the thin-client model has three critical drawbacks. First, some web objects may not be properly rendered, and dynamic web pages may not function correctly because what the browser shows to users are simply rendered images or preprocessed web pages [15]. For example, an interactive feature of a web page written in a script language will not react to user inputs because the browser is ignorant of the existence of the script code. Moreover, reloading or zooming in/out a part of a web page is difficult to handle for the same reason. The interactivity of mobile applications is important in the future internet [2]. Accordingly, the lack on supporting the interactivity can be an obstacle to a better web browsing experience on the mobile network.

Second, although it may reduce the number of small packets, the thin-client architecture sometimes increases the amount of data transferred over the mobile network. According to experiments in related studies, the mobile network data generated by a thin-client browser are typically larger than those generated by a conventional web browser [11-12], depending on the web page characteristics [13]. This mobile network traffic increase stems from the lack of a local cache for storing web objects and the size of the rendered web page images.

Finally, this approach increases burdens on the proxy server. If the proxy server is overloaded it could adversely affect the web browsing performance [15]. Because the proxy is in charge of fetching web objects and rendering web pages on behalf of its clients, the computation load to the proxy is much heavier than the load on conventional web proxy servers.

As the hardware performance of mobile devices rose to a level sufficient to execute high-performance web browsing engines (e.g., WebKit), many commodity mobile devices chose to employ native web browsers. The native browser handles every step of processing a web page request by itself in order to provide an interactive and dynamic browsing experience similar to that on a PC. Their interface, including zoom, feels more natural and their page loading time feels faster than their thin-client counterparts because they draw intermediate renderings during page loading and update the page being loaded continually [16].

Rapid improvements in processor performance together with the spread of mobile-device-oriented web pages are boosting the popularity of native web browsers in mobile devices. However, as mentioned in Section 2.2, native browsers generate a large number of small packets during the connection establishment phases, and this severely limits the page loading time over a mobile network where the RTT is exceedingly slow.

The thin-client-based approach can reduce the number of connection establishment phases over the mobile network. In addition, the native browser is capable of providing an outstanding user experience. Since both architectures have different advantages, we propose a new browser architecture that hybridizes both architectures.

III. TWO-TIER BROWSER ARCHITECTURE

In this paper, we propose a two-tier browser architecture (TwoB) which minimizes web page loading time on a mobile network. TwoB consists of a native mobile browser on a mobile device and a proxy server on the border between the WAN and the mobile network. The browser processes all computations ranging from requesting the content of a web page to rendering the page, and provides user interaction via touch capabilities. Accordingly, end users can experience natural interaction with interactive features in web pages. The proxy server is an intermediary for delivering request/response packets between the browser and web servers, and also processes the connection establishment phase on behalf of the browser. By eliminating the connection establishment phases for all web requests, web browsers do not need to wait for time-consuming connection packet transfers on a mobile network. In addition, from the nature of this request/response relay, our proxy server can be lightly loaded and handle more clients than the thin-client model does. The detailed architecture of TwoB is depicted in Fig. 3.

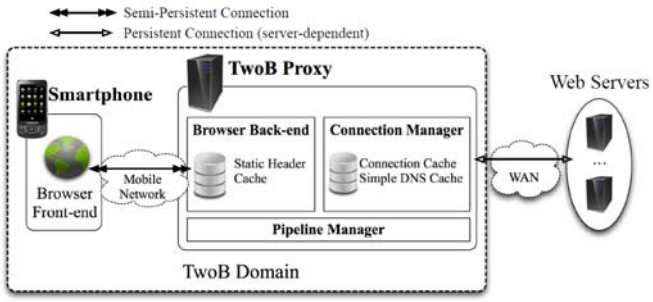


Figure 3. TwoB Architecture

A. Overview

The proxy server in TwoB (denoted as the TwoB proxy) consists of a browser back-end, a pipeline manager and a connection manager. The browser back-end is a counterpart of each mobile browser and relays web requests generated in each mobile browser. Its main role is to maintain a persistent connection between itself and a mobile browser. In addition, it manages the static header field cache in order to reduce the sizes of HTTP requests passing through the mobile network; the use of a static header field cache is illustrated in Section 3.4.

The connection manager manages connections between the proxy server and web servers. It maintains a TCP connection pool for web servers and provides connection sharing for multiple mobile clients which are connected with this proxy server. In addition, it provides a simple DNS cache in order to reduce duplicate DNS queries when multiple clients are requesting web pages from a small set of web servers.

Finally, the pipeline manager is used to multiplex HTTP requests between the browser back-end and the connection manager. Its main role is to keep the sequence of responses the same as the sequence of requests. Since our scheme reduces the size of HTTP requests passing over the mobile network, the number of packable HTTP requests in a single

TCP packet is different between the mobile network and the WAN. In addition, requests which are destined for different web servers arrive at the same connection in the proxy. Accordingly, the pipeline manager arbitrates HTTP pipelining between the mobile network and the WAN.

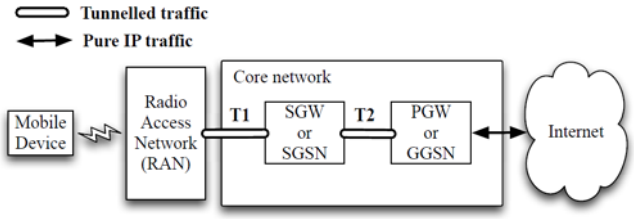


Figure 4. Simplified mobile network architecture

When we deploy the TwoB proxy server in mobile network, we should consider location of it. Fig. 4 shows current mobile network architecture. Mobile network generally uses GPRS tunneling protocol (GTP) not only to transfer data packets but also to support handover and QoS guarantee per user. These tunnels are depicted as T1 and T2 in Fig. 4. When a mobile device transmits a packet, the packet is first encapsulated by radio access network (RAN) and decapsulated by serving gateway (SGW) or serving GPRS support node (SGSN). The packet is encapsulated again by SGW or SGSN and decapsulated by packet data network gateway (PGW) or gateway GPRS support node (GGSN). Finally, the packet is forwarded to the internet. Considering the GTP of mobile network, when our proxy server resides in the middle of such tunnels, the packet should require additional encapsulation and decapsulation because the proxy works with decapsulated packets. In this regard, the gateway of mobile network (e.g., PGW or GGSN) is reasonable location of our TwoB proxy server without the burden of additional packet en/decapsulation.

The following subsections explain the details of our scheme from setting up the browser architecture to handling web requests.

B. TwoB Initialization

The main advantage of our scheme is that our proposed browser architecture does not need a full redesign of the previous web browsing environment. Instead, our scheme slightly modifies the browser in mobile clients in order to make it specific to the proxy server in our architecture.

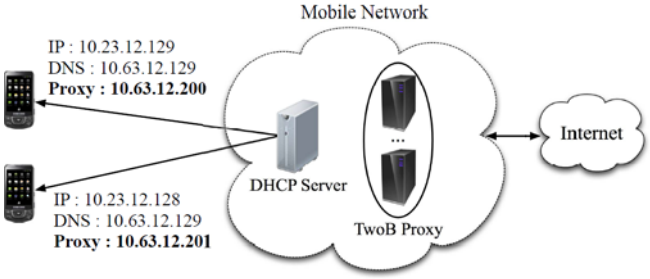


Figure 5. Assigning IP address of a proxy to devices

In order to seamlessly use our scheme, the module for providing the configuration of the proxy server is added to the DHCP server in the mobile network provider and the module for automatically setting up the information is installed in the mobile browser. When a smartphone connects to the internet through the mobile network, the configuration of the proxy server along with the smartphone's IP address and DNS server IP address is assigned to the smartphone as shown in Fig. 5.

When the browser architecture is managed by a mobile network provider, our scheme has three benefits. First, by eliminating the connection establishment phase over the mobile network, web loading time can be reduced for its subscribers. Second, end-users do not need to manipulate proxy setting in their smartphones. Since the configuration of the proxy is automatically set whenever the mobile network is available, users can easily exploit our scheme without manipulating the detailed configuration of the proxy server. Finally, from the perspective of the mobile network provider, a reduced number of packets decrease its network load.

The purpose of the proxy server in the proposed framework is clearly different from that of the conventional web proxy servers, which retrieves and stores frequently accessed contents to reduce network traffic. The conventional proxy servers are located in the client-end of slow network so that they reduce the traffic via the slow network and, in turn, provide fast response. However, the proxy server in the TwoB architecture is placed in the server-end of mobile network, which is significantly slower than WAN. In addition, previous studies revealed that data caching with a proxy server, which resides on the boundary between mobile network and WAN is less effective for reducing web browsing latency than connection caching [17]. Therefore, we believe that, the benefit from content caching by the proxy server in TwoB would be marginal to the overall response time. This means that the proxy server does not require large and fast storage devices and main memory, thus can be manufactured at low cost in comparison to the conventional proxy servers.

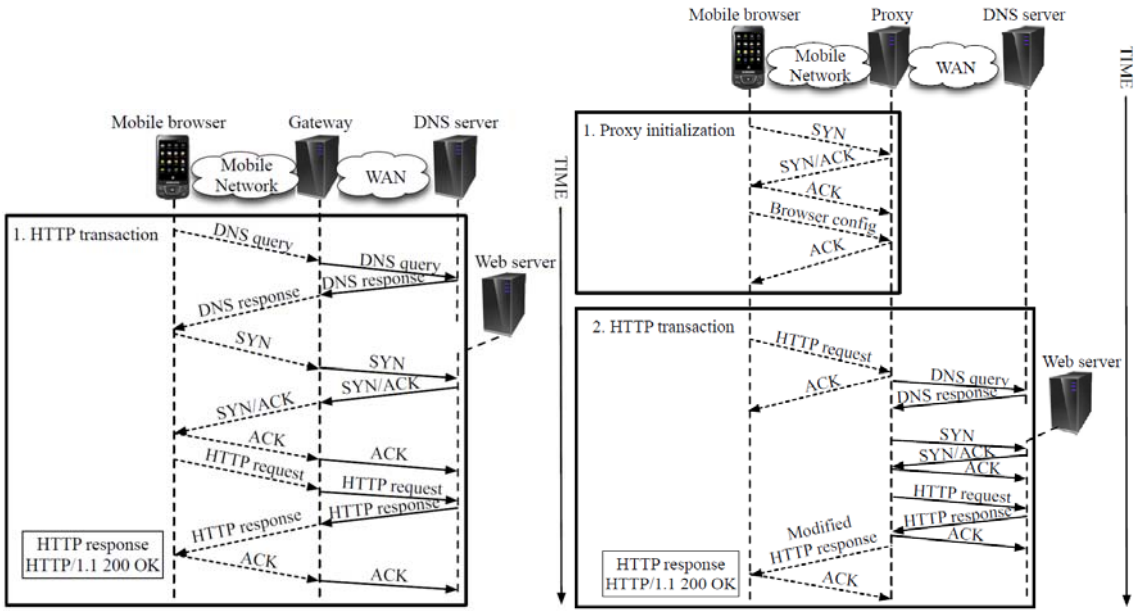
C. Exploitation of Persistent Connections

As described in Section 2.2, connection establishment phases during web page loading become a performance bottleneck when these phases are conducted on a long-RTT mobile network. An HTTP persistent connection, which is aimed at avoiding the connection establishment phase, however, cannot be fully exploited in a mobile browsing environment. Our scheme actively uses HTTP persistent connections for a better web browsing experience as follows.

1) On Mobile Network

The main purpose of using a proxy server is to avoid frequent connection establishment phases over the mobile network. General proxy servers, such as Squid, are in charge of only relaying HTTP requests and responses between web browsers and web servers. Those are not in charge of keeping connections with the browsers open. Accordingly, the connections between the browser and the proxy cannot be persistent when the proxy server disconnects them or when the HTTP responses have no keep-alive flags. In this regard, a new connection establishment phase can probably occur at any time if a mobile browser uses a general proxy server.

To avoid this unpredictable new connection establishment, our scheme tightly bounds the proxy browser, especially the browser back-end with a mobile browser. The connection



(a) In a general mobile browsing environment (b) In our scheme

Figure 6. The procedure for processing an HTTP request

between the browser and the proxy is only disconnected when a smartphone is powered off or is given a new IP address. Except for these cases, the connection is always kept open. We denote this type of connection as a semi-permanent connection.

In order to make use of the semi-permanent connection seamlessly, we use an HTTP persistent connection, which is a standard feature in HTTP version 1.1. By using an HTTP persistent connection, mobile browsers do not disconnect the opened connection with the proxy server. When a TCP connection is opened to fetch the main HTML page, the persistent connection feature keeps the connection open for succeeding HTTP transactions (for embedded objects such as images, JavaScripts, etc.) to the same web server. Accordingly, an additional connection establishment phase is not required, and the response time for fetching embedded objects is shortened. This feature is usually implemented in many web browsers and servers so that our scheme can be easily adopted into various native browsers, such as Firefox, Safari, and Android built-in browsers, with few modifications.

Instead of browser-side support, we need to manage the proxy server to completely support the HTTP persistent connection even if web servers do not support the persistent connection. Since the feature is adapted in HTTP version 1.1, many web browsers intentionally disconnect the TCP connection when the version field of an HTTP response is 1.0 and the keep-alive in the connection field is unset. If the proxy server relays such HTTP responses to the mobile without any modification of the responses, the browser will disconnect the connection even though the connection is actually established with the proxy instead of the web server. To cope with this issue, the connection manager in the proxy server replaces the value of the connection header field of each HTTP response with keep-alive if that value is close. As a result, the browser naturally keeps the connection open for subsequent HTTP transactions.

In the native browser approach, a persistent connection can be disconnected for the following two reasons. First, some browsers (e.g., Android built-in browser) terminate persistent connections even if the HTTP version is 1.1. Due to connection management cost, those browsers reap idle TCP connections after a configured timeout has occurred. In order to avoid this connection termination, we modified our prototype browser to disable the timeout function.

Second, most browsers limit the number of opened TCP connections below some threshold (e.g., eight entries in a connection cache in the Android built-in browser). When this connection cache is full, one connection is closed by a cache replacement policy (e.g., least recently used policy) to cache a new connection whose destination host is not found in the cache. In our scheme, this phenomenon never happens because the proxy server is the only destination host that the browser communicates with.

Fig. 6 shows how the communication occurs between the mobile browser, the proxy server, a DNS server and a web server in a general mobile browsing environment and in our scheme. In our scheme, the browser first initializes the proxy server by establishing persistent TCP connections and by sending the browser's information to the proxy which will be used for reducing the size of HTTP requests; Section 3.4 details the information. This phase occurs only once when the smartphone uses the mobile network. For each HTTP transaction, the HTTP request generated by the browser is sent to the proxy server. The proxy server retrieves the URL from the HTTP request and performs an HTTP transaction with the corresponding web server by resolving the IP address and by establishing a TCP connection. It is important to note that this connection establishment phase occurs only on the fast WAN. Then, the proxy sends the HTTP request and receives its response from a web server. Finally, the proxy server modifies the HTTP response to make it support the HTTP persistent connection and returns the modified response to the browser. As compared to Fig. 6(a), our scheme shows a shorter HTTP transaction phase as presented in Fig. 6(b) due to avoiding the connection establishment phase. In Fig. 6(b), the DNS query phase accounts for a large portion of time. But, successive transactions for embedded web objects that are hosted in the same web server do not require DNS query

phases.

Maintaining the semi-persistent connection does not affect the power consumption of a mobile device. The radio device is only turned on when it has some packets to transmit. Otherwise, it turns into a low-power state [25].

An open TCP connection can be disconnected when its associated NAT (network address translation) entry is removed from the NAT table of the mobile network carrier [26]. This, however, is an easily resolvable issue. The carrier can simply postpone the removal of NAT entries associated with the semi-persistent connections since the semi-persistent connection is a carrier-supported feature to provide better web browsing experience to its customers.

2) On WAN

In our browsing environment, the web browsing experience is not only determined by the mobile network side, but is also affected by the WAN side. Accordingly, it is important to optimize connections between the proxy server and web servers. In our architecture, the connection manager is in charge of managing this kind of connection.

The connection manager maintains two caches, a TCP connection cache and a simple DNS cache. The former is aimed at reducing TCP connection establishment phases by exploiting the HTTP persistent connection standard, and the latter is to reduce DNS query overhead. When an HTTP transaction is forwarded from the browser back-end, the connection manager looks up the DNS cache in order to reduce duplicated DNS queries. The DNS cache follows the DNS caching policy standard [18]. When the IP address of a web server is unknown, the connection manager sends the DNS query to an upper-level DNS server and stores the IP address in the DNS cache.

When the connection manager unveils the IP address of the web server to send an HTTP transaction, it should find out a cached (opened) connection in the connection cache. If this is not found, it opens a new connection with the desired web server. Then it forwards the HTTP request to the web server. When corresponding responses are transferred and if the web server supports the HTTP persistent connection standard, the connection is stored in the connection cache for further reuse.

Although our proxy server manages the connection cache, many web servers disconnect opened (idle) connections after some period. Accordingly, the connection manager only maintains opened connections until web servers disconnect them. Previous work revealed that 70-80% of real-world web servers keep an idle connection open for at least one second, and 65-76% of them for five seconds [15]. From this result, we believe that caching opened connections in the proxy server will help reduce the web loading delay when multiple mobile clients are managed in the proxy server.

D. Exploitation of Request Pipelining

As described in Section 2.2, the HTTP pipelining feature in HTTP 1.1 provides a way of increasing the responsiveness of HTTP transactions by sending multiple HTTP requests in a single TCP connection without receiving the previous responses. The feature also enables a single TCP packet to ship multiple HTTP requests. Accordingly the number of round trips for sending HTTP requests is decreased. The effectiveness of the pipelining, however, is degraded by the two factors described in Section 2.2: the size and the destination of HTTP requests.

In our browser architecture, we have an opportunity to increase the effect of HTTP pipelining. From the traced packets in Table I, we found that a substantial part of HTTP requests are invariant [19]. Those static header fields are Accept-Language, Accept-Charset, User-Agent, and so on; these fields usually specify the information for the mobile browser. Accordingly, the static header fields can be omitted in HTTP requests when the request is transferred to the proxy server. Instead, the omitted fields can be appended in the proxy server since the proxy server has a browser back-end that is specific to a mobile client. We denote the HTTP requests without static header fields as incomplete HTTP requests and HTTP requests having static header fields appended in proxy as complete HTTP requests.

The browser back-end in the proxy server in turn is responsible for appending the omitted static header fields. When the semi-permanent connection between the browser and the proxy is established, the browser sends its static header fields to the proxy. The browser back-end stores the static header fields in the static header cache by using the browser's IP address as a key. Subsequent HTTP requests sent from a browser are complemented by the proxy server using the stored static header fields. When the browser disconnects the semi-permanent connection for some reason (e.g., power off, assigning a new IP address, etc.) the stored static header fields for the browser are discarded.

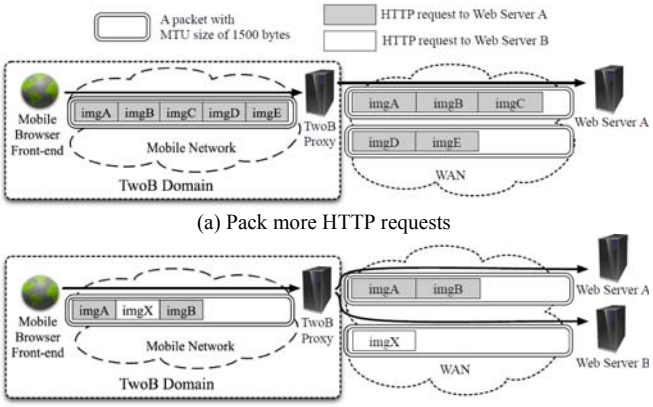


Figure 7. Improving the effectiveness of HTTP pipelining

By using this approach, the size of HTTP requests on the mobile network can be reduced, and the smaller the size of the HTTP requests is, the more HTTP requests can be packed into a single TCP packet. Therefore, the effectiveness of HTTP pipelining as limited by the size of the HTTP request can be improved. Even if the number of TCP packets holding HTTP requests is not decreased, the amount of data on the mobile network is still reduced. Fig. 7(a) shows that five HTTP requests can be packed in a single TCP packet in our architecture.

The second obstacle to HTTP pipelining depicted in Section 2.2, different destination hosts of HTTP requests, is naturally resolved in our browser architecture. When a browser directly connects to web servers, each HTTP

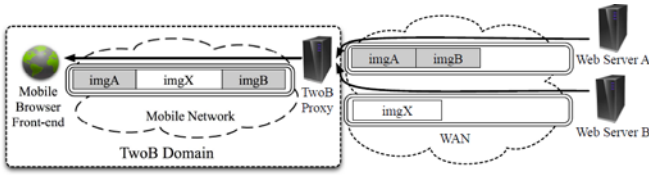


Figure 8. The procedure of processing HTTP response in our scheme

request with different hosts is sent to corresponding web servers through separate TCP packets. In our scheme, however, all HTTP requests are sent via the semi-permanent connection whose other end is the proxy server. Accordingly, the limitation caused by different hosts for HTTP requests is naturally resolved as shown in Fig. 7(b).

The side effect of using HTTP pipelining in our scheme is that when a browser requests multiple objects in different hosts, incomplete HTTP requests are sent in a single TCP connection, but the complete HTTP requests are sent to web servers in different TCP connections. Each web server may return HTTP responses in a different order due to their different levels of load, RTT and link speed. The standard of HTTP pipelining, however, is to keep the order of responses the same as the order of requests [6]. Accordingly, it is essential to adjust the sequence of HTTP responses to the browser front-end in the proxy server even though the responses from web servers arrive in different orders. In addition, if multiple responses from multiple web servers are the answers of requests from the same semi-persistent connection, the responses should be returned through that semi-persistent connection.

To cope with this problem, the pipeline manager in the proxy server arbitrates the sequence of requests and responses to make the order of responses the same as that of the requests. For example, a web browser front-end requests three objects, A, X and B. A and B are hosted in the same web server while object X is in a different web server. When the response for X first arrives, this response is not forwarded to the browser front-end until the responses for A and B arrive. Fig. 8 shows this response arbitration in the pipeline manager when the requests are sent as shown in Fig. 8(b).

Finally, the pipeline manager exploits HTTP pipelining supports of web servers. Even if WAN is fast in terms of link speed and RTT, reducing the overhead of network layers such as IP and data-link is important. Accordingly, the pipeline manager relays pipelined requests from the browser front-end to web servers.

IV. EVALUATION

A. Experimental Setup

We implemented the prototype of the web browser front-end based on the default web browser of the Android OS, and the TwoB proxy server based on Twisted, an event-driven web server framework. The Android OS with the prototype web browser was ported to a commercial smartphone for evaluation.

In order to emulate the web browsing over a mobile network, we set up the evaluation environment as shown in Fig. 9. Two Linux servers are used: (1) one emulates a mobile network provider, and (2) the other performs as a DNS server and web servers. The former node provides a Wi-Fi access point to the smartphone and runs a DHCP

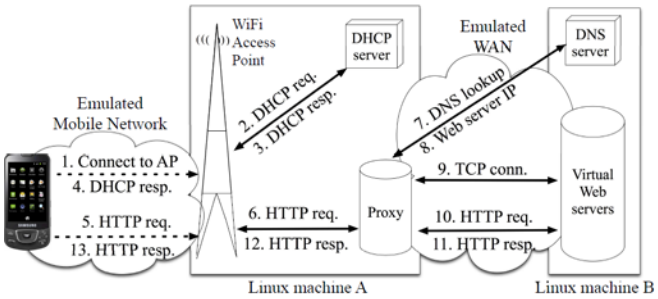


Figure 9. Evaluation system configuration

server and the TwoB proxy server. When the smartphone connects to the access point, the server assigns an IP address and provides the information for the TwoB proxy server and the DNS server in the second node. The second node also runs the Apache web server and provides multiple virtual hosts simulating different web sites. Since the DNS server returns the IP address of the virtual web servers for the DNS queries, every web request from the smartphone goes to our web servers. In each virtual web server, the time-out period of an idle connection is five seconds by default.

We configured the Wi-Fi network between the proxy and the smartphone to imitate the characteristics of 3G network services in Korea. In a series of experiments, the mobile network showed approximately 147.3 kbps and 13.4 kbps for downlink and uplink bandwidth, respectively, and the average RTT to various web sites was 265 ms. These characteristics are similar to those in a prior study [3]. Based on this observation, we injected artificial delay into the Wi-Fi network between the smartphone and the access point as shown in Table II. In addition, the two Linux machines have very low RTT compared with the WAN environment. In order to emulate the WAN, we injected 50 ms of RTT into the network link between the two machines as shown in Table II [3].

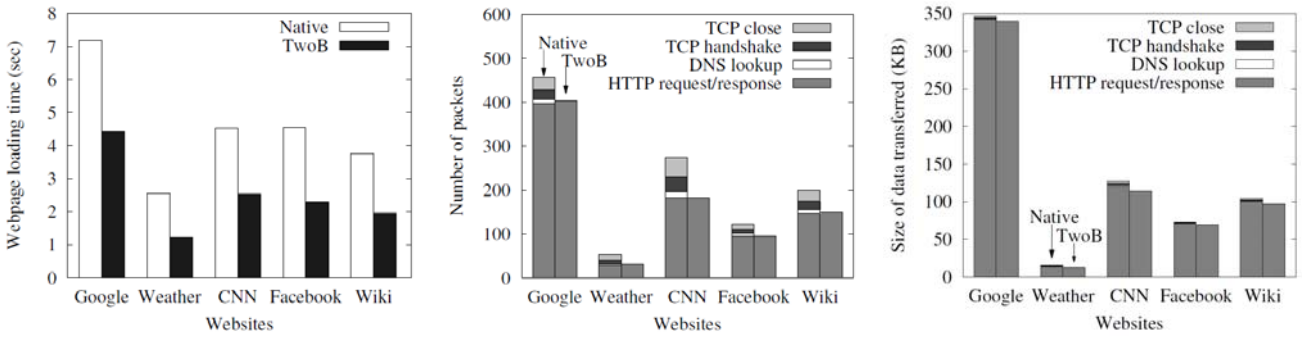
TABLE II. NETWORK CONFIGURATION FOR EVALUATION

Network	RTT(ms)	Uplink(kbps)	Downlink(kbps)
Emulated 3G network	200	50	250
Emulated WAN	50	-	-

Many web sites use dynamic web pages whose contents change by time and by accesses. Accordingly, both the number of packets and the size of transferred data for the same web site greatly vary during the evaluation. In order to eliminate these runtime variations, we replicated the contents of the web sites and used them in our evaluation. Detailed information about the replicated web pages is summarized in Table III.

TABLE III. AVERAGE NUMBER OF PACKETS/SIZE OF DATA (KBYTES) FOR LOADING REPLICATED WEB PAGES IN OUR EVALUATION. THE LAST COLUMN OF EACH WEB SITE DENOTES THE PORTIONS OF PACKETS/SIZE COMPARED TO THE ORIGINAL WEB PAGES SHOWN IN TABLE I

Web Sites	Whole Page TCP Closes	DNS Queries HTTP Requests	TCP Handshakes Portion to Original
Google	457/346.3	10/0.9	21/1.5
	28/1.8	11/5.8	82%/74%
Weather	53/15.7	4/0.3	9/0.6
	12/0.8	5/2.5	76% / 61%
CNN	275/127.1	14/1.2	33/2.3
	44/2.9	25/13.1	79% / 61%
Facebook	122/72.9	6/0.5	9/0.6



(a) Web loading delay (b) The number of packets (c) The size of data

Figure 11. Web loading delay, the number of packets passed through the mobile network and the amount of data passed in the native browser and TwoB

	12/0.8	6/2.8	64%/58%
Wiki	199/104.2	10/0.9	19/1.3
	25/1.6	12/6.4	97%/87%

B. *Evaluation Results*

In our evaluation, we measured the number of packets, the size of transferred data and web loading time to access a web site. The former two results are measured at the access point using tcpdump. The web loading time is the elapsed time from when a user clicks the load button to when the progress bar of the browser hits 100% [4]. To measure the web loading time, the browser was modified to record time stamps for its internal operations.

At every iteration, the local cache of the browser is flushed (this leads to flushing connection caches together) and all web sites are sequentially visited with a time interval of 30 seconds. The results are average values obtained from fifty iterations. Native denotes the results from the unmodified native browser for comparison and TwoB denotes the results from our proposed scheme.

1) *Native vs TwoB*

In this subsection, we measured the web loading time of our scheme in comparison with that of the Android native browser. Fig. 10 shows the web loading delay, the number of packets passed through the mobile network and the amount of data passed in both schemes, native browser and our scheme. As shown in Fig. 10(a), our scheme reduced web loading time by 38-52% compared to the native browser. Since all packets for connection establishment and close are handled in the TwoB proxy, our scheme eliminates the packets in this category as illustrated in Fig. 10(b). As compared to the native browser, our scheme reduces the number of packets by 11-41%. The reduction in the number of packet is not as significant as the reduction in the web loading time because the connection packets and DNS packets are synchronous to the web loading process. When these packets are not complete, no further processes can progress. Accordingly, by reducing these synchronous packets, the browser can quickly progress on fetching web objects.

Fig. 10(c) shows the amount of data passed through the mobile network. The main cause of data reduction is due to omitted connection packets and reduced size of HTTP requests, hence incomplete HTTP requests. But, since the amount of data for connection management and the HTTP request is tiny compared to the size of HTTP responses, the data size reduction is not as great as the reduction in the number of packets.

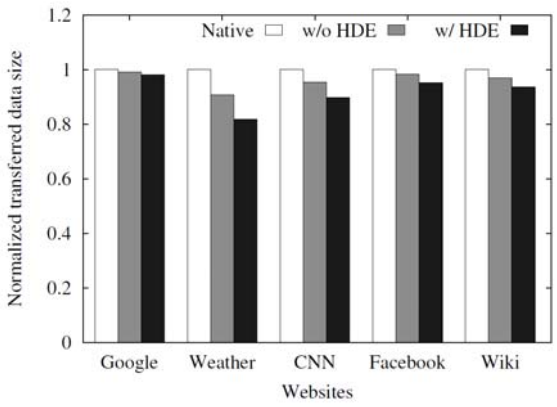


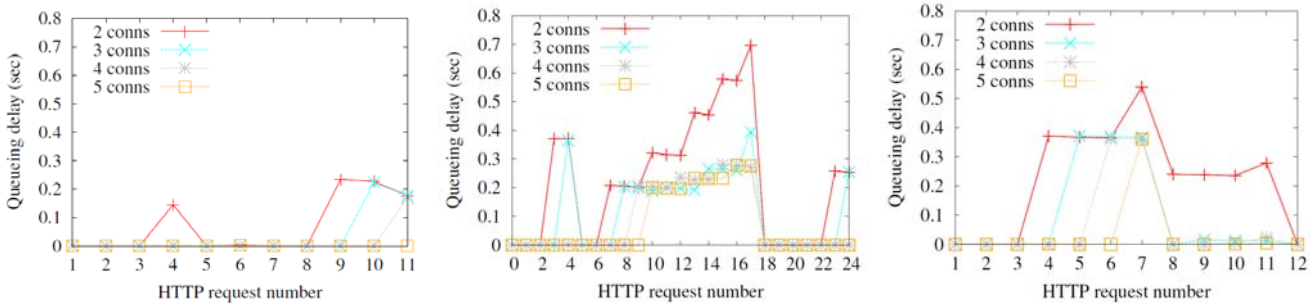
Figure 10. Normalized transferred data size on the mobile network when static HTTP header fields are complemented in the TwoB proxy

Many previous studies revealed that the web browsing experience is largely correlated not with the bandwidth of the mobile network but with its long RTT [3-4]. Our evaluation results are consistent with the results found in previous studies. The main benefit of our browser architecture is eliminating unnecessary connection-related packets by maintaining a semi-persistent connection. Accordingly, our scheme does not incur several round trips for connection-related packets as depicted in Fig. 10(b). Although the amount of data reduction by using our scheme is negligible, our scheme reduced web browsing latency by half by reducing the number of round trips.

As compared to the result in our preliminary study [23], the web loading time of the native browser increased by 1100 ms on average due to injecting realistic WAN delay. Our scheme, however, showed still decreased web loading time by 60 ms on average. Since the TwoB proxy maintains a local DNS cache and exploits HTTP persistent connections with web servers, these features help to offset the increased WAN delay to the measured web loading time.

Now, we measured the effect of appending static header fields in the TwoB proxy. This operation is denoted as HDE (header delta encoding). Fig. 11 shows the size of data transferred through the mobile network normalized to the Native browser. As shown in the figure, the effect of HDE resulted in a data size reduction of 2-19% compared to the native browser. The effect of data reduction outperforms over the effect of only using the semi-persistent connection (denoted as w/o HDE in Fig. 11) between the browser front-end and back-end (1-9%).

In addition, exploiting HDE can increase the number of packets that can be packed into one packet. Table IV shows



(a) Google (b) CNN (c) Wiki

Figure 14. Queueing delays of HTTP requests in the request queue in the browser front-end

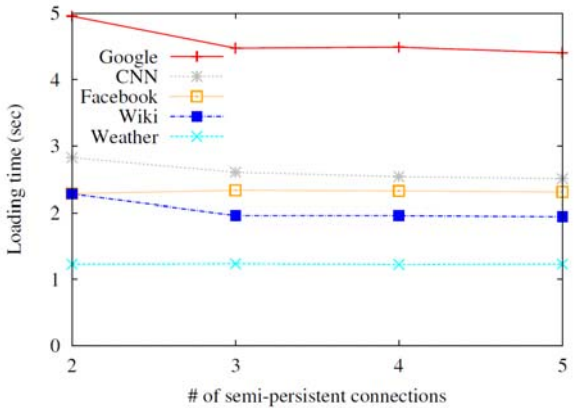


Figure 12. Web loading times as a function of the number of semi-persistent connections

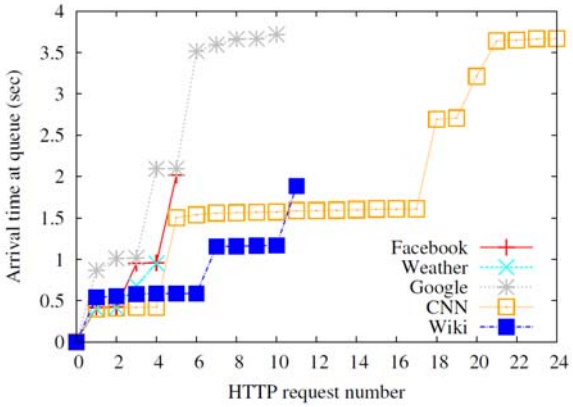


Figure 13. Enqueueing times of HTTP requests in each web site

the average size of HTTP requests in each web site. As depicted in the table, the requests including static header fields (complete HTTP requests) are 544 bytes in size on average. However, the average size of dynamic fields is only 283 bytes. This result means that omitting static header fields can increase the number of packable requests from 2.7 to 5.3 in the tested cases. Accordingly, the number of request packets in the mobile network can be reduced when our scheme is used by a mobile network provider.

TABLE IV. AVERAGE PACKET SIZE OF COMPLETE AND INCOMPLETE HTTP REQUESTS IN BYTES

Sites	Complete HTTP reqs	Incomplete HTTP reqs
Google	540	270
Weather	511	245
CNN	641	350
Facebook	479	213
Wiki	598	339
Average	554	283
# of reqs in on MTU (1500 bytes)	2.7	5.3

2) Effects of TwoB Proxy Configuration

Although our browser architecture can improve web browsing, it is important to find the best configurations of our TwoB proxy and the browser front-end. The most important parameter is the number of semi-persistent connections since if the number of connections per browser front-end is large, the number of browsers that can be handled by a TwoB proxy is limited.

Fig. 12 shows the web page loading times with varying numbers of semi-persistent connections. From the figure, two different results can be found: no loading time reduction and loading time reduction when the number of connections increases. The results of Facebook and Weather can be categorized in the former result, and those of Google, CNN and Wiki are in the latter one. In addition, the loading times of the three web sites are mostly saturated when the number of semi connections is four.

In order to reveal the reasons for the results in Fig. 12, we conducted additional experiments. Fig. 13 shows the time at which each HTTP request is enqueued in the HTTP request queue in the browser front-end. The x-axis shows the serialized HTTP requests in each web site and the y-axis shows the time each HTTP request is queued in the HTTP requests queue in the browser front-end since the web page loading began. As shown in the figure, the request arrival patterns of two web sites, Facebook and Weather, are sparse. Hence, both web sites have a few HTTP requests and each request arrives with a long time interval. Accordingly, the chance to exploit additional connections is low since the time gap is large enough to complete previous HTTP transactions.

The three web sites, Google, CNN and Wiki, however, have many HTTP objects and request tens of HTTP requests or more. Their arrival patterns are denser than the other two web sites. This means that more than five requests are queued at mostly the same time. Accordingly, exploiting more connections can lead to a reduction of queue sojourn time.

In order to validate our analysis, we also measured the queueing delay of HTTP requests in the browser's HTTP request queue. The queueing delay of an HTTP request is the time between when the request arrives at the queue and when the request is transferred to the TwoB proxy. Accordingly, when multiple objects are queued and if the throughput of the queue consumer, the number of semi-persistent connections in our case, is low, the objects will have long queueing delays.

Fig. 14 shows the queueing delays for the three web sites,

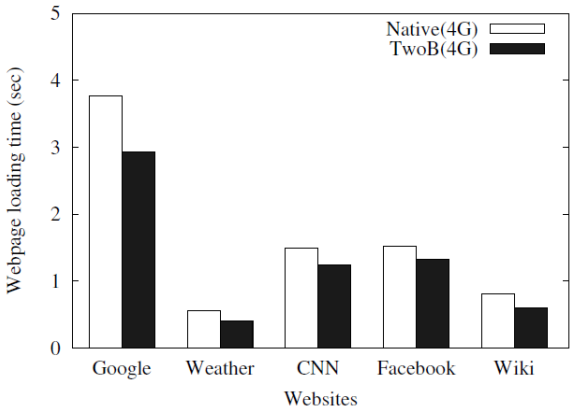


Figure 15. Web loading delay of the five web sites in the native and TwoB on 4G network

Google, CNN and Wiki, each of which showed a dense pattern in HTTP request arrival time. We also varied the number of semi-persistent connections to show the effect of the number of semi connections. As shown in the figure, when the number of connections increases, the queueing delays of many HTTP requests are reduced because multiple connections quickly consume the requests in the queue.

3) Performance on Emulated 4G Network

The network configuration we used assumes a slow mobile network, such as a 3G network, which is not a trivial option in recent mobile network providers. In order to figure out whether our scheme is still useful on a high-speed network environment, such as a 4G network, we measured the web loading time on a 4G-emulated network environment. In this environment, the RTT between the mobile web browser and the proxy server is configured to 54 ms and the RTT between the proxy and the web servers is 16 ms; the RTT between a mobile browser and a web server becomes 70 ms which is a typical performance of 4G network as depicted in [24].

Fig. 15 shows the web loading time of the five web sites. In the figure, the absolute web loading times are faster than those on the 3G-emulated network in Fig. 10(a) because of the decreased link delays. Our scheme, however, reduces the overall web loading time by 21% on average as compared to the native browser. Since the wireless delay is still higher than the WAN delay even in a high-speed mobile network, reducing the number of packet traversals on the slow part of communication paths results in the reduced web loading time.

V. RELATED WORK

Many optimization techniques in the protocol layer have been proposed for the sake of reducing delays in web browsing in a heterogeneous bandwidth network environment [17], [19-21]. Chakravorty et al. revealed that the optimizations in the application and session layer dominate the optimization techniques in other layers [21]. Our scheme also exploits the application and the session layer techniques to reduce web browsing delay.

Feldmann et al. analyzed the effectiveness of proxy in the middle of a heterogeneous network environment [17]. Their trace-driven simulation showed that not only caching web objects but also caching connections in the proxy improves web browsing performance. Our scheme, however, proposes a two-tier web browsing architecture specialized for a mobile network albeit we use a persistent connection between the proxy and the mobile browser. In addition, we increase the effectiveness of HTTP pipelining by appending the static header fields at the proxy side.

Transparent proxy-based approaches have been proposed to improve the web browsing performance in wireless and cellular networks [20-21]. A transparent proxy modifies the main HTML file or DNS reply to make the browser forward subsequent HTTP requests to the proxy. Accordingly, subsequent web accesses benefit from the proxy. These approaches, however, require at least one DNS lookup and one TCP handshake for opening each web site. A few round trips are sufficient to increase the delay in web browsing in a long-RTT mobile network. In addition, HTML rewriting could increase the load on the proxy server.

Our scheme can enable the smartphone to automatically set up a proxy server by the DHCP server in a mobile network provider when the smartphone accesses the mobile network. Therefore, our browser leverages the benefits of an explicit proxy configuration. The number of packets for establishment between the browser and the proxy server is equal to the maximum number of connections concurrently available in the browser regardless of the number of websites visited during web browsing. Also, all DNS lookups delegate to the proxy server.

Liu et al. proposed application-level compression techniques, called HTTP protocol aware compression (HPAC), that provide several HTTP header-specific encoding methods to reduce the delay in web browsing within mobile systems [19]. Their purpose is to reduce the size of HTTP requests and response messages transferred over the mobile network. Static binary encoding (SBE) and dynamic binary encoding (DBE) convert the encoding of HTTP header fields from ASCII code to binary code depending on HTTP header field characteristics. Along with the SBE and DBE, they introduced header delta encoding (HDE), which sends only the changed part of the HTTP header from a base header. Our scheme is a simplified version of HDE and consumes little CPU power for encoding. In addition, we demonstrated the increased effectiveness of HTTP pipelining in the tested web sites.

Belshe et al. proposed a SPDY protocol [22] that is compatible to the HTTP protocol but improves web browsing latency by multiplexing multiple web transactions in one connection. By using this protocol, a mobile browser does not need to establish multiple connections and can minimize time-consuming connection establishment phases. This protocol is complementary to our approach because each web page loading incurs at least one round trip over the mobile network to establish the first connection. This round trip time can also be eliminated if our scheme is used. In addition, The SPDY protocol also provides header field compression to reduce the size of web requests. But, it does not eliminate the static header fields so that even if the static header fields are compressed, the compressed data still pass through the mobile network. Our scheme, however, eliminates the static header fields passing through the mobile network so that the number and size of packets can be reduced.

Preliminary study on the effect of two-tier web browser architecture was studied in [23]. This study, however, only focused on taking advantage of the persistent connection between a mobile browser and a proxy server in a mobile network provider. Our approach enhances the preliminary study in terms of the detailed architecture and the effective configuration of the two-tier browser architecture. In more detail, our approach tries to reduce latency associated with WAN by exploiting HTTP persistent connections with web servers and by maintaining DNS cache inside the TwoB proxy server. In addition, since the proxy server is not a general proxy server but a specialized proxy to a mobile browser, we sought to find the best configuration of the browser and the proxy server in terms of the number of persistent connections between the browser and the proxy by analyzing the packet processing time with varying the number of persistent connections for well-known web sites.

VI. CONCLUSION

In conventional web browser architecture, loading a web page usually incurs a large number of DNS lookups for resolving web server addresses, and TCP handshakes with web servers. Web browsing on mobile network is sluggish in comparison to browsing on wired or Wi-Fi networks because the RTT of a packet is relatively long on the mobile network and transferring many small packets for the DNS lookups and TCP handshakes is required for a web page loading instance.

In order to improve the mobile web browsing performance, we proposed a two-tier web browser architecture that consists of a mobile web browser and a proxy server. The proxy server is located at the joint between the WAN and mobile network, and the mobile web browser stays connected to the proxy server with the persistent connection defined in the HTTP 1.1 standard. The proxy server conducts DNS lookups and TCP handshakes as a representative of the mobile web browser. In addition to this, the proxy server adds HTTP header fields to HTTP requests on behalf of the web browser so that the HTTP header field data is stripped from the packets on the mobile network. With these approaches, the proposed architecture reduces both the number and size of mobile network packets.

We implemented a prototype of the proposed architecture on a commercial smartphone and evaluated it in terms of the number of mobile network packets and web page loading time. The experiment results showed that the proposed scheme reduced the number of mobile network packets by up to 42% and shortened the web page loading time by up to 52% in comparison to the conventional web browser. Because the proposed architecture can be easily implemented with minor modifications to the existing proxy servers and mobile web browsers, we believe that our solution is practically applicable to the existing mobile network infrastructure and mobile electronics.

REFERENCES

[1] Ericsson, Traffic and market data report, Tech. Rep., Nov. 2011.

[2] P. JäPpinen, R. Guarneri, and L. M. Correia, "An applications perspective into the future internet," J. Netw. Comput. Appl., vol. 36, no. 1, pp. 249-254, Jan. 2013.

[3] J. Huang, Q. Xu, B. Tiwana, Z. Mao, M. Zhang, and P. Bahl, "Anatomizing application performance differences on smartphones," In Proc. 8th Int. Conf. Mobile Systems, Applications, and Services, pp. 165–178, 2010.

[4] Z. Wang, F. Lin, L. Zhong, and M. Chishtie, "Why are web browsers slow on smartphones?," in Proc. 12th workshop Mobile Computing Systems and Applications, pp. 91–96, 2011.

[5] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin, "A first look at traffic on smartphones," in Proc. 10th Int. Conf. Internet Measurement, pp. 281–287, 2010.

[6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, Hypertext transfer protocol–HTTP/1.1, Internet Request for Comments (RFC 2616), Jun. 1999.

[7] Nielsen.com, Top mobile phones, sites and brands for 2009, 2009

[8] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. Singh, "Who killed my battery?: analyzing mobile browser energy consumption," in Proc. 21st Int. Conf. World Wide Web, pp. 41–50, 2012.

[9] K. Kim, H. Yang, C. Kim, and S. Kim, "A parallel approach to mobile web browsing," in Proc. Int. Conf. Mobile Computing, Applications, and Services, pp. 338–344, 2012.

[10] A. Fox, I. Goldberg, S. Gribble, D. Lee, A. Polito, and E. Brewer, "Experience with top gun wingman: A proxy-based graphical web browser for the 3com palmpilot," in Proc. IFIP Int. Conf. Distributed Systems Platforms and Open Distributed Processing, pp. 407–424, 2009.

[11] J. Kim, R. Baratto, and J. Nieh, "pTHINC: a thin-client architecture for mobile wireless web," In Proc. 15th Int. Conf. World Wide Web, pp. 143–152, 2006.

[12] H. Shen, Z. Pan, H. Sun, Y. Lu, and S. Li, "A proxy-based mobile web browser," in Proc. Int. Conf. Multimedia, pp. 763–766, 2010.

[13] B. Zhao, B. Tak, and G. Cao, "Reducing the delay and power consumption of web browsing on smartphones in 3G networks," in Proc. Int. Conf. Distributed Computing Systems, pages 413–422, 2011.

[14] Z. Al-Qudah, M. Rabinovich, and M. Allman, "Web timeouts and their implications," In Passive and Active Measurement, pp. 211–221, 2010.

[15] E. Hernandez, "War of the mobile browsers," Pervasive computing, IEEE, vol. 8, no. 1, pp. 82–85, 2009.

[16] B. Zhao, Q. Zheng, G. Cao, and S. Addepalli, "Energy-Aware Web Browsing in 3G Based Smartphones," in Proc. 33rd Int. Conf. Distributed Computing Systems, pp. 165-175, 2013.

[17] A. Feldmann, R. Caceres, F. Douglis, G. Glass, and M. Rabinovich, "Performance of web proxy caching in heterogeneous bandwidth environments," in Proc. 18th Int. Conf. Computer Communications, volume 1, pp. 107–116, 1999.

[18] D. Barr, Common DNS operational and configuration errors, Internet Request for Comments (RFC 1912), Feb. 1996.

[19] Z. Liu, Y. Saifullah, M. Greis, and S. Sreemanthula, "HTTP compression techniques," in Proc. Conf. Wireless Communications and Networking, vol. 4, pp. 2495–2500, 2005.

[20] P. Rodriguez, S. Mukherjee, and S. Ramgarajan, "Session level techniques for improving web browsing performance on wireless links," in Proc. 13th Int. Conf. World Wide Web, pp. 121–130, 2004.

[21] R. Chakravorty, S. Banerjee, P. Rodriguez, J. Chesterfield, and I. Prat, "Performance optimizations for wireless wide-area networks: Comparative study and experimental evaluation," in Proc. 10th Annu. Int. Conf. Mobile Computing and Networking, pp. 159–173, 2004.

[22] M. Belshe, and R. Peon, SPDY Protocol, 2012

[23] J. Cho, J. Jeong, and E. Seo, "TwoB: a two-tier web browser architecture optimized for mobile network," in Proc. 10th Int. Conf. Advances in Mobile Computing & Multimedia, pp. 267-270, Dec. 2012.

[24] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck, "An in-depth study of LTE: effect of network protocol and application behavior on performance," in Proc. ACM SIGCOMM, pp. 363-374, 2013.

[25] F. Qian, S. Sen, and O. Spatscheck, "Silent TCP connection closure for cellular networks," in Proc. ACM Conf. Emerging Networking Experiments and Technologies, pp. 211-216, 2013.

[26] Z. Wang, Z. Qian, Q. Xu, Z. M. Mao. M. Zhang, "An untold story of middleboxes in cellular networks," in Proc. ACM SIGCOMM, pp. 374-385, 2011.