TwoB: A Two-Tier Web Browser Architecture Optimized for Mobile Network

Junguk Cho
College of Information and Communication Engineering, Sungkyunkwan University, Rep. of Korea 440-746
jmanbal@skku.edu

Jinkyu Jeong
Computer Science Dept., Korea Advanced Institute of Science and Technology, Rep. of Korea 305-701
jinkyu@calab.kaist.ac.kr

Euiseong Seo
College of Information and Communication Engineering, Sungkyunkwan University, Rep. of Korea 440-746
euiseong@skku.edu

ABSTRACT

The connection establishment phase including DNS lookups and TCP handshakes takes significantly long time during web browsing through mobile network. In this paper, we propose a novel web browser architecture that aims at improving mobile web browsing performance. Our approach delegates the connection establishment phase and HTTP header field delivery to a dedicated proxy server located at the joint point between WAN and mobile network to reduce both the number and size of packets on mobile network. Our evaluation showed that the proposed scheme reduces the number of mobile network packets by up to 52% and, consequently, shortens the average page loading time by up to 37%.

Categories and Subject Descriptors

H.4.3 [**Communications Applications**]: Information browser

General Terms

Design, Experimentation

Keywords

web browsers, smart phones, mobile network, proxy, HTTP

1. INTRODUCTION

More and more people are web browsing on mobile devices such as smart phones and tablets instead of PCs or laptops. However, the web browsing through mobile network is usually unsatisfactory due to its slow response. Lots of researchers have claimed that the long round-trip time (RTT) of mobile network is the most significant inhibitor of fast mobile web browsing [2, 6].

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

In order to improve the mobile web browsing performance over mobile network, 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 wide-area-network(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 on behalf of the mobile web browser. In addition to this, the proxy server adds HTTP header fields to HTTP requests for the web browser so that the HTTP header fields are stripped off from the packets on mobile network. With these approaches, the proposed architecture is expected to reduce both the number and size of mobile network packets.

2. MOTIVATION AND BACKGROUND

Mobile web browsers generally fall into two categories: the thin-client and the native browser architecture [1, 5].

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 web page, renders the web page image, and sends it 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 is appropriate for the limited computing power of mobile devices [3, 5, 7]. In addition, it reduces the web page loading delay by eliminating DNS lookups, TCP handshakings and HTTP header fields from mobile network communication between the proxy and the browser.

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 [1].

Second, although it may reduce the number of small packets, the thin-client architecture sometimes increases the amount of data transferred over mobile network [3, 5, 7] due to lack of local cache for storing web objects and the size of the

Table 1: Average number of packets/size of data (in KBytes) for loading mobile web pages.

Web sites	Whole page	Connection establishment	HTTP Requests	Reused conn.
google	528.7/372.8	27.0/3.7	15.4/10.0	7.1
weather	80.0/30.2	15.5/2.0	9.0/5.9	5.8
espn	908.7/556.8	103.4/11.4	40.5/25.7	16.2
cnnmobile	433.7/209.0	68.1/6.3	27.2/18.2	8.2
facebook	157.8/76.2	25.8/2.8	8.8/5.1	2.9
wikipedia	216.0/95.7	33.0/2.9	13.7/7.8	4.7

rendered web page images.

Finally, this approach increases burdens of the proxy server by heavy computation loads. If the proxy server is overloaded it could adversely affect the web browsing performance [1].

As the hardware performance of mobile devices improves, lots of commodity mobile devices employ native browsers.

Native browsers handle every step to process a web page request by itself so that they can provide interactive and dynamic web pages like most web browsers currently being used in PCs. Moreover, their human interface like zooming in/out feels more natural and their page loading time feels faster than the thin-client counterparts because they draw intermediate rendering during page loading and update it continually.

However, the native browsers process the connection establishment phases over mobile network, where the RTT is critically slow, and this severely harms the page loading time.

In order to reveal the proportion of the packets for the 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 which are announced by the Nielsen company in 2009.

Table 1 shows average number of packets and size of data for opening web sites.

Although the size of packet for the connection establishment operations is negligible, the connection establishment phase significantly affects the overall web page loading time since content download begins after completing the connection establishment phase and the RTT of mobile network is severely long.

To improve web page loading time, the HTTP 1.1 standard introduced the HTTP persistent connection and HTTP pipelining.

The HTTP persistent connection is a technique that keeps a TCP connection alive after finishing an HTTP transaction, and reuses the open connection for the forthcoming HTTP transactions instead of opening new connections.

A browser stores an open persistent connection in the connection cache after finishing an HTTP transaction. 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 the reused connections is smaller than the half of the total number of HTTP requests as shown in Table 1. This is because a web page usually has multiple embedded objects that are served by different web sites, and new connections are necessary to access such embedded objects. In addition, the limited capacity of the connection cache enforces the connection cache to discard cached connections that may be reused in the near future.

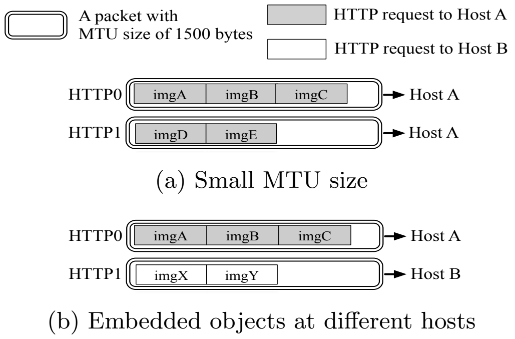


Figure 1: Disturbance factors of HTTP pipelining

The HTTP pipelining, which enables web browsers to issue HTTP requests without receiving the previous responses, improves the web page loading delay dramatically. With HTTP pipelining, a web browser can pack multiple HTTP requests in a single TCP packet. Accordingly, the number of round trips for HTTP requests is decreased. The effectiveness of the pipelining, however, is disturbed by two factors as shown in Figure 1. First, due to the MTU size, more than a single packet could be necessary. Second, when embedded objects are served from multiple web sites, transmitting multiple packets is unavoidable.

Considering the slow RTT of mobile network, the packet reduction from both persistent connection and request pipelining is crucial for the web page loading time. Our approach aims at maximizing the benefits from these two techniques.

3. OUR APPROACH

We propose a Two-Tier Web Browser architecture named TwoB that consists of a mobile web browser and its corresponding proxy server, which resides at the joint point between WAN and mobile network.

Like a native browser, the browsers in our model process all steps to load a web page except the connection establishment phase, which is handled by the proxy.

The IP address of the proxy server is delivered by mobile network provider's DHCP server when a mobile device connects to mobile network, and then the mobile OS configures the proxy server for its web browser automatically. Thus, users do not need to be aware of our scheme.

Figure 2 shows how the communication occurs between the mobile browser and the proxy server, and also between the proxy server and a DNS or Web server.

When a browser loads a web page first, the connection between the browser and the proxy is established. The browser always keeps the initial connection open. It sends and receives all future packets for HTTP transactions through this initial connection. In order to benefit from parallel processing, multiple connections between the proxy and browser can be open and managed.

All HTTP requests generated by the browser are sent to the proxy server through the open connections. Then, the proxy server retrieves the URL from the forwarded HTTP

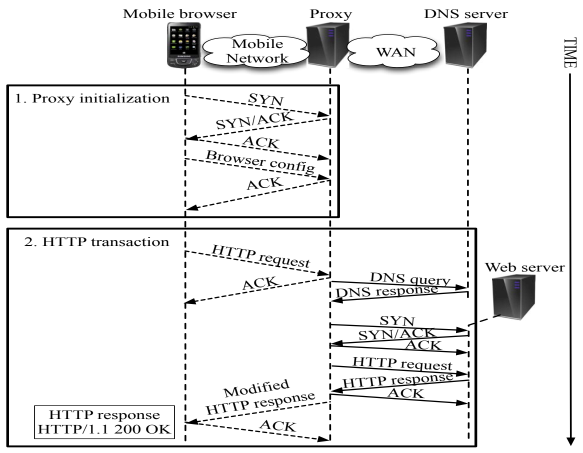


Figure 2: Processing HTTP requests in TwoB architecture

requests and performs required transactions with the DNS server and the corresponding web servers. After retrieving the requested web pages, the proxy server modifies the HTTP headers to make them support the HTTP persistent connection and returns them to the browser.

For this persistent connection, the proxy server must manage the HTTP persistent connection even when web servers do not support the persistent connection. Since the persistent connection is adapted from 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 of the HTTP header is unset. If the proxy server relays such HTTP responses to the mobile browser, the browser will disconnect the connection albeit the connection is actually established not with the web server, but the proxy. To cope with this issue, the proxy server replaces the version field of each HTTP response with 1.1 and eliminates connection header field regardless of the HTTP version. As a result, the browser naturally keeps the connection open for subsequent HTTP transactions.

Due to connection management cost, some of the existing browsers reap idle TCP connections after configured timeout interval. In order to avoid this, the timeout interval of the TwoB browser is set to be indefinitely long.

Most existing browsers limit the number of open TCP connections below a predefined number. When the connection cache is full, one connection is closed by a cache replacement policy to accommodate a new connection to a new web site. In our scheme, such case never occurs because the proxy server is the only destination that the browser communicates with.

In our browser architecture, we have an opportunity to increase the effect of HTTP pipelining by alleviating the two disturbing cases described in Section 2.

From the traced packets in Table 1, we found that a substantial part of HTTP requests are invariant static header fields [4]. Those static header fields are *Accept-Language*, *Accept-Charset*, *User-Agent*, and so on, which specify the mobile browser information.

In the TwoB architecture, such static header information of the browser is delivered to the proxy server when the first connection is established. The proxy uses the delivered HTTP header fields when it sends HTTP requests to a web server on behalf of the browser. Accordingly, such static header fields are unnecessary in the HTTP requests sent to the proxy server. Because of this, the size of HTTP requests on mobile network is reduced. The smaller the size of HTTP requests is, the more HTTP requests are packed in a single TCP packet. Therefore, the effectiveness of HTTP pipelining limited by the size of the MTU size improves. Even when the number of TCP packets holding HTTP requests is not decreased, the size of data on mobile network is still reduced.

The ineffectiveness in pipelining due to embedded objects located in multiple hosts is naturally resolved since the only destination of all HTTP requests is the proxy server in our architecture.

Table 2: Number of HTTP requests and necessary packets for accessing replicated mobile web pages.

Web sites	Number of HTTP requests	# of packets			
		Total	DNS lookups	TCP handshakes	TCP close
google	11.0	454.3	10.0	18.0	26.0
weather	6.0	65.3	6.0	12.0	16.0
cnnmobile	23.0	275.2	14.0	35.5	47.3
facebook	6.0	124.0	6.0	9.0	13.0
wikipedia	12.0	125.2	10.7	18.0	24.0

4. EVALUATION

We implemented the prototype web browser based on the Android OS web browser, and the proxy server with *Twisted*, an event-driven web server framework.

We set up the evaluation environment to emulate web browsing over mobile network.

First, we configured the Wi-Fi network with 200 ms injected delay in RTT and 150KBps/14KBps downlink/uplink bandwidth between the proxy and the smartphone, respectively, to imitate the characteristics of typical 3G network services.

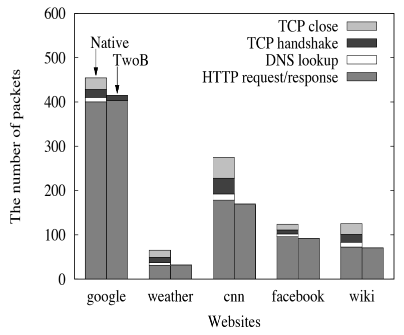
Second, we replicated the contents of the web sites and used them in our evaluation to eliminate the response time variations found in real web sites. The detailed information about the replicated web pages is summarized in Table 2.

Lastly, we ran an Apache web server to provide virtual hosts for the replicated web sites and hosted our own DNS server so that every web request from the smartphone goes to our web server with replicated web pages.

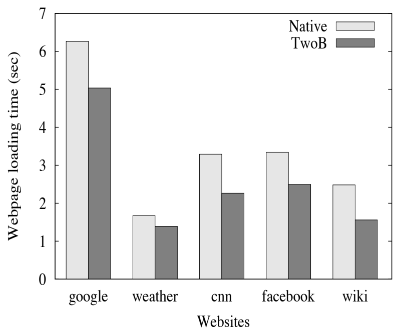
We measured the number of packets, the size of transferred data and the overall loading time to access a web site at the access point using *tcpdump*. The web loading time is defined as the elapsed time from when a user clicks the load button to when the progress bar of the browser hits 100%.

At every iteration, the local cache of the browser was flushed and this leaded to flushing connection caches together. All web sites were sequentially visited with a time interval of 30 seconds. The results shown in Figure 3 are average values obtained from fifty iterations. *Native* denotes the results from the unmodified native browser for comparison and *TwoB* denotes the results from the proposed scheme.

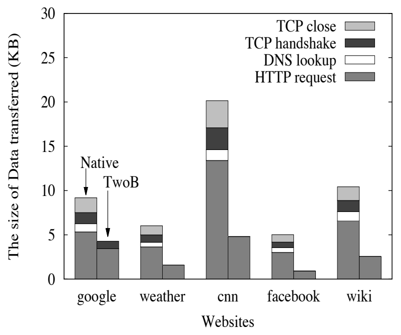
Figure 3(a) shows the number of packets to load each web site. Except *google* web site, the packets for the connection establishment phase were eliminated. Twelve TCP handshaking packets were transferred over the mobile network to



(a) Number of packets



(b) Web loading time



(c) Data size except web contents

Figure 3: Comparing results of a conventional web browser and TwoB

access *google* web site, which was the first site to visit, because the browser manages four simultaneous connections, and each connection requires three handshaking packets. After all persistent connections were established between the browser and the proxy, no more TCP handshakes occurred. As a result, the number of packets passing through the mobile network was decreased by 12% to 52% for each site. The reduction rate depended on both the number of embedded objects in web pages and the size of the embedded objects.

Figure 3(b) shows the time to load the web sites. Our scheme reduced the web browsing latency by 17% to 37% in comparison to the native browser. This improvement was majorly due to the elimination of the connection establishment phase. However, the time gaps between our scheme and the native browser were smaller than the expected values, which are the number of packets for connection establishment phase multiplied by the injected RTT. For example, loading the *weather* web site generates three and four of DNS lookups and TCP handshakes, respectively. Accordingly, the expected time reduction is 200 ms * 7 = 1.4 seconds. The process of the round-trip packets, however, is overlapped by the four concurrent persistent connections. As a result, the measured improvement was smaller than the expected.

Finally, we present the size of HTTP requests since our scheme minimizes the size of HTTP requests passing through the mobile network. Figure 3(c) illustrates the data size of packets that include TCP handshake, TCP close, DNS lookup and HTTP request. The data size for HTTP requests was significantly reduced by 35% to 69% compared to the native case. In addition, the overall traffic except web contents was reduced by 53% to 79%.

Since our scheme removes static header fields in each HTTP request at the browser side and appends them at the proxy server side, the data traffic passing through the mobile network is reduced and the effectiveness of HTTP pipelining is increased. In case of the *cnnmobile* web site, the number of packets including HTTP requests was reduced from 23 to 18 due to the increased effectiveness of HTTP pipelining.

In case of the *weather* web site, our scheme did not reduce the number of packets for HTTP requests. The reason is that this web site has a few HTTP requests (6 requests in the replicated web sites). In addition, the web browser used four threads to fetch web objects concurrently. Accordingly, when an embedded object was found at the parsing phase, the HTTP request for the object was promptly submitted to the network stack in the OS without any queueing delay. Accordingly, the chance that HTTP pipelining occurs was low.

5. CONCLUSION

Web browsing on mobile network is sluggish in comparison to that on other wired or Wi-Fi networks because the RTT of mobile network is relatively long on mobile network and transferring many small packets for DNS lookups and TCP handshakes is required for a web page loading instance.

We proposed a two-tier web browser architecture that consists of a mobile web browser and a proxy server. 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 fields data is stripped off from the packets on mobile network.

Our evaluation showed that the proposed architecture greatly reduces both the number and size of mobile network packets, and in turn, improves the page loading delay along with the mobile network load.

6. REFERENCES

[1] E. Hernandez. War of the mobile browsers. *IEEE Pervasive computing*, pages 82–85, 2009.

[2] J. Huang, Q. Xu, B. Tiwana, Z. Mao, M. Zhang, and P. Bahl. Anatomizing application performance differences on smartphones. In *Proc. of MobiSys'10*, pages 165–178, 2010.

[3] J. Kim, R. Baratto, and J. Nieh. pTHINC: a thin-client architecture for mobile wireless web. In *Proc. of WWW'06*, pages 143–152, 2006.

[4] Z. Liu, Y. Saifullah, M. Greis, and S. Sreemanthula. HTTP compression techniques. In *Proc. of WCNC'05*, volume 4, pages 2495–2500, 2005.

[5] H. Shen, Z. Pan, H. Sun, Y. Lu, and S. Li. A proxy-based mobile web browser. In *Proc. of MM'10*, pages 763–766, 2010.

[6] Z. Wang, F. Lin, L. Zhong, and M. Chishtie. Why are web browsers slow on smartphones? In *Proc. of HotMobile'11*, pages 91–96, 2011.

[7] B. Zhao, B. Tak, and G. Cao. Reducing the delay and power consumption of web browsing on smartphones in 3G networks. In *Proc. of ICDCS'11*, pages 413–422, 2011.