Session Level Techniques for Improving Web Browsing Performance on Wireless Links

Pablo Rodriguez
Microsoft Research
Cambridge, UK
pablo@microsoft.com

Sarit Mukherjee
Bell Laboratories
Holmdel, NJ
sarit@bell-labs.com

Sampath Rangarajan
Bell Laboratories
Holmdel, NJ
sampath@bell-labs.com

ABSTRACT

Recent observations through experiments that we have performed in current third generation wireless networks have revealed that the achieved throughput over wireless links varies widely depending on the application. In particular, the throughput achieved by file transfer application (FTP) and web browsing application (HTTP) are quite different. The throughput achieved over a HTTP session is much lower than that achieved over an FTP session. The reason for the lower HTTP throughput is that the HTTP protocol is affected by the large Round-Trip Time (RTT) across Wireless links. HTTP transfers require multiple TCP connections and DNS lookups before a HTTP page can be displayed. Each TCP connection requires several RTTs to fully open the TCP send window and each DNS lookup requires several RTTs before resolving the domain name to IP mapping. These TCP/DNS RTTs significantly degrade the performance of HTTP over wireless links. To overcome these problems, we have developed session level optimization techniques to enhance HTTP download mechanisms. These techniques (a) minimize the number of DNS lookups over the wireless link and (b) minimize the number of TCP connections opened by the browser. These optimizations bridge the mismatch caused by wireless links between application-level protocols (such as HTTP) and transport-level protocols (such as TCP). Our solutions do not require any client-side software and can be deployed transparently on a service provider network to provide $30-50\%$ decrease in end-to-end user perceived latency and $50-100\%$ increase in data throughput across wireless links for HTTP sessions.

Categories and Subject Descriptors

C.2 [**Computer Systems Organization**]: Computer-Communication Networks; H.4.m [**Information Systems**]: Miscellaneous

General Terms

Performance

Keywords

Wireless, Web, Optimizations

1. INTRODUCTION

In current third generation wireless networks, the wireless links have very large and variable Round-Trip Times (RTTs) [10]. This is due to the need for buffering and retransmissions from the base station to the mobile node (MN) at the link-layer to compensate for packet losses [2]. Experiments conducted on deployed systems show that the RTTs experienced across the wireless link vary from 400 msec up to 1 sec. Because of this, user experienced throughput for specific applications is much lower than the maximum possible physical layer data rate. For example, with CDMA2000-1xRTT [1] physical layer, the maximum physical layer data rate is 153.6 Kbps. With this, the maximum TCP throughput (with protocol overhead) works out to be 128 Kbps. But our measurements have shown that for FTP, the throughput achieved in an unloaded CDMA2000-1xRTT cell is in the range of $100-120$ Kbps, and for HTTP the throughput is much lower and is in the range of $50-70$ Kbps. With FTP connections, the throughput does approach that of raw TCP as the connections are usually long-lived. But HTTP throughput is degraded mainly due to the following two reasons.

DNS Queries: Popular web pages usually contain several embedded objects hosted under different domain names. For example, sites such as www.weather.com, finance.cnn.com, etc. have embedded objects that point to many distinct domains. This behavior is seen even with URL-rewritten [25] pages where the embedded objects are rewritten to point to Content Delivery Network's (CDN) server. For example, the embedded URLs in the top level pages for Shari's Berries (www.berries.com) and Britannica (www.britannica.com), both of which are URL-rewritten, point to sixteen different Akamai domain names. The web browser performs DNS queries for these domain names, each of which incurs one to three seconds delay. On top of this, the time-to-live (TTL) parameter for DNS responses to the popular web sites is kept small so that DNS based load-balancing to one of multiple servers is possible [22]. With web sites that are served through CDNs, this is certainly a requirement so that the CDN service provider can redirect requests to an "optimal" server in their network. A smaller TTL suppresses the advantages of DNS caching and leads to the browser making very frequent queries to the DNS server to resolve domain names. For a better discussion on the overhead of DNS queries on Web traffic, please refer to [22, 16].

TCP Connections: The web browser at the MN opens at least one (possibly more) TCP connection to each domain name referred to by the embedded objects in a top level web page. Thus, even if the browser and the server support persistent connections (HTTP/1.0 keep-alive or HTTP/1.1), given that at least one persistent HTTP connection has to be opened to each distinct domain, if the number of distinct domain names that host the embedded objects is large, the number of TCP connections opened is also substantial.

The above behavior affects web browsing performance in wireline networks as well but in wireless networks, the effect is amplified due to the large and varying RTT across the wireless link. A

WWW2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-844-X/04/0005.

large RTT increases the delay incurred by DNS lookups; with very many DNS lookups per web page, this delay increase is substantial to affect the user perceived performance. A large RTT also leads to an increase in TCP connection establishment and the ramp up delay. Again, with the need to establish many TCP connections per web page, this affects user perceived performance. Thus, TCP setup delays of a large number of TCP connections and delays due to DNS queries can account for a significant overhead leading to decreased HTTP throughput and degraded user perceived performance. Notice that the FTP application, whose throughput is close to the theoretical maximum, performs only one DNS lookup for the server name and uses only one long-lived TCP connection to transfer the data.

The focus of this paper is to design solutions to mitigate the effect due to the above mentioned problems. We propose session level optimization techniques to enhance the current HTTP download mechanisms, to "mimic" the behavior of FTP *over the wireless link* to achieve better throughput. These techniques strive to (a) minimize the number of DNS requests made across the wireless links and (b) minimize the number of distinct TCP connections opened across the wireless links when web pages are downloaded. In other words, most of the DNS lookups and short-lived TCP connections are pushed to the wireline part of the network, making the wireless part behave like an FTP session. The solutions are HTTP standards compliant and do not require any changes to be made to either web clients, web servers or DNS servers. We propose how our solutions can be deployed transparently (to the web clients, the web servers and the DNS servers) on a service provider network and how they can gracefully handle client mobility. Through experiments we demonstrate that the solutions can provide $30-50\%$ decrease in end-to-end user perceived latency and $50-100\%$ increase in data throughput across wireless links for HTTP sessions.

We now discuss how our schemes fit within the realm of a multitude of solutions that have been proposed to improve data performance over wireless links. As shown in Figure 1, solutions have

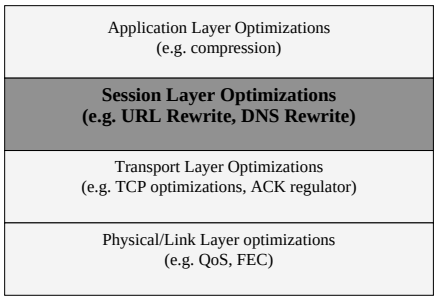

Figure 1: Different data optimization solutions.

been proposed at various levels of the protocol stack, such as the physical/link layer (MAC optimizations) [20, 6, 5], the transport layer (TCP optimizations) [3, 4, 13, 9, 10, 7, 23] and the application layer (data compression) [15, 14].

In the literature, physical/link/MAC layer enhancements have been proposed that aim to provide improved scheduling algorithms over the wireless links to increase the total system throughput [5, 6], provide fairness or priorities between the different users [20, 6], assure minimum transmission rates to each user [5] and incorporate forward error correction on the link to reduce retransmissions. The scheduling algorithms aim at controlling the system or user throughput at the physical layer.

For data applications, it is equally important to consider the data performance at higher layers in the protocol stack, especially at the transport (TCP/IP) layer. It has been observed that the round trip time for TCP packets can abruptly increase and lead to delay spikes (due to lower-layer retransmissions, channel condition changes, handoff delays or priority scheduling) over wireless links [10]. These delay spikes may cause TCP timeout, which triggers the congestion control mechanism in TCP, leading to a decreased TCP window size and consequently low throughput performance [10, 7, 23]. Techniques such as the ACK Regulator [10] or Flow Aggregation [9] have been proposed to minimize the impact of burstiness in TCP. The motivation for these solutions is to increase fairness, and avoid buffer overflow and the resulting congestion avoidance mechanism of TCP.

At the application layer several data compression techniques have been proposed [15, 14, 12] to increase the effective throughput of wireless links. Examples include degrading the quality of an image, reducing the number of colors, compressing texts, etc. To overcome some of the application-level performance problems of HTTP in wireless links, several proposals have suggested the use of a special client-side software to implement new wireless specific protocols [18, 8] or client-side includes to minimize the amount of data sent over the last mile [21]. Other application-level optimizations try to improve how application-level protocols such as HTTP perform in these wireless links. Examples include the use of HTTP1.1 request pipelining [8].

The techniques proposed in this paper can be thought to fall between application layer optimizations and transport layer optimizations and hence can be categorized as session layer optimizations (see Figure 1). These solutions are independent of optimizations at other layers and are complementary to those solutions. To the best of our knowledge, this is the first research work that proposes to enhance web browsing performance over wireless link using *session layer techniques* without adding any client and/or server side components.

The rest of the paper is organized as follows. The next section explores some obvious solutions to the HTTP throughput degradation problem across wireless links and discusses their shortcomings. Section 3 describes our session level optimization techniques. Section 4 discusses experimental results that illustrate the benefits of these optimization techniques. Numerical results show the improvements achieved by these techniques on user perceived delay and throughput during HTTP downloads. Section 5 discusses how the proposed scheme works with client mobility. Conclusions are presented in Section 6.

2. POSSIBLE TECHNIQUES FOR SESSION LEVEL OPTIMIZATIONS

In the previous section, we identified TCP setup delays and delays due to DNS queries as two major sources of decreased HTTP throughput and increased user perceived response times. Before we discuss our session level optimization techniques, we consider other possible obvious solutions to solve these problems and explain their shortcomings.

2.1 Explicit Proxy Configuration

The web browsers can be configured explicitly to point to a proxy cache which is on the wireline network. With such a configuration, (a) once a DNS lookup is performed to map the proxy's domain name to an IP address, no more DNS lookups are needed (as long

as the DNS cache at the MN does not time out; this timeout can be made large using a large TTL for the DNS entry). The DNS lookups required to identify the IP addresses of the domains that host the embedded objects are now pushed to the proxy which will perform these operations over the wireline network, and (b) the browser needs to open TCP connections only to the proxy. With support for persistent connections, only one or a few (for parallelism) TCP connections will be opened and these will be kept persistent over multiple top level web page downloads as well as embedded object downloads. The overhead of opening multiple TCP connections to different domains is now pushed to the proxy.

However, explicit proxy configuration is not a viable practical option as service providers *must* setup and maintain client's browser settings to point to the proxy. This is a management overhead that the service providers are not willing to take up. Another concern with an explicit proxy configuration is that this provides the user the flexibility to reconfigure the proxy setup in the browser and not go through the service provider's proxy. This is also a security concern because the service providers implement URL filtering and other security mechanisms at the proxy and being able to bypass the proxy defeats these security mechanisms. One possibility is for the browser to automatically detect and configure a proxy. But there is no standard solution for automatic proxy configuration.

The most common approach for request redirection is the use of a transparent proxy. With this approach, client traffic is transparently redirected to a proxy using a Layer 4 switch [19]. More than 90% of all proxy deployments happen in transparent mode [11]. But redirecting traffic to a transparent proxy does not solve the aforementioned problems due to TCP setup and DNS lookups. With a transparent configuration, the browser still thinks that it is directly connecting to a server and hence performs all the DNS lookups that it usually performs and opens the same number of TCP connections as it would without a proxy; now, all these TCP connections are terminated at the transparent proxy without the browser's knowledge, but the TCP setup delay across the wireless link is still the same.

Our solutions discussed in Section 3 permit the deployment of transparent proxies (i.e., no client configuration required), while enjoying the benefits of an explicit proxy configuration, e.g., no DNS lookups and few TCP connections.

2.2 Bundling Content

Another solution is to bundle content at the server and ensure that all embedded objects in a single top-level page are downloaded in a single file. The content could either be bundled at the server (which is not as efficient, as different domains host different embedded objects, and only content hosted within a domain can be bundled) or a web proxy can pre-fetch all objects within a web page, create a single large file, and transfer it to the web browser. The browser needs to break up the file into individual objects before displaying them. The goal here is to download one single file that carries all the embedded objects and thereby try to achieve a throughput performance across a wireless link that matches FTP performance.

There are two main problems with this approach. The first is that traditional proxies are not able to bundle content. Therefore a proxy that can bundle contents of a web page has to be built and deployed. Second, content bundling requires installing a client side component to break up the page into its individual components before passing them to the browser for display.

Both the aforementioned possible solutions can be used to enhance HTTP downloads. However, the solutions are not very practical. The goal of the session level optimization techniques presented in this paper is to ensure that the web browser fetches all embedded objects from a single proxy without the need to explicitly configure the browsers to point to the proxy. At the same time, our solutions will have the same benefits as an explicit proxy solution in that only the domain name of the proxy is looked up at the DNS server and only one or a few TCP connections to the proxy is opened which is then reused for multiple downloads. The solutions work with any standard browser (e.g. Netscape, Internet Explorer) and does not require any client-side modification. The next section details the session level optimization techniques.

3. SESSION LEVEL OPTIMIZATION TECHNIQUES

In this section, we discuss two different solutions for session-level optimization. One solution is based on URL rewriting and the other is based on DNS response rewriting. Each of these solutions has its own advantages and disadvantages. We discuss them qualitatively after presenting the solutions. In the remainder of the paper, the terms Mobile Node, Mobile/Wireless Client and Mobile User are used interchangeably.

3.1 URL Rewriting

Currently, some Content Distribution Network (CDN) service providers use URL rewriting to redirect requests for embedded objects to servers in the CDN [25]. The CDN service providers rewrite content on the origin servers by prefixing the URLs that refer to the embedded objects with the domain name of the CDN [25]. The browser gets a top level page from the origin server but because the embedded object URLs are rewritten to point to the CDN, to fetch the embedded objects, the browser sends DNS requests to resolve domain names within the CDN domain. These requests are resolved by a DNS server in the CDN network which returns IP addresses of servers in the CDN network. The embedded objects are then fetched from these servers.

The URL rewriting technique that we propose for session level optimization is quite similar to URL rewriting performed by CDN service providers except that the URL rewriting is performed closer to the client by a *URL rewriting proxy*. Further, instead of prefixing the URLs with domain names, the URLs are prefixed with the IP address of a caching proxy on the wireline network. The URL rewriting mechanism works as follows. When the browser sends a request to a top level page, the request as well as the response from the origin server are intercepted transparently by the URL rewriting proxy. The response is parsed by the URL rewriting proxy which rewrites the URLs of the embedded objects by prefixing them with the IP address of a *caching proxy*. The URL rewriting proxy and the caching proxy could be co-located or could be different entities on different machines.

Figure 2 shows an example of this process. Assume that the browser retrieves the top level page from www.foo.com. The URL rewriting proxy transparently intercepts this page and prefixes the embedded URLs with the IP address of the caching proxy (which is 10.0.0.12). For example http://i.cnn.net/images/plane.jpg is changed to http://10.0.0.12/i.cnn.net/images/plane.jpg. When the browser is required to fetch this embedded object, it opens a TCP connection to 10.0.0.12 and requests the URL i.cnn.net/images/plane.jpg. This is similar to a request that would be sent by the browser if it had been explicitly configured to connect to the caching proxy. The caching proxy connects to i.cnn.net to retrieve /images/plane.jpg or serve the object if it is locally available. Note that no DNS requests are made by the browser during this process as the IP address is prefixed to the embedded URLs. The only DNS request made is the one to resolve the domain name of the server that hosts the top level page. The DNS request for i.cnn.net is made by the caching

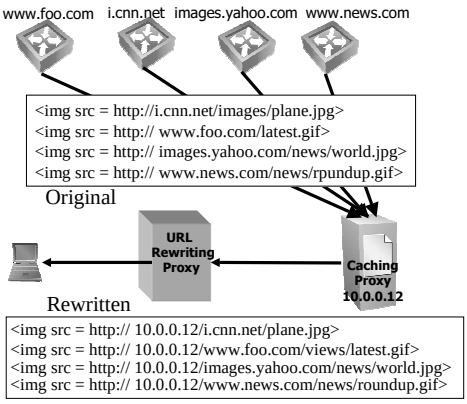

Figure 2: Example of URL rewriting.

proxy, if need be, over the wireline network.

Once a TCP connection is established to 10.0.0.12, the browser uses this connection to retrieve other embedded objects (i.e., the gif and jpg images as shown in Figure 2). Other top level pages are rewritten to prefix embedded URLs with the same IP address and thus more objects are retrieved through the same connection until the connection is teared down for some reason; thus TCP connection setup across the wireless link is restricted to only one (or a few if connections are opened in parallel) TCP connection to the caching proxy. As evident from the description, with URL rewrite, all the embedded objects in all top level pages from all web sites come from the same caching proxy.

3.2 DNS Rewriting

This mechanism for session level optimization rewrites the DNS responses to point to the caching proxy. When the browser makes a DNS request for a domain name (both to fetch top level pages as well as embedded objects within the pages) the DNS responses are intercepted by a DNS rewriting proxy. The DNS rewriting proxy and the caching proxy could be co-located or could be different entities on different machines. A DNS response may contain a list of IP addresses. The IP address of the caching proxy is added to the top of the list (so that this is the first IP address that the browser tries to connect to) and the original IP addresses in the list (that point to origin server IP addresses) are left as they are. This is done so that if the wireless client has roamed out and is not able to connect to the caching proxy, it can try to connect to one of the origin server IP addresses. We will consider in more detail the impact of client mobility in Section 5.

At the same time, the time-to-live (TTL) for the caching proxy IP address is set to a large value so that this is cached at the client for a reasonably long interval. When the browser receives the rewritten DNS response, it connects to the IP address of the caching proxy to retrieve the web pages. Because all DNS responses are rewritten to add the IP address of the same caching proxy to the top of the list, the browser opens a TCP connection to this caching proxy and reuses this connection to fetch multiple top level pages and all embedded objects contained within them. Note that unlike the URL rewriting mechanism, with DNS rewriting, DNS lookups are made for the domain names of the embedded URLs, but these are made only once and are cached and reused.

When DNS responses are rewritten, a question arises as to which DNS responses should be rewritten. DNS requests do not carry TCP port numbers and hence it is not possible to identify DNS requests that correspond to HTTP requests from those that correspond to other applications such as FTP and telnet. We suggest that the DNS rewriting proxy consult a pre-configured list of domain names (similar in concept to lists used for applications such as content filtering [24]) to decide which DNS responses should be rewritten. Only if a domain name is found in this list will the corresponding DNS response be rewritten. In addition DNS responses for any domain name that starts with a "www" may also be rewritten.

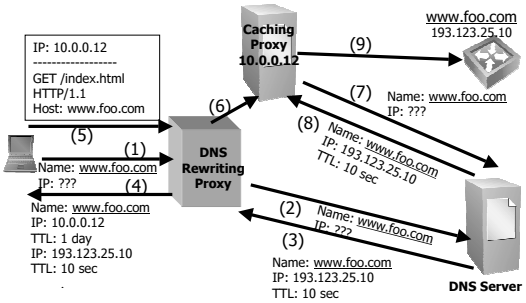

Figure 3: Example of DNS rewriting.

Figure 3 illustrates the DNS rewriting process where the browser retrieves http://www.foo.com/index.html. The browser makes a DNS request to the DNS server to resolve www.foo.com. The DNS server responds with the IP address 193.123.25.10 with a TTL of 10 seconds. The DNS rewriting proxy intercepts this response and adds the IP address 10.0.0.12 (which is the IP address of the caching proxy) and sets the TTL for this entry to be 1 day. The original IP address is left as is. When the browser receives this response, it makes a request to 10.0.0.12 to fetch /index.html. The `Host` header in the HTTP request contains the original domain name www.foo.com. The caching proxy will connect to www.foo.com to retrieve /index.html and send it to the client or serve the object to the client if it is available locally. As shown in the figure, when the caching proxy makes a DNS request to the DNS server to resolve the domain name, the DNS response does not go through the DNS rewriting proxy and is not rewritten. Only DNS requests from the MN will lead to DNS responses being rewritten to add the IP address 10.0.0.12 and so the MN will fetch all objects through the caching proxy and can use a persistent connection (or a few connections) to the caching proxy to fetch all the objects.

3.3 Comparison of Different Techniques

Table 1 compares the two flavors of session level optimization techniques (URL Rewriting and DNS Rewriting) with Explicit Proxy and Content Bundling techniques discussed earlier. The two major issues that favor URL rewriting and DNS rewriting over the other two techniques are the first two shown in the table.

4. EXPERIMENTAL RESULTS

	Explicit Proxy	URL RW	DNS RW	Content Bundling
Free from browser configuration	No	Yes	Yes	No
Client-side component not required	Yes	Yes	Yes	No
Works with legacy caching proxies	Yes	Yes	Yes	No

Table 1: Comparison of different techniques.

We implemented prototypes of the proposed session level optimization techniques in Linux and conducted some controlled experiments to measure the quantitative benefits of the proposed techniques. In this section we present the experimental setup and the summary of the results obtained from the experiments we conducted.

4.1 Experimental setup

The experimental setup we used is shown in Figure 4. There are six components to the experimental configuration.

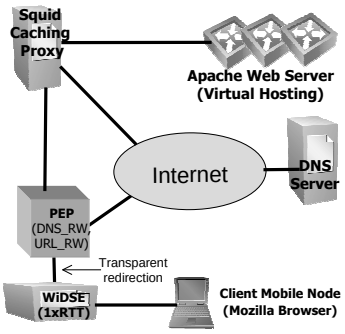

Figure 4: Experimental setup.

(1) Performance Enhancing Proxy (PEP): We built the DNS rewriting proxy and the URL rewriting proxy on Linux. Both of them are co-located in a machine. We use the term Performance Enhancing Proxy to refer to these proxies. Of course, both proxies will not be active at the same time; experiments with the DNS rewriting proxy and URL rewriting proxy are performed independently.

(2) Squid Caching Proxy: We used the Squid Proxy as the caching proxy. In the case of URL rewriting, the embedded URLs are rewritten by PEP to point to this proxy. Similarly in the case of DNS rewriting, the DNS responses are rewritten by PEP to point to this proxy.

(3) Apache Web Server: To perform the experiments with specific web pages in a controlled environment, the top level pages and the embedded objects in them from the web sites were copied to a local machine running apache web server. To run one set of experiments, the top level pages from the top 100 sites were copied (as determined by www.hot100.com). To run another set of experiments, the top level pages from Yahoo, CNN and Britannica were copied. The statistics for these sites are:

- Yahoo (www.yahoo.com): It has 16 embedded objects hosted in 3 different domains. The size of the page is 74 KB. This constitutes a typical web site with small number of domains.
- CNN (www.cnn.com): It has 58 embedded objects hosted in 6 different domains. The size of the page is 197 KB. This constitutes a typical web site with medium number of domains.
- Britannica (www.britannica.com): It has 32 embedded objects hosted in 15 different domains. The size of the page is 178 KB. This constitutes a typical web site with large number of domains.

In order to focus specifically on the wireless link delay characteristics, we downloaded and hosted all necessary objects on this web server located within our experimental setup. We also hosted our own DNS server with all necessary records to reproduce the exact setup of the target Web pages. The Apache Web Server was made to host the above three top-level domains as well as the domains that host the embedded objects contained within these top-level pages. The virtual hosting feature of the web server was used to accomplish this configuration. Each virtual host is assigned a different virtual IP address on this web server. The Squid Proxy retrieves the top-level pages as well as the embedded objects contained within these pages from this web server rather than from origin servers on the Internet.

(4) DNS Server: This is the DNS server to which DNS requests from both the browser and the Squid Proxy are made. When a request to fetch a top-level page is made, the DNS server is made to return the (virtual) IP address of the web server in response to requests to resolve these domain names. The DNS response from this server to the browser (but not the response to the Squid Proxy) is intercepted by the DNS rewriting proxy and rewritten to add the IP address of the Squid Proxy.

(5) Wireless Data Service Emulator (WiDSE): WiDSE [17] is a software emulator developed by Lucent Technologies to very accurately emulate a CDMA2000-1xRTT [1] airlink. The emulation environment supports multiple mobile users that connect to the emulator using Ethernet and uses data captured over Ethernet. It allows for error rates in fundamental and supplemental channels in both forward and reverse directions to be configured. It also allows for Radio Link Protocol (RLP) [2] retransmissions to be controlled. Further, base station scheduling is emulated. The WiDSE runs on a Linux PC and connects two LANs, one of which is the mobile LAN (the mobile users on the wireless link) and the other is the network LAN (wireline network). Ethernet packets from/to the mobile LAN to/from the network LAN are captured at the WiDSE machine and the entire protocol process from the mobile node to the PDSN (the Packet Data Service Node in a CDMA2000 network which is the gateway between the radio network and the IP network) is emulated. More details on WiDSE are beyond the scope of this paper. We use a different parameter setup on WiDSE to emulate different wireless link and background traffic load behaviors.

(6) Client Mobile Node: We use a custom instrumented Mozilla browser from the client to conduct the experiments. We use this particular browser for the availability of its source code. We instrumented the browser to measure and print out the relevant statistics (e.g., number of TCP connections and DNS requests made, page download time, etc.). The browser supports persistent connections but not request pipelining[1].

[1]We experimented with other popular browsers like Internet Ex-

In a service provider deployment, HTTP sessions and DNS requests and responses from the client could always go through the PEP (i.e., when PEP is co-located within PDSN) or could be transparently redirected to the PEP using techniques like Layer 4 switching [19] (i.e., when PEP is a separate network element). Note that secure HTTP sessions (which use port 443) are not redirected to the PEP cache and remain unaffected. With this setup, the web page download is as follows:

URL rewriting: For this set of experiments, the URL rewriting proxy of the PEP is activated. DNS requests for the domain names of the top level pages from the browser are responded to by the DNS server with the IP address of the Apache Web Server. The browser then makes HTTP connections to this web server to fetch these top level pages and these requests are transparently intercepted by the PEP. PEP forwards these requests to the Squid Proxy. The Squid Proxy delivers the top-level pages (after fetching them from the Web server if they are not locally cached) to the PEP which then rewrites the embedded URLs with the IP address of the Squid Proxy. The browser then fetches all the embedded objects from the Squid Proxy by opening one or a few TCP connections to it thereby emulating an explicitly configured proxy (these requests and responses transparently pass through the PEP as well). No more DNS requests are made over the wireless link. Of course, the Squid Proxy, if it does not have the requested object, will retrieve it from the web server and issue the required DNS request(s) to the DNS server.

DNS rewriting: For this set of experiments, the DNS rewriting proxy of the PEP is activated. DNS requests for the top level pages are made from the browser to the DNS server (transparently) through the PEP. The DNS responses are intercepted and rewritten by the PEP to include the IP address of the Squid Proxy. The browser then makes a TCP connection to the Squid Proxy to fetch the top level pages (these requests and responses transparently pass through the PEP). Following that the browser makes DNS requests to resolve the domain names of the embedded objects, again (transparently) through the PEP. The responses to these requests are also rewritten by the PEP to include the IP address of the Squid Proxy. The browser then fetches the embedded objects from the Squid Proxy over the same TCP connection(s) it had opened earlier to the Squid Proxy.

One point to note here is that with URL rewriting, DNS requests are made from the browser over the wireless link to resolve only the top level domain names. With DNS rewriting, DNS requests for the domain names that host the embedded objects are made once over the wireless link as well, but because the TTL for the DNS responses is made large, the DNS responses are cached at the browser for a long period of time and further DNS requests are minimized. Also note that the total number DNS requests made to the DNS server remains the same. But with session level optimization most of the request-response is pushed to the wireline network, which reduces the delay observed over the wireless link.

For clarity, in the description above, we describe the PEP and the Squid Proxy as two separate devices. In our experiments, we co-located the PEP and the Squid cache in the same device.

4.2 Results

With the above experimental setup, we measured three different performance metrics with and without session level optimization techniques. The results from these experiments are detailed below.

plorer and Netscape, none of which seem to support request pipelining.

4.2.1 Number of TCP Connections and DNS Requests

For this set of experiments, we copied the top level pages of the top 100 URLs (as determined by www.hot100.com) as well as the embedded objects on these pages and configured the web server in our setup to deliver these objects. The browser was instrumented to sequentially request this set of top 100 pages multiple times (20 in our experiment). Each time the set of pages was retrieved, we measured the total number of TCP connections established and the total DNS requests made. The results averaged over the 20 runs with and without session level optimizations are shown in Figure 5.

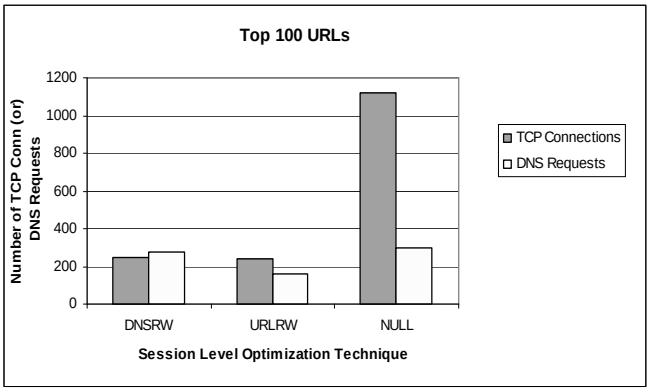

Figure 5: Number of TCP connections and DNS requests made with and without session level optimizations.

Consider the number of TCP connections without session level optimization (referred to as NULL). The number of TCP connections made is around 1110, which translates to an average of 11 connections per top level web page. During these experiments, the browser and the server were configured to keep persistent connections. However, since the embedded objects were hosted on different domains, the browser still had to make multiple connections to fetch a top level page. With URL rewriting, the browser makes individual TCP connections to fetch the 100 top level pages; thus at least 100 TCP connections will be made to the web server (which are actually transparently redirected to the caching proxy). However, on top of this, a little more than 100 extra TCP connections are made to fetch the embedded objects directly from the caching proxy. With DNS rewriting, the results are quite similar. Now, the 100 top level pages as well as the embedded objects are all retrieved directly from the caching proxy. However, due to the need for parallelism, a little more than 200 connections are used. Of course, if the requests had been completely serialized, only one extra TCP connection (which is now explicitly to the caching proxy) should be needed. But browsers tend to open extra connections for parallelism especially if an existing TCP connection is already in use. This is the reason for the extra connections.

The number of DNS requests without session level optimizations is around 300. This translates to an average of 3 different domain names per web page. With DNS rewriting, the number of DNS requests is about the same as the number of DNS requests for the NULL case since all domain names need to be resolved. With URL rewriting, the number of DNS requests is close to the number of top level pages, i.e. 100, and much lower than for DNS rewriting or for the NULL case. This is because with URL rewriting the browser performs DNS requests only for the top-level pages. It does not

perform any request for the embedded objects since all embedded objects are fetched from the same source (i.e. the caching proxy IP address).

Observe from the figure that both the number of TCP connections and the number of DNS requests are reduced with session level optimization. The reduction is more significant in the case of TCP connections.

4.2.2 Response Time

For this experiment, the Mozilla browser was instrumented to compute the time between the sending of the request for the top level page and the complete display of the page (including all embedded objects). We refer to this as the user perceived *response time* for the page. We measured this response time at the browser to download three popular top level pages (Yahoo!, CNN and Britannica) and the embedded objects contained in them. Each of these pages has different characteristics regarding the number of embedded objects on the page, the number of distinct domains that host the embedded objects, and the total size of the page.

The WiDSE emulator can be configured to provide different cell characteristics and background traffic load by tuning the forward and reverse error rates on the fundamental and supplemental channels, and the number of retransmissions on the radio-link. We configured these parameters to emulate two cells with different per user throughput and delay. As explained earlier, the maximum FTP throughput that we observed in an unloaded cell is in the range of $100 - 120$ Kbps. Using this as a yardstick and based on our own measurements in a live CDMA2000-1xRTT network, an average cell was configured to have an average per user FTP bandwidth of around 78 Kbps and an average round-trip time from the browser to the web server (within our controlled environment) of 400 msec. A congested cell was configured to have an average per user FTP bandwidth of around 56 Kbps and an average round-trip time from the browser to the web server of 600 msec. Note that the congested cell represents a scenario where the number of background users increases significantly, emulating the peak hour characteristics seen by a single user in a deployed cell.

The results are shown in Figure 6. The response times were measured and averaged over 20 downloads of each of the top level pages and the corresponding embedded objects. In both cell types, response times for CNN and Britannica are much higher than that of Yahoo!. This is because Yahoo! has lot fewer domain names and embedded objects than the other two. Among CNN and Britannica, CNN has more embedded objects but fewer domains. The time required for making DNS requests balances out with the time required to download the embedded objects. This results in similar response time for them. In all cases, the response time with session level optimization is much smaller than without the optimization. A mean decrease of around 30% is seen in the case of an average cell and a decrease of around 50% is seen in the case of a congested cell. The standard deviation was less than 10% for the average cell and less than 20% for the congested cell. This decrease is an artifact of saving several RTTs during the web page download. This is why the decrease is much more prominent in the case of a congested cell which has a higher RTT across the wireless link.

4.2.3 Throughput

We calculated the throughput for HTTP downloads and compared them with that for FTP. The results are shown in Figure 7.

Firstly, from Figure 7 (a) observe that the achieved throughput for all the three downloaded web sites was around 35-50% higher with both URL rewriting and DNS rewriting compared to the NULL case. Secondly, when we compare the HTTP download throughputs with FTP, we find that Yahoo! achieves a throughput that is closest to the FTP throughput. The reason for this is that in our experiments, although both the browser and server use persistent connections, pipelining is not enabled (not supported by the browser). Because of this, although multiple objects are downloaded through the same TCP connection, an object has to be fully downloaded before a request for another object is sent on the same connection. This introduces a RTT delay during which the connection is idle. With Yahoo!, this effect is minimal because the number of embedded URLs in the top level page for Yahoo! are not very many and hence the number of such idle times are small. The other two web sites have many more embedded objects and hence this effect is more pronounced.

In order to justify the above conjecture, we estimated the throughput that could have been achieved with CNN if we had request pipelining. At the browser, we observed an average of 4 simultaneous TCP connections to download embedded objects. Therefore, each persistent TCP connection is used to download roughly 15 objects. This results in each connection being idle for about 14 RTTs, or 14 x 400 msec = 5.6 sec. When we subtract the idle time from the observed response time and compute the throughput, it turns out to be around 73 Kbps, which is very close to the FTP throughput.

The results with a congested cell are even better. From Figure 7 (b) observe that the throughput achieved using URL rewriting and DNS rewriting is more than double compared to the NULL case. Further, with URL rewriting, the throughput for Yahoo! is almost the same as FTP throughput. The throughputs for the other two sites are further away from the FTP throughput for the same reasons as above.

5. EFFECT OF USER MOBILITY

We now consider the effect of mobility on the URL rewriting and DNS rewriting techniques under different scenarios. For this discussion, assume that the PEP and the Caching Proxy are co-located (as it would normally be the case in a real deployment). We refer to them as PEP since there is no ambiguity. Also recall that HTTP requests are transparently intercepted by the PEP, i.e., a Layer 4 switch transparently redirects all HTTP requests (i.e., port 80) to the PEP.

To study the impact of mobility we consider a mobile user moving from its *current region* to a *new region*, where "region" refers to an area served by a single PEP infrastructure (Layer 4 switch plus a PEP or a farm of PEPs). Such a definition is independent of whether mobility takes place within a single service provider or between different service providers. When a user moves from the current region with PEP service to a new region with PEP service, this means the user requests in the new region are serviced by a PEP infrastructure that is different from the one in the current region.

There are three interesting scenarios to consider: **(A)** PEP service is not available in the current region but is available in the new region, **(B)** PEP service is available both in the current and new regions and, **(C)** PEP service is available in the current region, but not in the new region. For these scenarios we address the following issues:

- In the case of URL rewriting, what is the effect of rewriting the links to embedded objects in a top-level page with the IP address of a specific cache? When a user either moves from one region to another in the middle of a page download, or reloads the top-level page from the browser cache after having moved, what if this cache is inaccessible? Will the retrieval of embedded objects fail in this case?

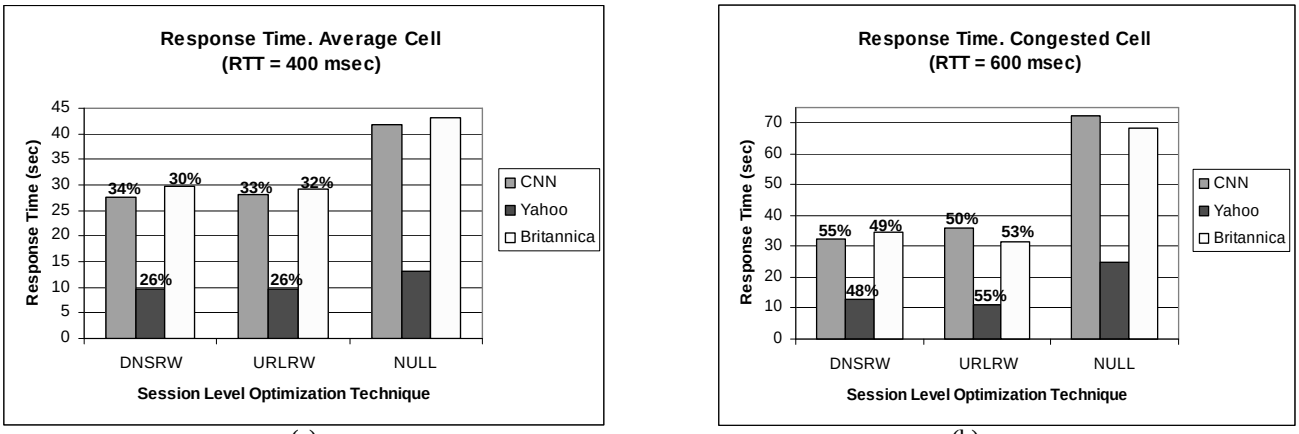

Figure 6: Response time in different types of cells. The numbers on the bars show the percentage decrease compared to the NULL case.

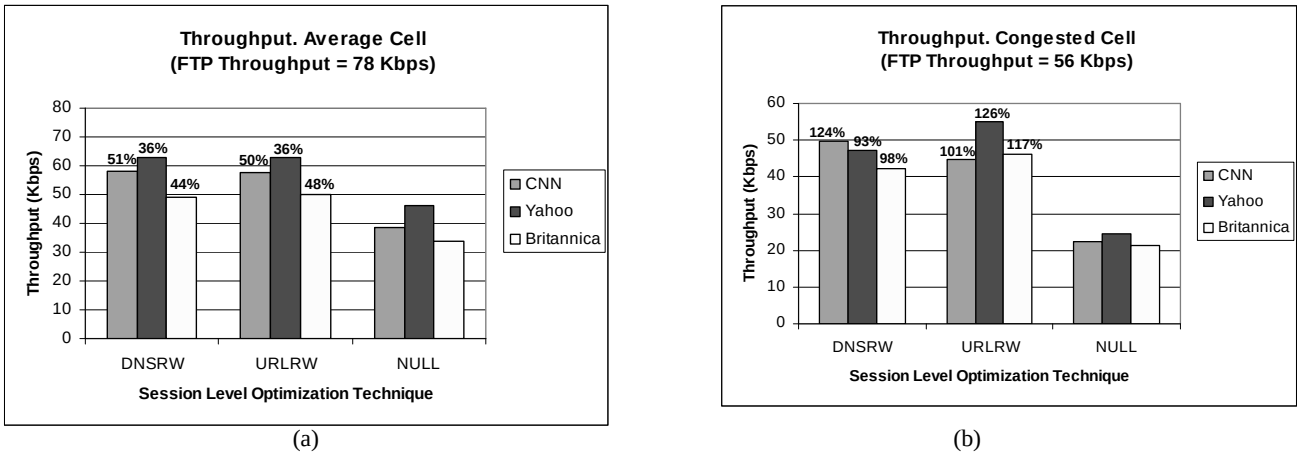

Figure 7: Throughput in different types of cells. The numbers on the bars show the percentage decrease compared to the NULL case.

- In the case of DNS rewriting, what is the effect of DNS caching at the client? Given that the IP address of a specific cache is returned to the client in response to domain name requests (both for top-level pages and embedded objects) what if the user moves and this cache is inaccessible? Because of DNS caching, the client will still try to fetch objects from this cache. Will these requests fail?

As we discuss next, both URL and DNS rewriting handle user mobility in a graceful manner. We point out there relative advantages and disadvantages under each scenario.

5.1 PEP service not available in the current region but available in the new region

Under this scenario only new HTTP session requests (TCP SYNs) are serviced by the PEP infrastructure in the new region while existing HTTP sessions initiated in the current region go directly to the origin server. This is done by the Layer 4 switch to redirect HTTP traffic to the PEP only if a TCP connection state already exists for a packet. Thus existing TCP connections from the current region are used to complete all unfinished downloads from the origin server. New connections in the new region benefit from session level optimizations.

The impact on the efficiency of the browser cache depends on the actual session-level optimization used. While in the current region, the browser cache indexes objects based on their domain names as the key (e.g., www.foo.com/image.gif). With URL rewriting in the new region, if the client refreshes a top-level page [2] the same embedded objects can now be referred to with a different URL (e.g., 10.0.0.12/www.foo.com/image.gif). This will cause the browser to fetch some embedded objects even if they are cached locally, though under a different name.

There is no impact on the browser cache with DNS rewriting. Since DNS rewriting only changes the mapping between a given domain name and its IP address, and does not require any changes to the embedded URLs, the browser cache does not get affected as the user moves and DNS rewriting becomes active.

5.2 PEP service available both in the current and new regions

In this scenario the service remains uninterrupted both with URL

[2]This could happen if the top-level page has expired and need to be refreshed, for example in case of a dynamic top-level page.

rewriting and DNS rewriting. We discuss these issues below in more detail.

5.2.1 URL rewriting

As the user moves, new connections for web pages get serviced by the new region. If the user moves from the current region to the new region in the middle of an object download, then two cases arise: (i) If the current PEP is accessible from the new region, then the existing TCP connections will still be serviced by the current PEP while new connections will be serviced by the new PEP. This situation is similar to Scenario A. (ii) If the PEP in the current region is not reachable from the new region, then the new PEP will reset the TCP connection and the browser will automatically open a new one to fetch the object.

The impact on browser caching in this scenario is minimal. Suppose the current PEP has rewritten the embedded object URLs with an IP address prefix of 10.0.0.12, and so the browser cache contains objects with the same prefix (e.g., 10.0.0.12/www.foo.com/image.gif). If the client now refreshes the top-level page in the new region (due to the same argument as in the previous section), two cases can arise depending on the IP address prefix used by the new PEP. Usage of the same IP address (e.g., 10.0.0.12) results in browser cache hit. If the IP addresses are different (e.g., 10.0.0.1 is used by the new PEP) there is a browser cache miss for the object (e.g., 10.0.0.1/www.foo.com/image.gif), and the object is fetched even though it exists in the cache with a different key.

Usage of the same IP address technically poses no problem. Consider Figure 2 where the links to embedded objects are rewritten to point to IP address 10.0.0.12, which is the same as the caching proxy. However, note that the IP address used to rewrite embedded object URLs does not have to match the caching proxy's IP address. As a matter of fact, it could be *any* valid IP address. When HTTP requests are made to the rewritten IP address, they are transparently redirected to the PEP based only on the *destination TCP port number* and not on the IP address. Therefore for the scheme to work, a PEP can use a (virtual) IP address of its choice to rewrite top-level pages. However, to improve the hit rate in the browser cache, it is recommended that the same IP address is used by all the PEPs. This requires pre-configuration of all PEPs to use the same IP address to rewrite embedded object URLs.

5.2.2 DNS rewriting

The above discussion applies to DNS rewriting as well. In Figure 3, it is shown that in response to a DNS request for www.foo.com, the IP address 10.0.0.12 is returned to the client. However, any other IP address could have been used. Assume that the same IP address is used by all PEPs to perform DNS rewriting (e.g. 10.0.0.12). DNS requests for both top-level pages and embedded objects within www.foo.com will return the same IP address. When the browser makes requests to 10.0.0.12 to fetch the top-level page and the embedded objects, these requests will be transparently redirected to the PEP. When the client moves to a new region, new DNS responses will be rewritten with the same IP address 10.0.0.12. Moreover, requests for domain names previously accessed will again be sent to 10.0.0.12. These requests will now be redirected transparently to a new PEP in the new region; thus the mobile client will not perceive any service disruption. If the client moves in the middle of an object download, the new PEP will reset the connection and the browser will automatically open a new connection to fetch the missing objects through the new PEP.

The effect of DNS rewrite on browser cache is same as in Scenario A and is not discussed further.

5.3 PEP service available in the current region but not available in the new region

In this scenario, a user moves from a region where PEP service is available to a region where PEP service is unavailable (for example, from an ISP who provides PEP service to another who does not). Let us consider the effect of both URL rewriting and DNS rewriting on embedded object retrieval in this scenario.

5.3.1 URL rewriting

Consider the situation where a user moves from a region with PEP service to a region without PEP service in the middle of a page download. Assume that the rewritten top-level page had been downloaded and the embedded objects are being downloaded. The requests to embedded objects from the new region will fail as the browser will try to fetch them from a cache IP address (say, 10.0.0.12). As the new region is unaware of the PEP service, there is no transparent redirection to a cache, and requests to this (virtual) IP address will fail. A similar situation occurs when a rewritten top-level page is retrieved from the browser cache after the user moves to a new region with no PEP service. The browser cache will try to fetch the embedded objects that have been rewritten using the 10.0.0.12 IP address, and unless they are locally cached as well, these requests will fail. If the top-level page itself is retrieved from the network after the mobile node has moved to the new region, all the operations will progress correctly; the top-level page and the embedded objects will now be fetched from the origin servers. However, similarly to Scenario A, the browser's cache efficiency will be reduced since existing cached objects are now referred under a different name.

One possible solution to prevent requests from failing when the browser cache tries to fetch objects with previously rewritten URLs (e.g. 10.0.0.12), is to pick a public and globally routable virtual IP address (e.g. 192.11.210.2) that is used by each PEP to rewrite embedded object URLs. This IP address would represent one or more caches in the current region's network that are globally reachable from any other region.

5.3.2 DNS rewriting

DNS rewriting is more resilient to this situation because of the following reason. Consider the situation where a user moves in the middle of a page download. The requests to retrieve embedded objects in the new region will either lead to a DNS request, in which case there will be no DNS rewriting and the objects will be fetched from the origin server, or a DNS entry will be found in the local DNS cache and the browser will make a request to the virtual IP address used by the PEPs. If the virtual IP address is globally routable and represents a globally reachable set of PEPs, then the client will be able to fetch the content without any problem. However, if the IP address is not reachable from the new region, then the initial request will fail, as it will be made to non-reachable PEP IP address. However, as described in Section 3.2, DNS rewriting also includes the IP address of the origin server as a secondary DNS entry. This allows the client to revert to the IP address of the origin server and fetch the content without suffering any disruption. To validate this scenario we studied the behavior of the most popular browsers under such a situation. We found that upon failure of a given DNS record IP address, the browser will make requests to the subsequent IP addresses in the list until one succeeds. Because of this feature, the browser will make a request to the next IP address in the list which will be the IP address of an origin server.

To summarize, both URL and DNS rewriting can handle mobility very gracefully when a user moves in regions with PEP service, avoiding any service disruption and maintaining the efficiency of

the browser cache. When a user moves between regions where one has PEP service and the other does not, DNS rewriting can handle mobility in a very elegant manner, with no service disruption or cache impact. However, URL rewriting may lead to higher browser cache miss. This is a minor problem and can be resolved with some of the solutions mentioned above. In case of deployment, we believe that the choice of a particular session-level optimization technique will be greatly influenced by the device that would implement the scheme. For example, a router or GGSN/PDSN may be better suited to implement DNS rewriting since it does not parse HTML pages, while a caching proxy may be better suited to implement URL rewriting.

6. CONCLUSIONS

In this paper, we presented two session level optimization techniques, namely URL and DNS rewriting, to enhance the performance of HTTP downloads over wireless links. Two major issues affect this performance and both of them are dependent on the large and variable round-trip time over the wireless link. One is the TCP setup delay due to multiple connections being initiated from the browser over the wireless link to different servers to download a single web page (and the embedded objects contained within it). The other is the DNS lookup delay due to multiple DNS lookups being performed over the wireless link. The techniques presented in this paper overcome these problems by (a) minimizing the number of DNS lookups over the wireless link and (b) minimizing the number of TCP connections opened by the browser. These techniques do not require any client-side software and can be deployed transparently on a service provider network. Experimental results based on a prototype implementation of the techniques show a 30-50% decrease in end-to-end user perceived latency and 50-100% increase in data throughput across wireless links for HTTP sessions.

7. REFERENCES

[1] T. G. P. P. 2. 3GPP2 — Developing the Next Generation of CDMA2000 Wireless Communications. http://www.3gpp2.org/.

[2] 3rd Generation Partnership Project 2. Data Service Option for Spread Spectrum Systems: Radio Link Protocol Type 3. http://www.3gpp2.org/Public_html/specs/C.S0017-0-2.10_v2.0.pdf, Aug. 2000. 3GPP2 C.S0017-0-2.10 v2.0.

[3] A. Bakre and B. Badrinath. Handoff and System Support for Indirect TCP/IP. In *Proc. of Second Usenix Symposium on Mobile and Location-Independent Computing*, Apr. 1995.

[4] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz. Improving TCP/IP Performance over Wireless Networks. In *Proc. of ACM Mobicom*, Nov. 1995.

[5] P. Bender, P. Black, M. Grob, R. Padovani, N. Sindhushayana, and A. Viterbi. CDMA/HDR: A Bandwidth-Efficient High-Speed Wireless Data Service for Nomadic Users. *IEEE Communications Magazine*, July 2000.

[6] P. Bhagwat, P. Bhattacharya, A. Krishna, and S. Tripathi. Enhancing Throughput over Wireless LANs using Channel State Dependent Packet Scheduling. In *Proc. of IEEE Infocom*, Mar. 1996.

[7] K. Brown and S. Singh. M-TCP: TCP for Mobile Cellular Networks. *ACM Computer Communications Review*, 27(5), 1997.

[8] R. Chakravorty, A. Clark, and I. Pratt. Gprsweb: Optimizing the web for gprs links. In *ACM/USENIX First International Conference on Mobile Systems, Applications and Services*, 2003.

[9] R. Chakravorty, S. Katti, J. Crowcroft, and I. Pratt. Flow aggregation for enhanced tcp over wide-area wireless. In *IEEE INFOCOM*, 2003.

[10] M. Chan and R. Ramjee. TCP/IP Performance over 3G Wireless Links with Rate and Delay Variation. In *Proc. of ACM Mobicom*, Sept. 2002.

[11] P. Communication. With Inktomi and Cisco. 2002.

[12] A. Fox, S. Gribble, Y. Chawathe, and E. Brewer. Adapting to network and client variation using active proxies: Lessons and perspectives. *IEEE Personal Communications*, 5(4), August 1998.

[13] G. Holland and N. Vaidya. Analysis of TCP Performance over Mobile Ad Hoc Networks. In *Proc. of ACM Mobicom*, Aug. 1999.

[14] B. Inc. The Macara Optimization Service Node. http://www.bytemobile.com/html/products.html.

[15] F. S. Inc. The Venturi Server. http://www.fourelle.com/pdfs/Venturi_V2.1_Brochure.pdf.

[16] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. Dns performance and the effectiveness of caching. *IEEE/ACM Transactions on Networking*, 10(5), October 2002.

[17] G. Li, M. Lu, M. Meyers, and P. Feder. Wireless Data Service Emulator (WiDSE): A Real-time Look and Feel Emulator for Wireless Packet Data Systems. Technical memorandum, Lucent Technologies, Mar. 2002.

[18] M. Liljeberg, H. Helin, Kojo, and K. Raatikainen. Enhanced services for world-wide web in mobile wan environment. In *ImageCom*, 1996.

[19] T. R. N. Networks. Layer N Switching. http://www.nortelnetworks.com /solutions/financial /collateral/may99 _layer4_v3.pdf.

[20] S. Paul, E. Ayanoglu, T. LaPorta, K. Chen, K. Sabnani, and R. Gitlin. An Asymmetric Link-Layer Protocol for Digital Cellular Communications. In *Proc. of IEEE Infocom*, Apr. 1995.

[21] M. Rabinovich, Z. Xiao, F. Douglis, and C. Kalmanek. Moving edge side includes to the real edge – the clients. In *4th USENIX Symposium on Internet Technologies and Systems*, 2003.

[22] A. Shaikh, R. Tewari, and M. Agrawal. On the Effectiveness of DNS-based Server Selection. In *Proc. of the IEEE Infocom*, Apr. 2001.

[23] P. Sinha, N. Venkitaraman, R. Shivakumar, and V. Bharghavan. WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks. *Wireless Networks*, 8(2-3), 2002.

[24] B. C. Systems. Content Filtering with Blue Coat Systems. http://www.bluecoat.com/solutions/content_filtering.html.

[25] A. Technologies. EdgeSuite. http://www.akamai.com/en/html/services/edgesuite.html.