Characterizing Resource Usage for Mobile Web Browsing

Feng Qian, Subhabrata Sen, and Oliver Spatscheck
AT&T Labs – Research, Bedminster, New Jersey, USA
{fengqian,sen,spatsch}@research.att.com

ABSTRACT

Multiple entities in the smartphone ecosystem employ various methods to provide better web browsing experience. In this paper, we take a first comprehensive examination of the resource usage of mobile web browsing by focusing on two important types of resources: bandwidth and energy. Using a novel traffic collection and analysis tool, we examine a wide spectrum of important factors including protocol overhead, TCP connection management, web page content, traffic timing dynamics, caching efficiency, and compression usage, for the most popular 500 websites. Our findings suggest that that all above factors at different layers can affect resource utilization for web browsing, as they often poorly interact with the underlying cellular networks. Based on our findings, we developed novel recommendations and detailed best practice suggestions for mobile web content, browser, network protocol, and smartphone OS design, to make mobile web browsing more resource efficient.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols – Applications; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design – Wireless Communication; C.4 [**Performance of Systems**]: Measurement Techniques

Keywords

Mobile Web; HTTP; SSL; SPDY; Energy Consumption; Cellular Networks; Smartphones

1. INTRODUCTION

Web browsing is one of the key applications on mobile devices. Cellular traffic volume of browser-based web browsing surpasses that of any other application except for multimedia streaming [26]. Multiple entities in the smartphone ecosystem employ various methods to provide better web browsing experience. Content providers publish mobile versions of their websites. Based on our examination of the Alexa top 500 websites in February 2013, 327 out of the 500 websites have mobile versions that are specifically designed for smartphone/tablet users (§3). Mobile carriers deploy middleboxes in their networks to specifically optimize web traffic. Mobile browsers also introduce a wide range of techniques to make web browsing on handheld devices easier, safer, and faster (*e.g.,* through the SPDY protocol [9] or a compression proxy).

In this paper, we take a first comprehensive examination of the *resource usage* of mobile web browsing by focusing on two important types of resources: bandwidth and energy. Our study is driven by two facts. *(i)* Bandwidth and energy are the key resource bottlenecks for mobile users. *(ii)* Web developers are often unaware of cellular-specific characteristics. Although mobile websites have tailored their appearances for mobile device screens, it is unclear how well their design fits cellular networks operating under severe resource constraints compared to Wi-Fi and wired networks.

Bandwidth is a vital resource for cellular customers who pay for their traffic. A recent white paper from Mobidia [15] suggests that in UK most smartphone customers (*e.g.,* 70% for Vodafone) use a monthly data plan of no more than 500 MB (less than 16.7 MB per day on average). For all carriers, data roaming is even much more expensive. Therefore under the constraints of delivering necessary contents and providing good user experiences, mobile web browsing should consume as less bandwidth as possible.

Battery Life is another critical bottleneck. We focus on the power consumed by the handset radio interface. When the radio is on, it contributes to 1/3 to 1/2 of the total handset power usage [28]. It is well known that in cellular networks, the radio energy consumption highly depends on network traffic patterns, and it is always more energy-efficient to batch all transfers in one traffic burst than to transfer them intermittently in separate bursts. This is because in 3G/4G networks, there exists a timeout, called *tail time*, for turning off the radio interface after a data transfer. The tail time (T_{tail}) ranges between several seconds to more than 10 seconds depending on the carrier settings [28, 22]. Therefore, if two data bursts are transferred separately, then within the timing gap between the two bursts, the radio interface is still on (and consuming power) for up to T_{tail} seconds, leading to waste of the radio energy.

To understand the resource consumption of mobile web browsing, we collected network traces and browser events for landing pages of the top 500 websites on smartphone, and conducted in-depth analysis on their bandwidth consumption and their energy efficiency. A *landing page* is loaded after a user enters the website URL and hits the "Go" button. The importance of landing pages is well recognized: they are the first and the most frequently viewed pages of websites. A high-quality landing page is one of the key factors of online marketing [11]. Therefore, developers usually put a lot of effort on optimizing this important resource. However, our findings suggest that many of them are far from

MobiSys'14, June 16–19, 2014, Bretton Woods, New Hampshire, USA.

http://dx.doi.org/10.1145/2594368.2594372.

resource-efficient. Further, web pages are also directly fetched by many smartphone apps that contain programmable browsers (*e.g.,* **WebView** in Android), so making web pages more resource-efficient can benefit many mobile apps as well. We have made the following methodological contributions and new observations.

1. The UbiDump Tool (§2). HTTPS and SPDY are becoming increasingly popular (*e.g.,* Google Chrome for Android already supports tunneling all browsing traffic to SPDY proxies [2]). We thus built a measurement tool called UbiDump, which runs on a mobile device and precisely reconstructs all web transfers carried by HTTP, HTTPS, and SPDY [9] from captured **tcpdump** traces. It supports on-device SSL decryption *without* requiring servers' private keys or a man-in-the-middle (MITM) proxy. The decryption is realized by instrumenting the SSL Library and intercepting session keys during SSL handshakes. UbiDump provides a complete view of today's web traffic, and enables cross-layer analysis that cannot be performed by a browser-based data collection approach.

2. Protocol Bandwidth Overhead (§4.2). We performed a first analysis of the bandwidth impact of HTTPS and SPDY protocols based on real smartphone web browsing traffic, and discovered surprisingly high protocol bandwidth overheads. The overall protocol efficiency rate, defined as the size of HTTPS/SPDY payload divided by the total size of data transferred in the network (except for TCP retransmissions), is only 66% to 73% for cold-cache (*i.e.,* empty cache) load. For warm-cache (*i.e.,* non-empty cache) load, the efficiency can be as low as 11%. Such low efficiency is primarily caused by a tendency to open new TCP connections rather than to reuse existing ones. Each new TCP connection incurs high bandwidth overhead due to SSL handshake.

3. New Insights on Mobile Web Contents (§4.4). We examined causes of large websites, objects, and images. We also conduct in-depth analysis on hosts that are accessed by multiple websites, and found that they provide third-party services such as user tracking, and their functionalities are usually not related to the main content of the website (exceptions exist though). Removing them saves at least 39% (20%) of bandwidth for 20% of mobile websites in warm-cache (cold-cache) load.

4. Energy and Root-cause Analysis for Network Idleness (§4.5, §4.6). We perform novel analysis to identify network idle periods during a page load. The idle periods, during which the radio may still be on and be consuming power, can be caused by a few reasons such as delayed transfers triggered by Javascript. Surprisingly, reducing such idle time gaps brings a median radio energy saving of up to 59% across all mobile websites.

5. New Findings of Caching and Compression (§4.7, §4.8). We found that caching is poorly utilized for many mobile sites. 26% of the top mobile sites (*e.g.,* **m.bing.com**) mark their main HTML files as non-storable, leading to potential bandwidth waste. About 23% of objects in mobile sites are cacheable but have freshness lifetime of less than 1 hour. Within them, more than 70% of the objects are images, fonts, CSS, and Javascript whose contents are not likely to change within such a short time period of 1 hour. We investigate compression opportunities for HTTP, SSL, and SPDY, and found SSL stream-level compression outperforms the traditional HTTP object-level compression.

Overall, we found that factors at different layers can affect resource utilization in mobile web browsing, including web contents, HTTP, SSL, and even TCP. In particular, although many mobile websites have tailored their appearances for smartphone screens, many of the adjustments are very superficial, and the resulting sites are often poorly interacting with the cellular network. We observe very little difference between mobile and non-mobile websites in many aspects such as caching aggressiveness and object sizes.

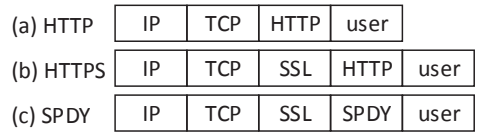

Figure 1: Web protocols supported by UbiDump.

And for some metrics such as the number of redirections, mobile websites are even worse. Based on our findings, we developed novel recommendations and detailed best practice suggestions for mobile web content, browser, protocol, and smartphone OS design, to make mobile website access more resource efficient.

The remainder of this paper is organized as follows. §2 and §3 describe the UbiDump tool and our methodology for collecting top 500 landing pages, respectively. §4 details the measurement findings, based on which we provide our recommendations in §5. We present related work in §6 before concluding the paper in §7.

2. THE UbiDump TOOL

The first step towards analyzing mobile web browsing is data collection. This can be obtained directly from a browser (*e.g.,* a browser extension), from a server, from a middlebox (*e.g.,* a proxy server), or from network traces. We adopt the last method by building a measurement tool called UbiDump. It runs on a mobile device, and precisely reconstructs all web transfers from **tcpdump** traces captured on the handset, thus requiring no change to the existing browser, server, or middlebox. Further, since **tcpdump** runs below the application layer, UbiDump allows capturing traffic across all applications (not only browsers). Another advantage of UbiDump is it also captures protocol headers at lower layers such as TCP, IP, and SSL. This enables cross-layer analysis (*e.g.,* §4.2) that cannot be performed by using the browser-based approach.

Handling SSL. SSL is becoming increasingly popular. Previous work handle the decryption of SSL traffic in **tcpdump** traces in three different ways. *(i)* Simply ignore the content of SSL traffic [26, 23, 37]. *(ii)* Some tools such as **Wireshark** [13] and **ssldump** [12] require servers' private keys for decryption. However obtaining private keys of commercial servers is usually impossible. *(iii)* Redirect traffic to a man-in-the-middle (MITM) SSL proxy such as **mitmproxy** [8] and Stanford MITM proxy [10]. HTTPS requests are then terminated by the proxy and resent to the remote web server in a new TCP connection. The certificate of the proxy needs to be installed on the client. To decrypt the SSL traffic, one can provide **Wireshark** or **ssldump** with the private key of the proxy. This MITM approach however has limitations: it requires additional infrastructures and may change traffic pattern and content, as will be shown in §2.2.

A key contribution of UbiDump is it supports *on-device SSL decryption* without requiring servers' private keys or a proxy. Throughout this paper, "SSL" will be used to refer to SSL v2, SSL v3, and TLS v1, all supported by UbiDump. Our key observation is that, unlike a private key that is only known to the server, a *session key* is derived independently by both the client and the server when an SSL session is established. Symmetric encryption is used in subsequent data transfers where data is encrypted and decrypted using this session key. Therefore, UbiDump realizes on-device decryption using an instrumented SSL library, which dumps SSL session keys during SSL handshakes. The session keys are then combined with **tcpdump** traces to derive the decrypted data.

Protocol Analyzers. Today's web traffic is carried by three application-layer protocols: HTTP, HTTPS, and SPDY, with the

latter two becoming increasingly popular. UbiDump thus contains analyzers for all three protocols shown in Figure 1, plus DNS. SPDY [9] is developed primarily at Google for transporting web content. It provides an HTTP interface to applications but differs from HTTP mainly in three ways. *(i)* In SPDY, concurrent transfers of multiple web objects belonging to the same domain can be multiplexed in a single TCP connection. *(ii)* SPDY can prioritize some object loads over others. *(iii)* Request and response headers embedded in SPDY headers are always compressed using a customized dictionary.

2.1 System Implementation

UbiDump is a handy traffic capture and analysis tool supporting on-device SSL decryption and web content analysis. We describe the implementation of UbiDump on Android[1]. The system consists of the following three components.

1. A `tcpdump` program running on the handset. It records all network traffic in **`pcap`** format with negligible runtime overhead incurred.

2. An instrumented Android SSL library (**`libssl.so`**). During an SSL handshake, it dumps a premaster secret p, and a 48-byte master secret m to the Android debugging log. They will be used for SSL decryption to be explained soon.

3. Protocol analyzers. They parse the **`tcpdump`** trace from IP to application layers and extract all web transfers over HTTP, HTTPS, and SPDY. The analyses can be performed in an online manner although they are currently offline. We leveraged the existing PCAP/HTTP analyzer implementation in the ARO tool [28], and wrote 4K lines of C++ code for SSL, SPDY, and DNS analyzers. Additionally, we leveraged several cryptographic functions in **`openssl`** and Linux WPA Supplicant [7] projects.

SSL Decryption is performed as follows. During an SSL handshake, the handset generates a premaster secret p, encrypts p using server's public key into p_e, and then sends p_e to server, which decrypts p_e using its private key. Both sides then independently compute the master secret m:

$$m = \Theta(p, r_c, r_s) \tag{1}$$

where Θ is a publicly known function, r_c and r_s are random strings generated by client and server, respectively, at the beginning of the handshake. They are exchanged in plain text. Let there be multiple SSL sessions captured by **`tcpdump`**, which outputs a list of (r_c, r_s, D) triples where D is the traffic data to be decrypted in an SSL session. Meanwhile we obtain from the modified SSL library a list of (p, m) pairs. Equation 1 bridges the two lists so that we can associate each (p, m) pair with its corresponding D. The final session key for decrypting D is derived from m, r_c, and r_s using another publicly known function.

2.2 Discussions

Limitations. We describe cases where UbiDump cannot decrypt SSL traffic from the packet trace. First, if the capture starts in the middle of an SSL session, it is impossible for UbiDump to decrypt that session since the key is not captured. One solution is to instrument all places in the SSL library where decrypted bytes are delivered to upper layers.

Second, the current SSL protocol analyzer only implements the most popular RSA key exchange algorithm. We therefore restrict the set of supported key exchange algorithms in the handshake message sent from the handset by adding one line of code to **`libssl.so`**. We are working on making our analyzer support other key exchange algorithms such as Diffie-Hellman. UbiDump supports all symmetric encryption (AES, 3DES, RC4) and MAC algorithms (SHA, MD5) used by the original Android SSL library.

Third, the entire decryption mechanism can be bypassed if a smartphone app implements its own SSL logic, or if the SSL library is statically linked into the app binary. We examined several popular built-in and third-party apps on Android such as the browser, the Google Map, and banking apps. SSL traffic of the vast majority of those apps can be decrypted by UbiDump. For completeness, UbiDump can also read private keys from files and use them to decrypt SSL. Note none of the above limitations affects the SSL decryption in our measurement study.

Comparing with the MITM-Proxy-based Approach. We perform a case study to show the potential impact of an MITM proxy on the traffic. We collected two traces A and B for the same website (**`https://www.chase.com`**). The mobile handset (Samsung Galaxy S III running Android 4.0.4), the network (Wi-Fi), the collection time and location are all same for the two traces. Trace A is collected using the MITM-Proxy-based approach (we use **`mitmproxy`** [8] due to its popularity) and Trace B is collected by UbiDump without a proxy.

By comparing the two traces, we found the following differences. *(i)* In Trace A, the phone issues an **`HTTP CONNECT`** request to the proxy before establishing the SSL session while in Trace B, the SSL session is established without that additional round trip. *(ii)* Their TCP connections are closed in different ways. This affects handset energy consumption in cellular networks (§4.3). *(iii)* Some objects in Trace B are transferred in SPDY that is never used in Trace A. This is because the proxy does not support SPDY so the handset has to fall back to HTTPS. The above observations indicate that an MITM proxy may change the traffic pattern and content compared to the case where the phone directly connects to the original server.

3. COLLECTING TOP 500 LANDING PAGES

We describe how we use UbiDump to collect landing pages of the top 500 websites to be characterized in §4.

We obtained a list of top 500 U.S. websites from Alexa in February 2013. According to Alexa, the sites are ordered by their one-month traffic ranks, which are calculated "using a combination of average daily visitors and page views over the past month". Although the ranks are derived from the usage of all Internet users, we expect they are also popular among just mobile users.

Automated Testbed. We installed UbiDump on a Samsung Galaxy S III smartphone running Android 4.0.4. We wrote another Android program that loads the website URLs in a **`WebView`** in an automated manner. A **`WebView`** is the programmable version of the default Android browser.

Our testbed leverages UbiDump to collect two types of traces: a *cold-cache* load where all caches (DNS/SSL/web content cache caches) are cleared before loading, and a *warm-cache* load right after the cold-cache load without clearing any cache. For each website, we repeat the cold and warm load pair for three times in a row, obtaining six traces. We collected all websites using both a 3G HSPA+ network and a corporate Wi-Fi network, resulting in two datasets which we call DS1 and DS2, respectively. Both DS1 and DS2 were collected at late nights in spring 2013 at good signal strength levels, to minimize the impact of network congestion, as well as to provide a best case for radio energy consumption. All collected SSL sessions were successfully decrypted by UbiDump.

Completion of Page Load (so that we stop the data collection) is determined by a timeout of 30 seconds (*i.e.,* when no request has been made for 30 seconds). To validate that the load is indeed

[1]An Android device needs to be rooted to run UbiDump.

successful, we let the testbed take a picture of the entire rendered page and examine the screenshot manually to check failures such as any unrendered part due to network timeout. For each website, we collect all its six traces again if any of them is found to be problematic. We also impose a limit of 90-second data collection duration for pages that continuously load new data (*e.g.,* due to periodic user tracking).

Landing Pages. Each collected page is a *landing page* that is loaded if a user enters the URL (*e.g.,* `cnn.com`) and hits the "Go" button. The importance of landing pages is well recognized: they are the first and often the most frequently viewed pages of websites. A high-quality landing page is one of the key factors of online marketing as it can keep visitors at the website by capturing their short attention span of 2 to 3 seconds [11]. Therefore, developers usually put a lot of effort on optimizing this important resource. However, we found many of them are far from resource efficient.

Identifying Mobile and Non-mobile Sites. For each website, we manually label it as either mobile version or non-mobile (desktop) version. For most landing pages, this is straightforward to tell from the screenshot. Another obvious clue is that for many mobile websites, their redirected landing page URLs begin with `m.xyz.com` or `xyz.com/mobile`. For a small number of websites, we label them by comparing between the rendering result on the phone and that on a desktop. Among the top 500 websites, 327 (65.4%) are found to be mobile versions designed specifically for mobile handsets. 143 (28.6%) are non-mobile versions *i.e.,* mobile users are served with the same contents as desktop users are. The remaining 30 (6%) websites could not be loaded on either a mobile or a desktop browser, and are therefore excluded from our analysis. Many of such failed websites are "helper" domain names such as `googleusercontent.com` not providing a web interface.

3.1 Discussions

Mobile Apps vs. Websites. About 1/4 of our measured websites provide official apps in Android market (in Feb 2013). Although we do not have the statistics, we do believe many users prefer using the apps to access the content providers, in particular for those popular ones (*e.g.,* top 100). However, lessons and recommendations derived from this study are generally applicable to any mobile web page. Also, web pages are often directly fetched by many smartphone apps containing programmable browsers (*e.g.,* `WebView` in Android), so making web pages more resource efficient can benefit those apps as well. Conducting detailed comparisons between mobile apps and websites is our future work.

Web Page Complexity. It is possible that our analysis underestimates the complexity of today's mobile web pages since landing pages can be simple (*e.g.,* Facebook login page). However, it is difficult to identify "simple" web pages due to a lack of objective criteria. Further, we observe many cases where a visually simple landing page contains complex Javascript and CSS scripts, as well as involves large data transfers when being loaded. We thus include all loadable websites in subsequent analyses. Also, for many sites, their landing pages may not be the pages on which users spends most of their time. But we still believe they are worthy of being studied due to the importance of landing pages described early.

Popularity of HTML5. HTML5 provides new features, such as web sockets, enhanced multimedia, and offline caching, to enhance web browsing experience. We found HTML5 is still not widely used for mobile websites (as of Feb 2013). For example, only eight mobile sites leverage the offline caching feature to enhance the user experience when the device is offline. Studying HTML5 is our future work as it will eventually become popular on mobile devices.

Table 1: Usage of HTTPS and SPDY across landing pages of all websites (the DS1 Dataset).

Protocol	Mobile Sites		Non-mobile Sites	
	Cold	Warm	Cold	Warm
HTTPS	58.7%	48.6%	93.7%	68.5%
SPDY	22.3%	21.4%	42.7%	39.2%

The Connection Split Proxy. Mobile carriers deploy in-network middleboxes or proxies for various purposes. In particular, the carrier from which we collected the DS1 dataset deploys a proxy in the cellular core network for HTTP traffic (server port 80/8080). It transparently splits an end-to-end TCP connection into two, one between a handset and the proxy, and the other between the proxy and the remote server. Although splitting connection can improve TCP performance in certain circumstances [19], it has negligible impact on our measurement results in §4 except for the TCP connection management part, on which the impact is also small. We revisit this in §4.3 and §4.6.

4. MEASUREMENT RESULTS

We next examine various aspects that can affect resource consumption of mobile web browsing: protocol bandwidth overhead (§4.2), TCP connection management (§4.3), page content (§4.4), network idle time during page loading period (§4.5, §4.6), caching semantics (§4.7), and compression usage (§4.8). As will be shown, improving the resource efficiency of mobile web browsing requires *joint efforts* of multiple entities in the smartphone ecosystem.

From this point on, a *website* refers to its landing page unless otherwise noted. An *object* refers to a URL fetched by the handset. The *transfer* of an object includes both sending the request and receiving the response. Each figure or table can present a distribution across websites, TCP connections, or objects, which are clearly stated[2]. Also, the analysis results of DS1 and DS2 are qualitatively similar unless otherwise noted. So we usually only present the results of DS1. We do not find anything distinguishing Wi-Fi and 3G in request headers.

4.1 Protocol Prevalence

Table 1 shows the prevalence of HTTPS and SPDY across websites, for both cold-cache load ("Cold") and warm-cache load ("Warm"). We say a website (*i.e.,* its landing page) uses SSL (SPDY) if we observe at least one TCP flow that carries an SSL (SPDY) session, which is therefore counted by Table 1. Table 1 indicates many top websites use SSL. In DS1, we observed SSL sessions are established to 300 distinct content providers identified by their host names. We also found from the datasets that as of today, SPDY is only employed by a few content providers including Google, YouTube, Facebook, Twitter, and AdMob, *etc.* However, other websites often embed their services (*e.g.,* Facebook feeds, see §4.4). This explains why Table 1 still indicates SPDY is reasonably prevalent among the top websites.

Regarding to traffic volume and object count, HTTP still dominates the web usage in both metrics. For DS1, HTTPS and SPDY together account for around 10% and 6% of the total bytes for mobile sites and non-mobile sites, respectively (similar case for object count). However, SSL and SPDY are becoming increasingly popular. In 2010, Gmail and Twitter made HTTPS the default

[2]Each dataset contains 470*3 cold-load traces and 470*3 warm-load traces. All are used in our analysis unless otherwise noted. Therefore, for example, a CDF plot of a certain distribution across all cold-load websites contains 470*3 data points.

Table 2: Protocol byte breakdown for DS1.

Protocol	Category	*tcpip_ctl*	*udp_usr*	*ssl_ctl*	*ssl_usr_enc* – *ssl_usr_dec*	*http_ctl* or *spdy_ctl*	*http_usr* or *spdy_usr*
HTTP	Cold, Mobile sites	9.0%	0.2%	0	0	6.9%	83.9%
	Cold, Non-mobile	9.2%	0.2%	0	0	8.6%	81.9%
	Warm, Mobile sites	13.2%	0.3%	0	0	21.5%	65.0%
	Warm, Non-mobile	13.6%	0.5%	0	0	25.3%	60.6%
HTTPS	Cold, Mobile sites	8.6%	0.1%	11.6%	0.0%	6.2%	73.4%
	Cold, Non-mobile	9.7%	0.3%	17.0%	0.3%	7.0%	65.8%
	Warm, Mobile sites	12.3%	0.1%	33.9%	0.3%	17.1%	36.2%
	Warm, Non-mobile	13.6%	0.3%	44.5%	-0.4%	16.3%	25.6%
SPDY	Cold, Mobile sites	9.7%	0.4%	10.5%	1.6%	5.2%	72.6%
	Cold, Non-mobile	10.8%	0.4%	12.1%	1.9%	5.2%	69.5%
	Warm, Mobile sites	15.7%	0.3%	24.9%	2.0%	12.0%	45.1%
	Warm, Non-mobile	22.9%	0.5%	44.0%	3.1%	18.3%	11.2%

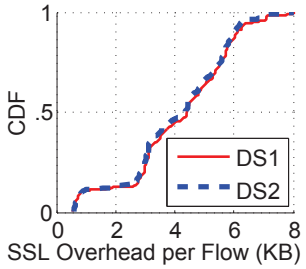

Figure 2: SSL handshake overhead across TCP flows.

for all users [4]. Facebook started to support HTTPS in 2011, and moved all users to HTTPS in 2012 [4]. Google Chrome for Android already supports tunneling all browsing traffic to SPDY proxies [2]. Understanding the resource impact of SPDY and HTTPS is therefore critical.

4.2 Bandwidth Impact of Protocols

The additional protocol layers introduced by HTTPS and SPDY may incur additional overheads. Previous studies mostly focus on their security implications and CPU overhead. Here our emphasis is to understand the *bandwidth* impact of diverse protocols for mobile web browsing.

Our analysis goes from lower layer to higher layer. The raw bytes captured by UbiDump on the wire consist of four components: TCP/UDP/IP headers ($tcpip\_ctl$), UDP payload carrying DNS traffic (udp_usr), TCP payload (tcp_usr), and retransmitted TCP packets (tcp_rt).

$$raw = tcpip\_ctl + udp\_usr + tcp\_usr + tcp\_rt \quad (2)$$

For HTTP, the TCP payload contains HTTP headers ($http\_ctl$) and HTTP payloads ($http\_usr$) in plain text.

$$tcp\_usr = http\_ctl + http\_usr \quad (3)$$

HTTPS and SPDY use SSL. Therefore the TCP payload consists of SSL header and control data (ssl_ctl), as well as the encrypted SSL payload (ssl_usr_enc).

$$tcp\_usr = ssl\_ctl + ssl\_usr\_enc \quad (4)$$

Let the decrypted SSL payload be ssl_usr_dec. For HTTPS, it is comprised of HTTP headers and payloads. SPDY is similar to HTTPS but we use different symbols for SPDY headers/control data ($spdy\_ctl$) and SPDY payloads ($spdy\_usr$). Note $http\_usr$ and $spdy\_usr$ might be encoded by server (*e.g.,* compressed or transferred in HTTP chunked mode).

$$ssl\_usr\_dec = http\_ctl + http\_usr \quad (5)$$

$$ssl\_usr\_dec = spdy\_ctl + spdy\_usr \quad (6)$$

Since UbiDump captures all traffic at the network interface, it can precisely calculate all values in Equations 2 to 6. Table 2 shows the breakdown for DS1. We classify all TCP flows (a TCP flow may include DNS lookup(s) before the flow starts) into 12 categories based on the combinations of (app protocol, cold/warm loads, mobile/non-mobile sites). Also Table 2 excludes all TCP retransmissions which are network-dependent. In other words, we compute the fractions of aforementioned traffic components within $(raw - tcp\_rt)$. Numbers in each row add up to 100%.

We describe the findings regarding to Table 2 as follows. *(i)* Byte contributions of TCP/IP headers and HTTP/SPDY headers are non-trivial, especially for warm-cache loads whose website payload sizes (Figure 4), object payload sizes (Figure 5), and packet payload sizes are all statistically smaller than those of cold-cache loads. *(ii)* The $ssl\_usr\_enc - ssl\_usr\_dec$ column quantifies the overhead due to symmetric encryption. Usually it is positive due to the MAC (Message Authentication Code) or paddings (for blocked ciphers) introduced in the encryption process. However, SSL supports stream-level compression that compresses the entire SSL stream before encrypting it. If that option is used, ssl_usr_enc refers to the compressed and encrypted bytes appearing on the wire, and ssl_usr_dec denotes the decrypted and decompressed bytes delivered to the upper layer. In that case, ssl_usr_enc can be smaller than ssl_usr_dec. We found that only a few HTTPS flows use this compression option but none of the SPDY flows uses it (§4.8). This explains why $ssl\_usr\_enc - ssl\_usr\_dec$ for HTTPS is much smaller than that of SPDY. *(iii)* DNS (udp_usr) accounts for negligible traffic.

Why is SSL Overhead so High? Table 2 indicates that SSL header and control traffic (ssl_ctl) is responsible for the highest protocol overhead among all eight non-HTTP categories *e.g.,* as high as 45% for warm-cache load using HTTPS/SPDY. This is explained as follows. *(i)* We found that most such overhead comes from the SSL handshake phase. Figure 2 plots the distribution of the number of bytes incurred by the handshakes across all SSL sessions, for both datasets. The overhead ranging from 0.5 KB to 8 KB (mean: 4.2 KB, median: 4.4 KB) depends on several factors including the certificate size, whether a full or

Table 3: What-if analysis of changing protocols. The numbers shown are the overall protocol efficiency rates (DS1).

What-if Scenario	Mobile Sites		Non-mobile Sites	
	Cold	Warm	Cold	Warm
Original	83.0%	62.3%	81.0%	58.1%
HTTP→ HTTPS	71.1%	42.7%	68.2%	38.6%
Plus GTP	78.2%	57.1%	76.2%	53.1%
HTTP→HTTPS + GTP	66.7%	39.1%	63.9%	35.4%

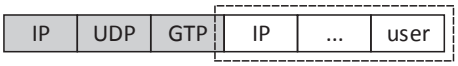

Figure 3: A common configuration of GTP. The user packet is surrounded by dashed line.

an abbreviated handshake is performed, *etc.*[3] (In contrast, the overhead of a TCP handshake is only around 100 bytes). In the SSL data transfer phase, each encrypted data block only has a 5-byte header, and the overhead introduced by symmetric encryption ($ssl\_usr\_enc - ssl\_usr\_dec$) is also small as shown in Table 2 so the SSL overhead in data transfer phase is negligible. *(ii)* TCP connections are not effectively reused. We will detail the connection reuse issue in §4.3. *(iii)* As will be shown in Figure 5 (§4.4), most objects are small in payload size. By putting together the above three observations, we found the high overhead of SSL is caused by inefficient reuse of TCP connections, each incurring high SSL handshake overhead compared to the small payload it carries. Warm-cache loads incur higher SSL overhead than cold-cache loads do because for warm loads, their object payload sizes are smaller (Figure 5) and their connections are more poorly reused.

The Overall Protocol Efficiency Rate is defined to be the fraction of $http\_usr$ or $spdy\_usr$ shown in the rightmost column in Table 2. It quantifies the fraction of data (*i.e.,* the HTTP/SPDY payload) ultimately consumed by the user. HTTP significantly outperforms HTTPS and SPDY, and cold-cache load is more efficient than warm-cache load. Warm-cache load for HTTPS and SPDY always yields efficiency rates less than 50%, and sometime as low as 11%. The reasons have been detailed above.

What if All HTTP is Replaced by HTTPS? We consider a "what-if" scenario where all HTTP flows in the dataset switch to HTTPS. This is possible in the near future given the rapid deployment of HTTPS and SPDY. We construct the what-if scenario by adding to TCP flows carrying HTTP objects dummy SSL handshakes whose sizes conform to the distribution shown in Figure 2. The overhead in the SSL data transfer phase is ignored. Also existing HTTPS and SPDY flows are not changed in the what-if scenario. The "Original" row in Table 3 lists the overall protocol efficiency rates of the original DS1 trace, considering all three protocols. The "HTTP→HTTPS" row corresponds to the what-if scenario. The results indicate that switching to HTTPS will reduce the *overall* protocol efficiency rates by at least 12% to 20%.

What if Transferred in Cellular Core Network? When passing through the 3G/LTE cellular core network, user packets will be tunneled by the GPRS Tunneling Protocol (GTP) [5]. As shown in Figure 3, when tunneled, a user packet (surrounded by dashed line) is encapsulated by an IP header (20 bytes), a UDP header (8 bytes), and a GTP header (at least 8 bytes). In LTE, when a base station receives a user packet over air interface, it will perform the encapsulation and forward the packet towards the serving gateway (SGW). The GTP header contains the tunnel information, and the outer IP header contains the base station IP as the source address and the SGW IP as the destination address. UMTS uses a conceptually similar scheme for tunneling.

The "Plus GTP" row in Table 3 indicates that under the aforementioned configuration, the overall protocol efficiency rates will decrease by at least 5% due to the additional IP/UDP/GTP headers. If the above two what-if scenarios are combined, the reduction will be up to 23% as shown in the last row of Table 3.

[3]The implementation limitation where only RSA key exchange is used by the modified SSL library of UbiDump has negligible impact on the handshake overhead, based on our local experiments.

Table 4: TCP reuse status across objects.

Data Set	Protocol*	Reused	First	Connection Not Reused		
				Closed	Busy	Other
DS1	HTTP(S)	35.5%	28.6%	4.5%	13.5%	18.0%
	SPDY	47.4%	52.6%	0	0	0
DS2	HTTP(S)	37.2%	28.4%	10.6%	7.3%	16.5%
	SPDY	47.4%	52.6%	0	0	0

* HTTP(S) means HTTP or HTTPS.

4.3 TCP Connection Management

We study two aspects of TCP connection management policy: connection reuse and connection closure.

Reusing TCP Connection for multiple objects helps reduce the resource utilization in two ways. *(i)* It reduces overheads such as TCP handshake, slow start, and SSL handshake, leading to fewer round trips. In cellular networks with high latency, this means lower radio energy consumption since usually the radio is not turned off during the entire page loading period due to the long tail time explained in §1. *(ii)* Reusing connections also helps lower the bandwidth utilization caused by SSL handshake (§4.2).

To understand whether TCP connections are properly reused, we start by measuring the number of distinct server IPs the handset contacts during loading a website. In DS1, the median values for (mobile sites, cold-cache load), (mobile, warm), (non-mobile, cold), (non-mobile, warm) are 8, 5, 10, and 7, respectively. However, complex sites may involve up to 100 distinct IPs. The number of distinct IP addresses is clearly the *lower bound* of the number of required TCP connections, which we found to be much higher: the median values of the total number of actual established TCP connections of a website are 22, 7, 37, and 12, respectively, for the aforementioned four schemes.

Why are TCP Connections not Reused? To answer this, we classify each object into the following five categories shown in Table 4[4]. *(i)* The object is transferred by reusing an existing TCP connection ("Reused"). *(ii)* Reusing a connection is inherently impossible because it is the first time that the client connects to that server ("First"). *(iii)* No connection is reused because all existing connections are closed or marked as `Connection:close` ("Closed"). *(iv)* All existing connections are busy transferring user data, and (probably because of this) the client opens a new connection ("Busy"). *(v)* There is at least one reusable idle connection, but the client still opens a new connection ("Other").

Table 4 indicates striking differences between HTTP(S) and SPDY in the "Connection Not Reused" categories. *(i)* In HTTP(S), a connection may not be reusable if it closes too soon or if

[4]Usually a browser does not reuse a single connection for multiple host names, even if they resolve to the same IP address. Our analysis follows this practice although the results are qualitatively similar between IP-based and host-based reuse.

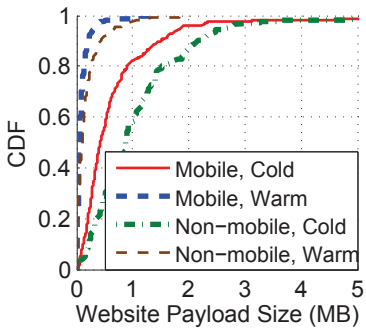

Figure 4: Payload sizes across websites (DS1).

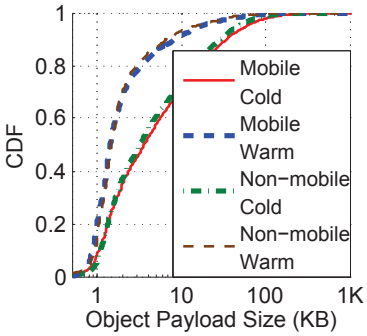

Figure 5: Payload sizes across objects (DS1).

`Connection:close` is used. SPDY addresses this problem by using a much longer timeout (describe soon) and by forbidding to use the `Connection` header field (*i.e.,* SPDY always assumes a persistent connection). *(ii)* In HTTP(S), a busy connection may block a new request to the same server but this does not happen in SPDY, which allows multiplexing transfers of multiple objects of the same domain in a single connection. SPDY's approach can potentially create head-of-line blocking at the transport layer, whose negative effects can somewhat be mitigated by prioritizing object loads [9]. *(iii)* The "Other" category indicates that for HTTP(S), often the client should have reused the connection but it did not, probably due to an implementation issue. Overall, we found SPDY outperforms HTTP(S) in reusing TCP connections, leading to the benefits described at the beginning of this subsection.

Connection Closure: HTTP vs SPDY. When an HTTP(S) data transfer finishes, a client or a server usually waits for a while before sending a FIN, to increase the chance of the TCP connection being reused [27]. Our measurement shows that for HTTP(S), most of their TCP connections are closed by the handset using TCP FIN, and the remaining are usually terminated by FIN initiated by the server, or by the connection split proxy in the studied cellular network for server port 80/8080 (§3.1). Negligible fraction of connections are terminated by TCP RST. The closure delay (*i.e.,* the delay between the last data packet and the FIN/RST) usually ranges between 0 and 10 seconds.

On the other hand, most TCP connections carrying SPDY traffic are not terminated when the data collection terminates. They are kept open for a long time *i.e.,* longer than the 30-second timeout used to determine the completion of a page load (§3). This is because SPDY recommends "clients do not close open connections until the user navigates away from all web pages referencing a connection, or until the server closes the connection". Servers are also encouraged to "leave connections open for as long as possible, but can terminate idle connections if necessary" [9]. This creates more opportunities for reusing TCP connections (Table 4). Note HTTPS/SPDY traffic does not traverse the connection split proxy.

Silent Connection Closure. As mentioned before, it is useful to reuse TCP connections to amortize overheads such as TCP and SSL handshakes. In cellular networks, delayed FIN/RST packets can cause significant radio energy drainage, due to their incurred tail times. To address such a dilemma between connection reuse and delayed connection closure overhead, recent work [27] proposes a TCP extension called Silent TCP connection Closure (STC). The key idea is that both sides close the connection silently without performing FIN handshake based on a negotiated keep-alive timer. It allows both sides to delay connection closure using the negotiated timer (and thus reuse connections). In the meanwhile it eliminates the tail time incurred by the FIN/RST packets that usually appear as separate traffic bursts. STC is generally applicable to any mobile application. It is lightweight, incrementally deployable, and backward compatible. All its introduced changes can be confined within cellular networks, which STC is specifically designed for, by upgrading the mobile device and cellular proxy software. In §4.6, we show STC is particularly useful for web browsing.

Table 5: Mobile websites with the largest payload sizes (DS1).

Website (Cold)	Type	# Objs	Avg Size	Height*
deadspin.com	Blog	71	6.5 MB	25.3
pinterest.com	OSN	128	5.4 MB	47.2
disney.go.com	Entertain	99	5.3 MB	6.5
salon.com	News	279	5.2 MB	15.2
Website (Warm)	**Type**	**# Objs**	**Avg Size**	**Height***
disney.go.com	Entertain	26	1.4 MB	6.5
perezhilton.com	Blog	54	1.1 MB	5.9
cbs.com	TV & Video	61	1.0 MB	2.4
yahoo.com	News	62	0.7 MB	14.0

* The unit is the screen height in portrait orientation.

4.4 Web Page Content

We now shift our focus to the web page content. Figure 4 plots the distribution of the payload sizes of the top websites (*i.e.,* their landing pages). The "payload size" refers to the size of $http\_usr$ or $spdy\_usr$ in Table 2. The payload may be compressed if the server does so. As expected, non-mobile sites are usually larger than mobile sites, and cold-cache loads transfer more bytes than warm-cache loads do.

Large Websites. Figure 4 indicates that for cold-cache load, 20% of mobile sites and more than 40% of non-mobile sites have more than 1 MB of payload. Table 5 lists the largest mobile sites (in terms of the total payload size) for cold and warm load. We found that statistically, large mobile sites tend to be much "taller" in their appearances than small mobile sites do[5]. We measure the height of a website from the captured screenshot in which the width of the page already automatically aligns with the handset screen width in portrait orientation. The "Height" column in Table 5 thus shows the page height in multiples of the handset screen height. The heights are usually far greater than 1 for large mobile sites. In contrast, mobile sites with small sizes are much more likely to have heights of exact one unit. We discuss solutions for reducing sizes of "tall" websites in §5.

[5]A few sites may do "infinite scrolling": when the user scrolls to the bottom of a tall page, the browser will load more contents, which are not captured by our automated testbed.

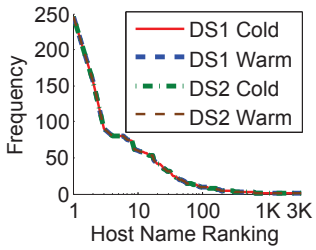

Figure 6: Occurrences of host names in distinct websites.

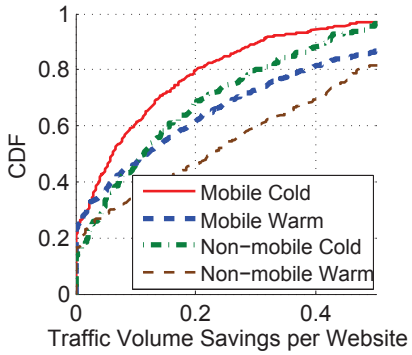

Figure 7: What-if analysis with shared hosts removed (DS1).

Table 6: Breakdown of large objects (mobile websites in DS1).

Type	% bytes (objs)	Large images (subset of large objects)	
Images	64.5% (60.2%)	Cause	% bytes (objs)
JS	22.8% (25.5%)	High Res. JPG/PNG	78.5% (78.1%)
Html	4.4% (5.4%)	Sprite JPG/PNG	6.7% (7.4%)
Fonts	4.0% (4.7%)	Animated JPG/PNG	4.1% (2.8%)
CSS	3.8% (3.9%)	Animated GIF	6.2% (3.7%)
Json	0.5% (0.3%)	Vector SVG images	4.6% (8.0%)

Table 7: Top hosts accessed by multiple websites (DS1, cold).

Host name	# Websites	Description
www.google-analytics.com	245	User tracking & stats
b.scorecardresearch.com	156	User tracking & stats
www.facebook.com	91	Facebook
ad.doubleclick.net	81	Advertisements
pixel.quantserve.com	81	User tracking & stats

Large Objects. We now look at individual web objects. As shown in Figure 5, the object payload size (including both request and response, but dominated by response payload) follows a heavy-tail distribution similar to that in Figure 4, except that the difference between mobile and non-mobile object sizes is much smaller. We next study the largest 2,632 objects of mobile websites in DS1 since they are the most bandwidth-consuming. They account of 2.4% of all mobile site objects while their payload size contribution is 50.0%. Their content type breakdown is shown on the left-hand side of Table 6. Images dominate the large objects in terms of both the bytes and the object count.

The right part of Table 6 investigates why the images belonging to the largest 2,632 objects have large sizes based on our visual inspection of more than 1,500 images. 78% of them are single images with (often unnecessarily) high resolutions, while about 10% of the large images consist of multiple sub-images whose resolutions are not high. Those sub-images can be animation frames. Unlike GIF, neither JPEG nor PNG has built-in support for animation so the animation will be "played" by quickly rolling over each sub-image using Javascript. Another usage of such an image slicing technique is called "Sprite Image" *i.e.,* to transfer multiple small images in one large image to save round-trip delays. A potential issue with this approach is the reduced caching efficiency, as the entire sprite image needs to be transferred if any sub-image changes. So sprite image better works with small sub-images.

Hosts Shared by Multiple Websites Providing Third-Party Services. We obtained from both datasets about 3,200 (2,200) unique host names for cold (warm) cache loads. Figure 6 ranks the host names by the number of websites they appear in (*i.e.,* the "frequency" on Y Axis). We found that although most hosts are only associated with one single website, there are a small amount of hosts that are accessed by a large number of websites. The top host name appears in 245 distinct websites. Table 7 describes the top shared hosts for DS1 (cold load for both mobile and non-mobile websites). We also manually verified host names appearing in at least 10 distinct websites in DS1 for both cold and warm load. All those hosts seem to provide third-party "add-on" features such as user tracking, usage measurement, advertisements, and social network feeds for the websites they are embedded in.

What if Shared Hosts are Eliminated? We consider a what-if scenario where the above third-party hosts are eliminated. Specifically, we remove traffic from/to hosts appearing in at least 10 distinct websites since they are verified to be not related to the main content of the website[6]. Doing so reduces both the bandwidth consumption and the radio energy utilization for the mobile handset. We consider the bandwidth saving here and quantify the energy benefits in §4.6. Figure 7 plots the CDF of traffic volume (HTTP/HTTPS/SPDY header and payload) savings across websites in the what-if scenario. For 20% of websites ($y = 0.8$), the bytes reduction achieves at least 20%, 39%, 30%, and 48% for (mobile sites, cold-cache load), (mobile, warm), (non-mobile, cold), and (non-mobile, warm), respectively. The 95-percentiles are even as high as 43%, 75%, 49%, and 73% for the four scenarios, respectively. Note these are the lower bounds of bandwidth savings since hosts shared by less than 10 distinct websites can also provide third-party services. The results indicate that for many websites, compared to their main contents, the embedded third-party services impose high bandwidth overhead, and optimizing them can lead to non-trivial or even significant bandwidth savings.

Redirections. Table 8 lists the distribution of the number of redirections (HTTP 301 or 302) before the main html file is transferred, for cold-cache load of each website[7]. To our surprise, 57.8% of the mobile sites take at least two redirections while 90% non-mobile sites require no more than one redirection. We found that some popular mobile sites have as many as 5 redirections before the phone can fetch the main html file, such as `msn.com`, `wsj.com`, and `bankofamerica.com`. Excessive redirections hurt in particular user experience in cellular networks with high latency by bringing in additional round trips. For example, loading the mobile version of `live.com`, the Microsoft online service gateway, on cellular network takes five redirections, with four having DNS lookup and three being transferred in HTTPS. All redirections take about 6.7

[6]We do handle a few exceptions in our analysis *e.g.,* we do not remove the `facebook.com` host from the Facebook website itself.
[7]For most sites, loading `www.xyz.com` instead of `xyz.com` can reduce the number of redirections by one.

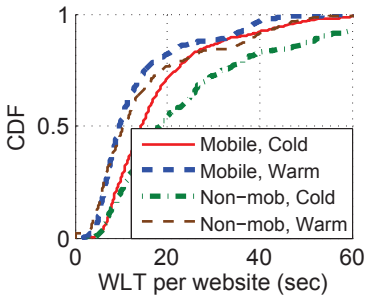

Figure 8: Distributions of WLT across websites (DS1).

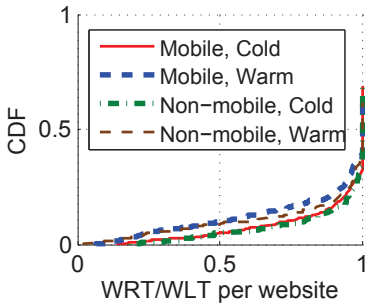

Figure 9: Distributions of WRT/WLT across websites (DS1).

Table 8: Number of redirections (DS1, **cold load**).

# Redirections	Mobile	Non-mobile
0	7.3%	24.7%
1	34.9%	65.3%
2	42.5%	6.3%
3	9.9%	2.3%
4	3.6%	0.7%
5	1.8%	0.7%

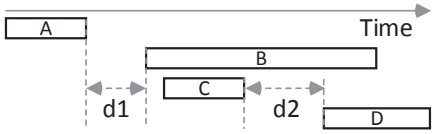

Figure 10: An example illustrating IIT. Each box corresponds to a web object transfer including both request and response.

seconds (median) in total, leading to significant user experience degradation and radio energy waste.

4.5 Timing Dynamics

In this subsection, we begin by measuring website load time and rendering time. Although performance is not our focus in this study, it is correlated with the radio energy utilization in cellular networks since usually the radio is on during the entire page loading period due to the long tail times. They also lead us to examine a key metric called inter-object idle time, which causes many websites to have very long load time, significantly hurting the resource efficiency.

Website Load Time (WLT) is defined to be the duration from the client sending the first byte of the web/DNS request to the client receiving the last *data* packet from the web server. Note that control data such as TCP FIN can go beyond WLT. Figure 8 plots its distributions for DS1. As expected, mobile sites usually have shorter WLT values than non-mobile sites do, and warm-cache loading outperforms cold-cache loading. Websites in DS2 are loaded faster than DS1 due to lower latency in Wi-Fi networks (figure not shown).

Website Rendering Time (WRT) is defined to be the duration from the client sending the first byte to the moment when the page is fully rendered. It is measured by `WebView.PictureListener.onNewPicture()`, a callback function triggered at each time when the rendering results of the `WebView` changes. One issue of this method is that, for some pages containing animated elements (*e.g.,* an animated ad banner), the callback is being continuously triggered till the end of the data collection. We therefore eliminate from our rendering time analysis such websites (account for about 30% of all websites) whose WRT cannot be accurately determined. We plan to use more accurate metrics (*e.g.,* Speed Index [14]) to quantify WRT in future work.

WRT can be regarded as the *user-perceived* page loading time. Figure 9 plots the distributions of the proportion of rendering time within the total transfer time (*i.e.,* WRT/WLT) across websites for DS1. For 40% of the websites, their page rendering finishes before the data transfer finishes. DS2 exhibits a similar distribution.

We are interested in the origins of objects that are transferred after the page is fully rendered. We found that some of them belong to shared hosts providing "add-on" features such as user tracking (Table 7). But many more of them correspond to objects whose transfers are not necessary, such as duplicate contents and not consumed data. For example, a single load of `usmagazine.com` transfers about 650KB of advertisement images that are never displayed on the page, leading to waste of bandwidth utilization.

Figure 8 indicates the WLT can last for more than 60 seconds, because often there are long idle periods among object transfers. Such idle periods can cause energy inefficiencies due to the tail effect. Recall that a tail time (T_{tail}) is the timeout period before the radio is turned off (§1). As an example, in Figure 10, after Transfer A, if $d_1 \geq T_{\text{tail}}$, the radio will be on for T_{tail} seconds, then be turned off, and then be switched on by Transfer B. If $d_1 < T_{\text{tail}}$, the radio will be always on from A to D (D incurs another tail). The tail can be triggered by any data burst and T_{tail} can last for more than 10 seconds. Our subsequent analysis aims at understanding the root causes of idle periods like d_1 in Figure 10.

Inter-object Idle Time (IIT) depicts the idle time period when no object transfer is in progress. Recall a transfer includes both the request and the response of an object. For a transfer W starting at time t, the IIT *before* W is defined to be $\text{IIT}(W) = \max\{0, t - t_L\}$ where t_L is the latest transfer ending time for all other objects whose transfers start before t. Therefore the network is idle from time $t - \text{IIT}(W)$ to t waiting for request (but the radio might still be on). For example, in Figure 10 consisting four transfers, $\text{IIT}(B) = d_1$ but $\text{IIT}(D) = 0$ instead of d_2 since B partially overlaps with D. IIT is a conservative estimation of the part of the website load time (WLT) that can be reduced without violating the dependencies among objects. Eliminating IIT reduces WLT and saves radio energy.

Origins of Long IIT. Figure 11 plots across all websites the distribution of the maximum IIT for each mobile website load. DS1 and DS2 exhibit very similar distributions, because IIT does not consider the idle time *within* a transfer that is usually caused by the network or server latency. Instead, IIT is always caused by the delay introduced by the handset. Figure 11 indicates that for more than 20% of the mobile websites, their maximum IITs are longer than 1 second. Some IITs can be as long as 30 seconds.

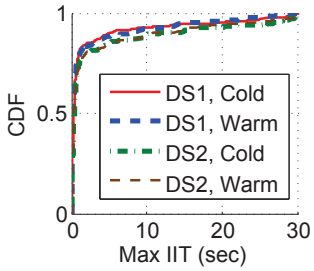

Figure 11: Max IIT across mobile sites.

We are interested in the long IITs that lengthen the WLT and therefore increase the radio energy consumption. By sampling a few long IITs, we classify their root causes as follows. *(i)* Periodic transfers triggered by Javascript. For example, ChartBeat (`http://chartbeat.com`), a popular user tracking service used by more than 30 different websites, sends a beacon to `ping.chartbeat.net` periodically (the periodicity depends on the level of user interaction). *(ii)* Delayed transfers controlled by Javascript. The object is only transferred once instead of periodically. For example, a few mobile sites (*e.g.,* `m.sprint.com`) use the TeaLeaf Customer Experience Management framework (`http://www.tealeaf.com/`), which sends some measurement data 30 seconds after the main page is loaded. *(iii)* Client computation delay such as parsing complex Javascript and CSS files. This usually happens for IITs less than 1 seconds during which the CPU usage is 100% (our testbed logs the CPU usage). *(iv)* Other delays not caused by the CPU bottleneck. For example, some sites use the Google Location Service (`https://www.google.com/loc/m/api`) for mapping a GPS coordinate to a street address. There is usually some network idle period (from hundreds of milliseconds to several seconds) before the request is sent out because the handset is obtaining its location as indicated by the GPS icon displayed on the status bar.

Among the above causes, periodic transfers impose the highest impact on the radio energy utilization. We found that compared with those generated by mobile apps [28], periodic transfers in web pages are more aggressively involving much smaller periodicities (*e.g.,* 5 seconds), which can keep the radio always on as long as the user stays on the page. We show examples of identified periodic transfers in Table 9.

4.6 Energy Impact of Timing Dynamics

We quantify the radio energy impact of three previously described issues: delayed connection closure (§4.3), the shared hosts (§4.4), and the inter-object idle time (IIT, §4.5). All of them can inject network idle time and therefore cause more radio energy consumption due to the tail effect. Our analysis methodology is as follows. For each website, we feed the original trace into a radio energy model and get the energy consumption denoted as E_0. We then construct a what-if scenario (*e.g.,* removing all shared hosts) by modifying the trace. The energy model then computes the radio energy of the modified trace as E_1. The resultant radio energy saving in the what-if scenario is $(E_0 - E_1)/E_0$. We next describe three what-if scenarios S1, S2, and S3.

S1: Silent Connection Closure. As mentioned in §4.3, most FIN/RST packets are delayed, incurring radio energy overhead. In S1, we assume silent TCP connection closure (STC) is used by eliminating FIN handshakes[8]. Similar analysis was performed in the original STC proposal [27] by applying STC to packet traces collected from a cellular carrier in an application-agnostic manner. Here we investigate whether STC is useful for web browsing.

[8]We conservatively do not remove any RST packets.

Table 9: Periodic transfer examples (DS1)**.**

Website	Periodicity
ChartBeat (30+ sites)	$\geq$ 15 sec
`m.espn.go.com`	15 sec
`comcast.net`	10 sec
`m.staple.com`	5 sec
`boston.com`	1,2,4,8,... sec

S2: No Third-Party Services from Shared Hosts. We remove TCP connections containing host names appearing in at least 10 distinct websites. The method and parameters are justified in the what-if analysis of the bandwidth impact of the shared hosts (§4.4).

S3: Reducing Inter-object Idle Time (IIT). We reduce an IIT to δ if it is longer than δ, otherwise we do not modify it. For example, in Figure 10, B, C, and D will be shifted backward by $d_1 - \delta$ if $d_1 > \delta$. The threshold δ specifies the maximum processing delay of each object. We conservatively choose $\delta = 0.5s$ which we believe is achievable if websites are well optimized for smartphones. Changing δ to 0.3s or 0.7s does not affect the results qualitatively.

In fact, to reduce IIT properly without impacting the functionalities, we need IITs' app-level semantics, which however are difficult to infer in an automated manner. We therefore consider the aforementioned upper bound of energy saving by reducing all long IITs to δ. In many cases, objects with long IITs are *delay-tolerant* (*e.g.,* periodic advertisement update and user tracking). Wisely scheduling their transfers using prefetching, batching, or piggyback [16] can eliminate IITs without impacting functionality. Also, even when there are legitimate reasons to use high-energy-overhead features such as periodic transfers, it is a good practice to provide options for users to disable them. We thus believe S3 has practical relevance.

The What-if Analysis Results are shown in Figure 12 and 13 for warm loads of DS1 and DS2, respectively (results for cold loads are similar). We leverage two smartphone radio energy models: the UMTS/HSPA model [28] and the LTE model [22]. Both models[9] take as input a packet trace and compute the radio energy consumption by simulating the Radio Resource Control (RRC) state machine used by the handset to manage the radio interface. We found both models yield very similar results in terms of relative radio energy savings so we present the results using the UMTS model. In each figure, each curve plots the CDF of energy savings across all websites. For example, a point at (0.2, 0.4) means the energy savings are at least 20% for 60% of the websites.

We describe our findings. *(i)* For 60% to 80% of the websites, S2 brings no saving. This is because although shared-hosts are popular (Table 7), their transfers often temporally coexist with other objects. Therefore removing them does not help turn the radio off. The energy savings brought by S3 is also insignificant, because many websites do not have a long IIT of more than 0.5 second (Figure 11). *(ii)* For some websites, S2 or S3 brings considerable energy savings. Fixing their resource-inefficient traffic patterns by removing the third-party objects or reducing the IIT helps save the radio energy by up to 80%. For example, by removing the ChartBeat traffic (Table 9), the overall radio energy saving for `bloomberg.com` (warm-cache load) is 66%. Note the S2 curve

[9]UMTS/HSPA model: Google Nexus One handset, DCH→FACH timer 5s, FACH→IDLE timer 12s; LTE model: HTC Thunderbolt handset, CONNECTED→IDLE timer 11.6s.

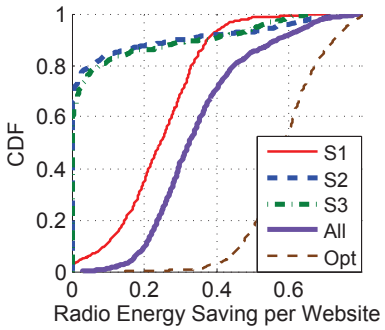

Figure 12: What-if analysis of saving radio energy by changing traffic patterns (mobile sites, warm load, DS1).

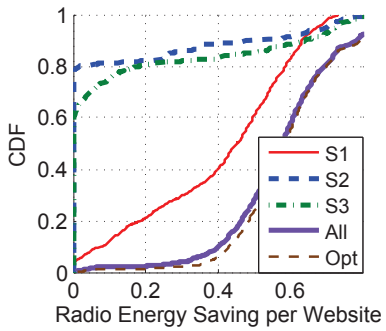

Figure 13: What-if analysis of saving radio energy by changing traffic patterns (mobile sites, warm load, DS2).

Table 10: Cache status across objects (cold load in DS1).

Cacheability Category	Mobile Sites		Non-mobile Sites	
	Objs	Bytes	Objs	Bytes
C1: Cacheable w/ explicit lifetime	51.2%	71.3%	48.3%	67.6%
C2: Cacheable w/o explicit lifetime	23.2%	19.7%	24.6%	22.2%
C3: `no-store` due to server	10.8%	2.4%	11.9%	3.0%
C4: `no-cache` due to server	12.5%	6.2%	13.8%	7.0%
C5: HTTP POST objects	0.6%	0.1%	0.7%	0.1%
C6: Other	1.7%	0.3%	0.7%	0.1%

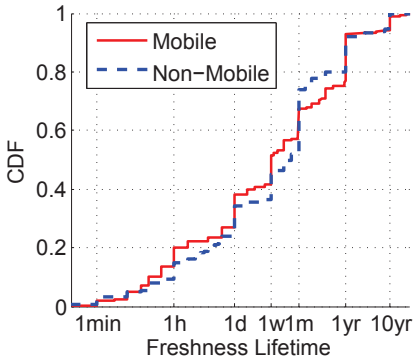

Figure 14: Freshness lifetime across objects (DS1, cold load).

underestimates the savings because hosts shared by less than 10 distinct websites can also provide third-party services but we do not remove them. *(iii)* For S1, it brings the highest radio energy savings, indicating STC is particularly effective for mobile web browsing. The median radio energy saving is 25% for DS1 and 45% for DS2. In contrast, as measured in [27], applying STC to all mobile traffic patterns yields a saving of only 11%, since many long-lived connections (*e.g.,* video streaming) cannot benefit much from STC. Also, the connection split proxy (§3.1) has small impact on the S1 savings for DS1 (Figure 12), because *(i)* the vast majority of connections in both datasets are closed by handsets instead of the proxy or remote servers, and *(ii)* the S1 curves for both datasets exhibit qualitatively similar patterns as shown in Figure 12 and 13.

The "All" curve represents a what-if scenario where S1, S2, and S3 are jointly applied. In that scenario, the median radio energy savings in Figure 12 and 13 are 32.5% and 58.7%, respectively. DS2 achieves higher savings than DS1 does, because for the same website, its overall load time (WLT) in DS2 is shorter than that in DS1, while the absolute durations of eliminated timing gaps are similar for both datasets. Therefore the removed fraction of radio-on time is higher in DS2 than in DS1.

The "Opt" curve refers to a what-if scenario same as that of the "All" curve, plus that all inter-packet time (IPT) longer than δ (0.5s) is reduced to δ. IPT is different from IIT in that IPT further includes timing gaps *within* a transfer. The "Opt" curve depicts the upper bound of the radio energy saving for each website. For DS2, the "All" and "Opt" curves are almost identical, indicating the timing gaps optimized by S1, S2, and S3 contribute to almost all idle time periods except for those caused by the high network latency (*i.e.,* high IPT), which seldom exist in DS2 but often appear in DS1 (so the "All" and "Opt" curves differ a lot in DS1).

4.7 Caching Semantics

Caching effectively reduces network bandwidth consumption and decreases user-perceived latency. In mobile networks, caching on user devices is particularly important because it can potentially eliminate any network-related overhead. Efficient caching relies on two factors: good caching *implementation* and *semantics*. The former requires a browser to strictly conform to the HTTP caching protocol [20], while the latter means proper configuration of caching parameters such as cacheability and expiration time, as the entire caching mechanism will be bypassed if, for example, the server marks all object as **`no-store`**.

Web caching implementation on mobile devices has been extensively investigated by previous work [26] so here we study the caching semantics. We have verified using the techniques introduced in [26] that our testbed (using **`WebView`** on Android 4.0.4) does follow the HTTP/1.1 caching protocol except that it does not cache byte-range responses that are not observed in the datasets. The total cache size and the size limit of a single cache entry of the **`WebView`** are measured to be 20 MB and 2.62 MB, respectively. These are greater than the maximum landing page and object sizes, respectively, in our datasets. Also note HTTP, HTTPS, and SPDY use the same caching protocol.

Object Cacheability is summarized in Table 10, which classifies each object in a cold-cache load into one of the six categories C1 to C6. C1 and C2 are cacheable objects. They differ in that C1 objects have freshness lifetime[10] explicitly set by the server using **`Expires`** and/or **`Cache-Control:max-age`** header fields, while C2 objects are without such explicit lifetime. For C2, a client is allowed to use a heuristic lifetime whose value completely depends on the implementation. C3 refers to objects that are not cacheable,

[10]The freshness lifetime is used to keep the consistency between a local copy and server's copy. Within the lifetime of an object, a client just uses the local (cached) copy. Otherwise the client needs to contact the server to revalidate the freshness of the object.

Table 11: Caching across main HTML pages (cold load, DS1).

Cacheability Category	Mobile Sites	Non-mobile Sites
C1: Cacheable w/ explicit lifetime	16.5%	15.4%
C2: Cacheable w/o explicit lifetime	33.6%	42.7%
C3: `no-store` due to server	26.0%	18.2%
C4: `no-cache` due to server	23.9%	23.8%

Table 12: Compression what-if analysis (DS1).

What-if Scenario	Mobile Sites		Non-mobile Sites	
	Cold	Warm	Cold	Warm
T1	10.8%	11.9%	11.3%	13.1%
T2	3.1%	10.3%	4.1%	13.0%
T3	13.9%	22.2%	15.4%	26.1%
T4	14.4%	24.5%	16.4%	29.0%

as indicated by the **`no-store`** cache control directive. C4 corresponds to must-revalidate objects that are tagged by **`no-cache`** or **`must-revalidate`** cache control directives. Such objects are cacheable, but a client must always revalidate with the origin server before serving them (*i.e.,* their lifetime is zero). C5 consists of objects requested using HTTP POST (usually not cacheable). Remaining objects are classified as C6.

We describe our findings in Table 10. *(i)* About 23% to 25% of the objects belong to C2, and their lifetimes are determined by heuristic values that vary across diverse HTTP client implementations: a short heuristic lifetime (*e.g.,* 30 minutes used by the **`HttpResponseCache`** library [26]) can cause unnecessary revalidation requests, while a long lifetime (*e.g.,* 48 hours adopted by the Safari browser) may lead to out-of-date local copies. *(ii)* 11% to 12% of the objects (C3) are tagged by the **`no-store`** directive so they cannot enjoy any benefit of caching. The situation is improved for those 12% to 14% of the must-revalidate objects (C4), since if the object content does not change, the server only needs to reply a small **`304 Not Modified`** response header instead of transferring the entire object. However, revalidation requests can still incur additional network round trips and potentially additional resource consumption. In Table 10, 23% of all mobile sites' objects fall into C3 or C4, meaning that every request for any of these objects will involve a request to the remote server and therefore incur at least one round trip. Also recall that in cellular networks, even transferring such small amount of data can turn on the radio and keep it on for T_{tail} seconds when no concurrent transfer is in progress. *(iii)* Both mobile and non-mobile sites share similar distributions in Table 10, implying caching semantics are not specifically optimized for mobile websites that are delivered in a more resource-constrained environment compared to their desktop counterparts.

Cacheability of Main HTML Pages is listed in Table 11. 26% (18%) of mobile (non-mobile) websites mark their main HTML pages as **`no-store`**, causing bandwidth waste for warm-cache load, as such main HTML pages are frequently accessed and their sizes are often not small. For example, the main HTML pages of **`m.bing.com`** and **`mobile.twitter.com`** are non-storable, with sizes (after compression) of 65KB and 85KB, respectively.

Freshness Lifetime. We examine the server-specified lifetime for objects belonging to Category C1 in Table 10. As shown in Figure 14, overall about 75% of the C1 objects have common lifetime values such as one hour, one day, and one month. For objects in mobile websites, their lifetime settings are statistically even shorter than those in non-mobile websites. We found many short lifetime values tend to be aggressive. For example, about 20% of the C1 objects in mobile websites have lifetime shorter than 1 hour. Within such objects, 35.7% are images and fonts, 34.3% are Javascripts, and 7.8% are CSS scripts. It is unlikely that their contents can change within such a short period of 1 hour or less. Given the resource-constrained environments of cellular networks, it is important for web developers to ensure that objects are not assigned too short freshness lifetimes.

4.8 Data Compression

Compression is widely used to reduce bandwidth consumption. Previous work (*e.g.,* [25]) characterized the effectiveness of compression on today's smartphone HTTP traffic. Our analysis goes beyond previous studies by further considering compression at the SSL and SPDY layers.

Compression support provided by HTTP, HTTPS, and SPDY are different. *(i)* For HTTP, the server can choose to compress an individual HTTP response payload (*i.e.,* object-level compression) if the client offers this option in an HTTP request. *(ii)* For SSL, the server can choose to compress the entire SSL stream before encrypting it based on the compression option negotiated during the handshake. *(iii)* In SPDY, web request and response headers are embedded in a particular type of SPDY header, and are mandatorily compressed using a customized dictionary. HTTP, by contrast, never compresses headers.

How Often is Compression Utilized? We summarize our finding for DS1 (DS2 yields similar results). *(i)* For HTTP, compression is always offered by **`WebView`** while only 31.8% of HTTP **`200(OK)`** responses are actually compressed by the server. We found that most of the under-utilization is associated with images that are already in compact formats. They account for 82.6% of non-compressed HTTP **`200`** responses. But as will be shown in Table 12, there still exist non-trivial amount of uncompressed contents compressing which can achieve additional bandwidth savings of up to 11% (13%) for cold (warm) load. *(ii)* In almost all SSL sessions, the client offers the stream-level compression option. But the option is adopted by server in only 8.4% of all SSL sessions. There can be multiple causes for such significant under-utilization *e.g.,* due to unawareness or performance concerns. Note that in fact, the gzip compression scheme used by HTTP, SSL, and SPDY is quite efficient. A single core of a commodity server can achieve throughput of 100+ Mbps [25]. Decompressing data on a commodity smartphone is even faster. So performance concerns should not prevent the use of SSL compression.

What-if Analysis. We study the following what-if scenarios T1 to T4 to understand how compression can be further improved to help reduce the bandwidth consumption. T1 corresponds to the scenario where object-level compression is fully used in HTTP/HTTPS/SPDY response payload. T2 is the hypothetical case where SPDY header compression is also employed in HTTP(S) headers. T3 combines T1 and T2 by compressing both payload and headers separately. In T4, SSL stream-level compression is fully used in HTTPS and SPDY, and it is also ported to HTTP *i.e.,* the entire TCP connection payload carrying HTTP transfers will be compressed as a whole. For each what-if scenario, we first perform the corresponding compression to get a more compact dataset, and then compare it with the original dataset to compute the relative saving of all HTTP/HTTPS/SPDY header and payload bytes.

Table 12 shows the what-if analysis results. *(i)* Full usage of object-level compression for response payload (T1) yields *additional* bandwidth savings of 11% to 13%. Such benefits mainly come from compressing Html/Javascript/CSS/XML objects that

Table 13: A summary of identified resource inefficiencies.

Layer	Description	§
TCP	Inefficient connection reuse	§4.3
SSL	Bandwidth impact of SSL	§4.2
HTTP	Impact of delayed connection closure	§4.6
HTTP	Inefficient caching semantics	§4.7
SSL/HTTP	Inefficient object/stream compression	§4.8
Content	Root causes of large contents	§4.4
Content	Impact of third-party services	§4.4, §4.6
Content	Excessive redirections	§4.4
Content	Unnecessary post-render objects	§4.5
Content	Causes & impact of in-load timing gaps	§4.5, §4.6

are not compressed in the original trace. *(ii)* For T2, surprisingly, just compressing headers brings non-trivial savings (3% to 4%) for cold-cache load. For warm load, the savings are even much higher (10% to 13%), almost as high as those of T1. This is because today's HTTP headers are not as small as one might think. For the entire DS1 dataset, the ratio between total payload size and header size is 12:1 and 3:1 for cold and warm load of mobile sites, respectively. Since HTTP and HTTPS never compress their headers, compressing them reduces the overall header size by more than 40%. *(iii)* The savings achieved by T3 is always the sum of those of T1 and T2. *(iv)* If always used, the stream-level compression (T4) can achieve more savings than T3 does, because it further eliminates redundancies across multiple objects within the same TCP connection. In contrast, T1, T2, and T3 handle each object separately. With SSL compression, HTTP header and payload traffic can be further reduced by 14% and 25% for cold and warm load, respectively, for mobile websites.

5. RECOMMENDATION

We provide detailed recommendations for improving the discovered inefficiencies summarized in Table 13.

1. High SSL Bandwidth Overhead. Mobile OS venders can use the following approaches to mitigate the overhead. *(i)* Perform lightweight SSL handshakes. Specifically, Session Identifier [17] and Session Ticket [29] can be used to resuming previously terminated SSL sessions with low handshake overheads (UbiDump can analyze both of them). Session ticket is preferred because the SSL state is stored at the client side. *(ii)* Reuse TCP connections so that multiple HTTPS/SPDY objects can share the same TCP connection and therefore SSL session.

2. TCP Connection Management. Reuse TCP connections amortizes overheads such as TCP handshake, SSL handshake, and TCP slow start. SPDY outperforms HTTP(S) in connection reuse. Also, delaying closing TCP connections can cause severe radio energy drainage. Such overhead can be eliminated by the silent connection closure scheme, which works particularly well for web browsing.

3. Large Websites and Objects. Under the constraints of delivering necessary contents and providing good user experiences, a mobile website should consume as less bandwidth as possible. The content provider should carefully determine the proper resolution of an image, keeping in mind both the context of the image, and the display capability of the requesting device. Apply "sprite images" only to small sub-images. Double check transfers after the page is fully rendered, as they are often unnecessary (*e.g.,* duplicate or not-consumed objects).

4. Image Lazy-loading Consumes More Energy. One way to mitigate the high bandwidth consumption of the "tall" pages (Table 5) is to lazily load images *i.e.,* deferring the loading of images until they appear in the client's viewport. This optimization is already implemented by Google modpagespeed [6] using Javascript tricks. In cellular networks, however, doing so can cause intermittent data transfers leading to high radio energy overhead due to the tail effect. In the worst case, the radio can be on as long as the user is scrolling. A better solution might be to split a tall page into several subpages based on content semantics.

5. Excessive Redirections (HTTP 301 and 302) should be minimized as they incur additional round trips especially in cellular networks with high latency.

6. Third-party Services (*e.g.,* ads and user tracking) providers should carefully examine traffic patterns of their data transfers. Javascript-triggered delayed or periodic transfers are particularly energy inefficient in cellular networks. In many cases, transfers incurring long inter-object idle time (IITs) are delay-tolerant or can be prefetched. Wisely scheduling their transfers using prefetching, batching, or piggybacking can minimize IITs without impacting the functionality.

7. Caching Semantics. Content providers should determine carefully if an object should really be marked as `no-store`. For those non-storable large HTML pages (*e.g.,* `mobile.twitter.com`, 85KB), one possible improvement is to separate the html into two parts: a static part and a (usually small) varying part. The static part can have a long lifetime while the varying one can be dynamic content or be marked as `no-cache`. Also, for cacheable objects, good practices are to *(i)* provide explicit freshness lifetimes, and *(ii)* avoid using unnecessarily short lifetimes.

8. Compression. The server should compress large HTML/XML/CSS/Javascript objects. For HTTPS/SPDY, performing compression at the SSL layer saves more bandwidth than using HTTP object-based compression.

6. RELATED WORK

We describe related work in three categories below.

Analyzing SSL Traffic. As described in §2, most previous studies simply ignore the content of SSL traffic [26, 23, 37]. Existing tools such as Wireshark [13] and ssldump [12] require servers' private keys for decryption. The Man-in-the-Middle (MITM) Proxy approach [8, 10] also has limitations as they may change traffic patterns and contents (§2.2). Previous work [30] leverages correlations across protocols and time for classifying HTTPS-based webmail traffic. But the technique does not decrypt the HTTPS traffic. The contribution of the UbiDump tool is it supports on-device SSL decryption without requiring private key or an MITM proxy. Further, previous analysis of SSL usually focuses on its security implications and CPU overhead. We instead study the bandwidth overhead in §4.2.

Mobile Web Measurement and Optimization. Huang *et al.* measured mobile web browsing performance early in 2009 [24]. Gember *et al.* compared web contents consumed by handheld and non-handheld devices [21]. Akamai Mobitest [1] is an online tool measuring performance for web pages loaded on mobile devices. Thiagarajan *et al.* presented a system for measuring the energy consumed by a mobile browser and applied it to diverse websites [32]. The authors provided recommendations such as shrinking Javascript and using energy-efficient image formats. Wang *et al.* studied how client-based caching, prefetching, and speculative loading help improve mobile browser performance [36]. Erman *et al.* examined the interplay between SPDY and the cellular radio layer [18]. Sivakumar *et al.* investigated cloud-based mobile web browsing and revealed that it does not provide clear benefits over the non-cloud-based approach in many cases [31].

Compared with the above work, our study provides new insights on resource efficiency of mobile browsing, considering the content, the browser, the smartphone OS design, and lower-layer protocols. The delayed FIN issue was identified by [28] and was addressed by the recent silent connection closure (STC) proposal [27]. Here we conduct a comprehensive measurement of the benefits of STC on a large number of popular websites. Vallina-Rodriguez *et al.* characterized mobile advertisement traffic [33]. We focus on a broader category of "add-on" services by examining shared hosts and quantifying their energy and bandwidth impact for top websites. Recent work [26] studied implementation issues of HTTP caching on smartphones. Study [25] investigated applying other redundancy elimination (RE) techniques such as object-level compression and delta encoding on smartphone traffic. We complement them by looking at caching semantics and by examining RE techniques used by SSL and SPDY.

General Web Optimization (not mobile-specific) has been a hot area towards which numerous efforts have been made. Recently there are new protocols and optimization frameworks such as Google SPDY [9, 35], QUIC [3], Google modpagespeed [6], and WProf [34] from both industry and academia. We believe our findings will provide them with new insights in the context of mobile devices and cellular networks.

7. CONCLUDING REMARKS

Using the UbiDump tool, we examine a wide spectrum of factors including protocol overhead, TCP connection management, web page content, traffic timing dynamics, caching efficiency, and compression usage, for the landing pages of the most popular 500 websites. As summarized in Table 13, we found that all above factors at different layers can affect resource utilization (*i.e.,* radio energy and/or bandwidth consumption) for mobile web browsing. Our findings and recommendations will assist web developers, browser developers, OS vendors, and protocol designers in making mobile web browsing more resource efficient. In our future work, we plan to thoroughly evaluate the cellular-friendliness of other web optimization approaches (*e.g.,* modpagespeed [6]) and to explore specific optimization techniques for LTE networks.

8. ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers and especially Sharad Agarwal for shepherding the paper.

9. REFERENCES

[1] Akamai Mobitest. http://mobitest.akamai.com/.
[2] Chrome for Android may get a speedy websurfing boost. http://gigaom.com/2013/03/04/chrome-for-android-may-get-a-speedy-websurfing-boost/.
[3] Experimenting with QUIC. http://blog.chromium.org/2013/06/experimenting-with-quic.html.
[4] Facebook moves all users to HTTPS for added security. http://www.pcworld.com/article/2015185/facebook-moves-all-users-to-https-for-added-security.html.
[5] General Packet Radio Service (GPRS); GPRS Tunnelling Protocol (GTP) across the Gn and Gp interface. 3GPP TS 29.060.
[6] Google modpagespeed. https://code.google.com/p/modpagespeed/.
[7] Linux WPA/WPA2/IEEE 802.1X Supplicant. http://hostap.epitest.fi/wpa_supplicant/.
[8] mitmproxy: an SSL-capable man-in-the-middle proxy. http://mitmproxy.org/.
[9] SPDY Protocol - Draft 3. http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3.
[10] SSL MITM Proxy by Stanford Crypto Group. http://crypto.stanford.edu/ssl-mitm/.
[11] The Importance Of Landing Pages. http://www.connectwisdom.com/the-importance-of-landing-pages/.
[12] The ssldump tool. http://www.rtfm.com/ssldump/.
[13] The Wireshark tool. http://www.wireshark.org/.
[14] Webpage Speed Index. https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index.
[15] Understanding Today's Smartphone User (Part 2). Informa White Paper, August 2012.
[16] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *IMC*, 2009.
[17] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246, 1999.
[18] J. Erman, V. Gopalakrishnan, R. Jana, and K. Ramakrishnan. Towards a SPDY'ier Mobile Web. In *CoNEXT*, 2013.
[19] V. Farkas, B. Heder, and S. Novaczki. A Split Connection TCP Proxy in LTE Networks. In *Information and Communication Technologies*, volume 7479, pages 263–274. 2012.
[20] R. Fielding, J. Gettys, J. Mogul, H. F. L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol - HTTP/1.1 . RFC 2616, 1999.
[21] A. Gember, A. Anand, and A. Akella. A Comparative Study of Handheld and Non-Handheld Traffic in Campus Wi-Fi Networks. In *PAM*, 2011.
[22] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. A Close Examination of Performance and Power Characteristics of 4G LTE Networks. In *Mobisys*, 2012.
[23] J. Huang, F. Qian, Z. M. Mao, S. Sen, and O. Spatscheck. Screen-Off Traffic Characterization and Optimization in 3G/4G Networks. In *IMC*, 2012.
[24] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing Application Performance Differences on Smartphones. In *Mobisys*, 2010.
[25] F. Qian, J. Huang, J. Erman, Z. M. Mao, S. Sen, and O. Spatscheck. How to Reduce Smartphone Traffic Volume by 30%? In *Proc. of Passive and Active Measurement conference (PAM)*, 2013.
[26] F. Qian, K. S. Quah, J. Huang, J. Erman, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Web Caching on Smartphones: Ideal vs. Reality. In *Mobisys*, 2012.
[27] F. Qian, S. Sen, and O. Spatscheck. Silent TCP Connection Closure for Cellular Networks. In *CoNEXT*, 2013.
[28] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Profiling Resource Usage for Mobile Applications: a Cross-layer Approach. In *Mobisys*, 2011.
[29] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig. Transport Layer Security (TLS) Session Resumption without Server-Side State. RFC 5077, 2008.
[30] D. Schatzmann, W. Muhlbauer, T. Spyropoulos, and X. Dimitropoulos. Digging into HTTPS: Flow-Based Classification of Webmail Traffic. In *IMC*, 2010.
[31] A. Sivakumar, V. Gopalakrishnan, S. Lee, S. Rao, S. Sen, and O. Spatscheck. Cloud is not a silver bullet: A Case Study of Cloud-based Mobile Browsing. In *Proc. of ACM HotMobile*, 2014.
[32] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. Singh. Who Killed My Battery: Analyzing Mobile Browser Energy Consumption. In *WWW*, 2012.
[33] N. Vallina-Rodriguez, J. Shah, A. Finamore, H. Haddadi, Y. Grunenberger, K. Papagiannaki, and J. Crowcroft. Breaking for Commercials: Characterizing Mobile Advertising. In *IMC*, 2012.
[34] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. Demystifying Page Load Performance with WProf. In *NSDI*, 2013.
[35] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. How Speedy is SPDY? In *NSDI*, 2014.
[36] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie. How Far Can Client-Only Solutions Go for Mobile Browser Speed? In *WWW*, 2012.
[37] Q. Xu, J. Erman, A. Gerber, Z. M. Mao, J. Pang, and S. Venkataraman. Identifying Diverse Usage Behaviors of Smartphone Apps. In *IMC*, 2011.