PARCEL: Proxy Assisted BRowsing in Cellular networks for Energy and Latency reduction

Ashiwan Sivakumar[†] Shankaranarayanan P N[†] Vijay Gopalakrishnan[‡]
Seungjoon Lee[⋆*] Sanjay Rao[†] Subhabrata Sen[‡]
[†]Purdue University, [‡]AT&T Labs - Research, [⋆]Two Sigma

ABSTRACT

Today's web page download process is ill suited to cellular networks resulting in high page load times and radio energy usage. While there have been notable prior attempts at tackling the challenge with assistance from proxies (cloud), achieving a responsive and energy efficient browsing experience remains an elusive goal. In this paper, we make a fresh attempt at addressing the challenge by proposing *PARCEL. PARCEL* splits functionality between the mobile device and the proxy based on their respective strengths, and in a manner distinct from both traditional browsers and existing cloud-heavy approaches. We conduct extensive evaluations over an operational LTE network using a prototype implementation of *PARCEL*. Our results show that *PARCEL* reduces page load times by 49.6%, and radio energy consumption by 65% compared to traditional mobile web browsers. Further, our results show that *PARCEL* continues to perform well under client interactions, owing to its judicious functionality split.

Categories and Subject Descriptors

C.4 [**Performance of systems**]: Design studies; Measurement techniques; C.2.2 [**Computer communication networks**]: Network Protocols—*Applications*

Keywords

Proxy-assisted Browsing; Mobile Web; Energy Consumption; Cellular Networks; Smartphones; Cloud Browsers; Web Optimization

1 Introduction

Along with the spread of higher speed cellular technologies like 3G and LTE, the past few years have witnessed an explosive growth in mobile Internet data traffic (projected to increase 11-fold between 2013 and 2018 [4].) Web browsing is a key activity on mobile devices, accounting for more cellular traffic than any other application, excluding multimedia streaming [24]. There exists tremendous interest in improving user's Quality of Experience (QoE) for the mobile web. Key challenges include the resource constraints of common cellular devices like smartphones and tablets, and the radio access network (RAN). While processing capabilities of mobile devices have dramatically improved in recent years, mobile device battery energy seems likely to remain a major resource limitation for the foreseeable future.

Several factors make current approaches to web downloads ill-suited to cellular networks (§2). On the one hand, web pages consist of hundreds of objects spread over multiple server domains, and downloading pages involves a large number of HTTP request-response interactions. On the other hand, cellular networks involve large round-trip times resulting in substantially longer download times compared to wire-line. The delays are exacerbated since initial objects fetched during the download (e.g., HTML, style sheets (CSS), JavaScript (*JS*)) may need to be processed to identify what objects to fetch subsequently. Higher download latencies and frequent short data transfers in turn leave the radio in a high power state for longer duration, resulting in increased cellular radio energy usage [25].

Some notable prior attempts have been made in tackling the challenges associated with web downloads on cellular networks [2, 5, 6, 8, 21, 35, 36] by leveraging proxies to enhance performance.[1] However, while important first steps, these prior efforts fall short in several ways (e.g., real-world web page design and user interactivity can result in some of these approaches increasing user perceived latencies and radio energy usage [32]), as we describe in detail in §3.

In this paper, we seek to better realize the potential of such proxy-assisted approaches by addressing the question: *what should be the right division of web download functionality between the mobile device and the cloud?.* Our primary goal is to improve user experience by reducing page download times and radio energy consumption over the entire user session, covering initial page download as well as subsequent user interactions with the page. We focus on radio energy consumption, since studies show that the power consumed by the cellular radio interface contributes a considerable fraction (1/3 to 1/2) of the total device power consumption for normal workloads [26].

To this end, we present *PARCEL*, a new proxy-assisted mobile web-browsing system (§4). The key ideas underlying

[*]This work was done when the author was employed at AT&T Labs

[1]By proxies, we refer to well-provisioned servers with good network connectivity. We also use the terms proxy and cloud interchangeably.

CoNEXT'14, December 2–5, 2014, Sydney, Australia.
Copyright 2014 ACM 978-1-4503-3279-8/14/12:$15.00.
http://dx.doi.org/10.1145/2674005.2675004.

PARCEL are: (i) perform object identification and download at the proxy, leveraging its superior network connectivity; (ii) support interactive operations locally at the client to avoid network communications to the extent possible; and (iii) support cellular friendly data-transfers by greatly reducing the number of HTTP request-response interactions, and by providing the proxy with the flexibility to push objects in a manner that balances latency and radio energy use.

Realizing *PARCEL* requires us to address a number of important system issues. To demonstrate our ideas, we have implemented an initial custom Android-based browser prototype of *PARCEL* using the Webview library, addressing many pragmatic considerations (§5). We discuss how the flexibility of data transfer provided by *PARCEL* may be exploited by proposing multiple scheduling strategies and analyzing the performance trade-off between page load time and radio energy usage (§6).

We validate *PARCEL* through extensive evaluations in live LTE network settings, and compare its performance to both a traditional web browser, and an existing cloud-heavy browser. We also evaluate multiple policies for scheduling data transfers from the proxy to the client within the *PARCEL* framework. Our evaluations employ a carefully crafted methodology to ensure that the performance comparisons are not affected by variability in LTE signal strengths or the web pages themselves (§7).

Our results are promising (§8). They show that *PARCEL* can reduce web-page latencies by 49.6% and radio energy consumption by 65% on average compared to conventional web-browsers. Further, unlike a popular cloud-heavy browser, *PARCEL* continues to perform well with client interactions. Overall, these results indicate that judiciously splitting functionality between the mobile device and the proxy can substantially enhance user browsing experience in cellular network settings, and show that *PARCEL* is a promising step towards this end.

2 Web download in cellular networks

In this section, we discuss why the web download process is a poor fit for cellular networks.

2.1 Modern web-pages and pageload process

Modern web-pages are complex constructs, easily comprising of tens to hundreds of static and dynamic objects (banners, images, style-sheets, multiple different types of *JS* files, etc.) from multiple different domains. An analysis of the Alexa top-500 web pages indicates that 40% had at least 100 objects (20 *JS* files). Further, the individual objects are typically small (a few KB) to moderately sized (a few MB). Across all the pages, the 95^{th}, 80^{th} and 50^{th} percentile of the object sizes were $386, 107$ and 18 KB respectively.

Figure 1 depicts the various stages involved in loading a page at a typical mobile browser. The browser first initiates a DNS lookup to resolve the domain address for the main page URL, and fetches the main page from the content web server using the HTTP protocol. It then begins parsing the page and dynamically builds and updates the Document Object Model (DOM) tree (an in-memory data structure to represent the parsed nodes in a page). As it encounters new objects in the page that are not available locally, it initiates new HTTP requests to the relevant servers for those objects. There are inter-dependencies among objects – e.g., downloaded *JS* files have to be executed, which

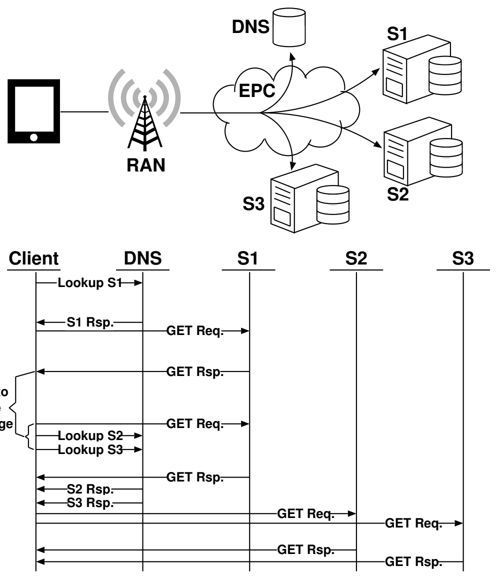


Figure 1: Web-page load process in mobile networks.

in turn may lead to additional new objects (including more *JS* files) being downloaded. The resulting network traffic pattern typically consists of a large number of short data transfers, related to (i) establishing distinct TCP connections per-domain; (ii) DNS lookups to resolve the potentially large number of servers involved; and (iii) a HTTP request/response associated with each object.

Measuring page latencies: We next discuss typical metrics used to quantify web page download latencies. An *Onload* event is triggered by the browser when it has received sufficient objects for rendering an initial version of the page. The time from the request initiation to the time of the Onload event, is referred to as the *Onload time* (**OLT**) of the web page. OLT is a commonly used Key Performance Indicator (KPI) for measuring the latency of the page load process and indicates the initial responsiveness of the page. Note that objects can be requested by the page even after the OLT [33]. This happens due to the presence of *asynchronous* JS files, often used for displaying independent sections of the page like advertisements and chat widgets in parallel to the main web page. We define the time required to fetch all objects required by the page beyond OLT and in the absence of any user interaction as the *Total pageload time* (**TLT**).

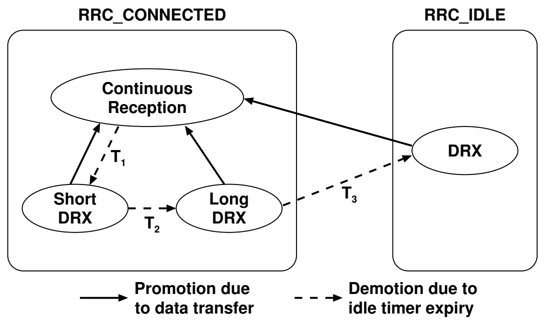


Figure 2: LTE RRC State Machine

2.2 Cellular network characteristics

A key determinant of performance in cellular networks is how the application traffic interacts with the LTE Radio

Resource Control (RRC) State Machine [11, 18] The device consumes very different levels of radio energy in the different RRC states(Figure 2). The device has to be in the highest energy state (*Continuous Reception* or CR in the CONNECTED state) for any data transfer to occur. The Discontinuous Reception (DRX) modes enable the device to trade off some responsiveness for energy savings (different trade-offs for Short and Long DRX) within the CONNECTED state: it actively polls for data transfers periodically and turns off the radio at other times, thereby consuming less power compared to CR, but higher power than the IDLE state. For the radio to transition to IDLE, typically the device has to be idle for > 10 sec. If the device is in IDLE or DRX states, and a packet needs to be sent/received, the device needs to get promoted to the CR state. Research has shown that small data transfers on LTE are very costly and large data bursts are much more energy efficient as a device in the CR state consumes significant base energy even if it uses a small fraction of the available network bandwidth [25].

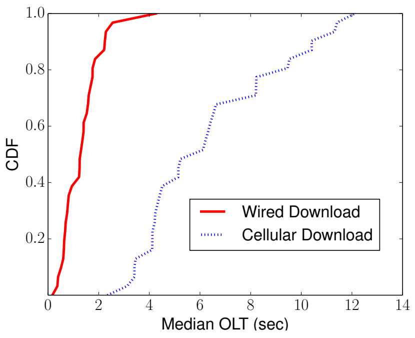


Figure 3: Median OLTs on Cellular and Wired

2.3 Page load performance in cellular networks

The interplay between web-page downloads and cellular network characteristics leads to two key concerns:

• *High latencies:* Figure 3 shows a CDF of the median OLT for a subset of the Alexa top-500 web-pages when downloaded using an LTE network and a wired network. The OLT with LTE is > 6 sec for 50% of the pages, with the maximum being about 13 sec. For the wired network, the corresponding values are 1.1 sec and 4 sec. Web downloads on LTE networks incur high latencies since typical RTTs for LTE are high (of the order of $70 - 86$ msec. [18]), and since initial objects fetched during the download must be processed to determine what objects to subsequently fetch. The use of traditional web proxies [9] partially ameliorates the situation since DNS resolutions are performed by the proxy. However, the solution still involves a large number of short data transfers due to the request-response semantics of the HTTP protocol.

• *High radio energy usage:* The typical web download pattern of a large number of short data transfers over several seconds means the device has to stay in CONNECTED state for an extended period of time, frequently transitioning to CR. Because of the energy characteristics of the RRC state machine described above, such a traffic pattern can use up a lot of radio energy.

3 Related Work

There has been interest in the industry [2, 5, 6, 8, 17] and academia [21, 35, 36] to address the limitations of web browsing in general and cellular networks in particular. The techniques used may be classified as:

Offloading processing to the cloud: Many existing solutions have advocated a cloud-heavy, thin-client approach [6, 8, 36] with the cloud performing most of the compute intensive tasks including parsing and rendering web pages, and *JS* execution. This approach has the potential to reduce CPU usage on the mobile device as well as device CPU energy consumption, since most processing is done on the cloud. However, real world web page design and cellular network characteristics have meant that the performance of these browsers has not lived up to their promise. For instance, recent work [32] shows that interactive user actions (e.g., mouse hover, button clicks) with cloud-heavy browsers incur higher latency and device radio energy consumption than traditional browsers. This is because while conventional browsers execute *JS* associated with the events locally, cloud-heavy approaches require communicating these events back to the cloud, executing *JS* remotely and transferring the results back to the client. Consequently, the total device energy consumption with these browsers may in fact be *higher* than conventional browsers. We present a detailed example in § 8.2 to demonstrate these limitations of cloud-heavy approaches and how *PARCEL* performs better. Our approach with *PARCEL* is not to blindly offload processing to the cloud, but rather determine the right split in functionality between the cloud and the device.

Splitting functionality between cloud and device: Silk [2] claims to run browser functions both on the device and in the cloud. Since Silk is closed source, little is publically known about what functions Silk offloads to the cloud or under what conditions. Moreover, in real-world testing, several sites have recommended disabling cloud-based acceleration in Silk to improve page loading speed [1, 3]. A recent position paper [35] suggests off-loading partial logic to the cloud, with the cloud sending back a page that consists of processed and unprocessed portions back to the mobile device. The paper, however, does not provide details on how to achieve this. In contrast, we present a detailed design and implementation, and present comprehensive evaluations of our approach.

Data transformation and compression: A number of browsers [2, 5, 6] offer support for data transformation and compression in the cloud, with the primary focus being to reduce the amount of data downloaded. While useful, these techniques do not address the causes for high page load latencies (multiple RTTs associated with per-object http request response semantics). Further, these techniques by themselves do not lower device energy requirements, and can in fact hurt radio energy consumption since the time for compression and transformation could result in the radio staying in the high energy state for a longer duration [32]. That said, compression and transformation is orthogonal to our work, and can be easily integrated with *PARCEL*.

New protocols (e.g., SPDY): There is increasing interest in devising protocol-based approaches that overcome TCP's limitations. For example, SPDY [17] establishes only one connection per server domain and multiplexes the transfer of objects from that domain. SPDY also eliminates the need for one outstanding request per connection and supports the ability for the server to "push" objects to the client. There have also been proposals to deploy cloud-based proxies that use SPDY between the client and the proxy [2, 5, 35]. The performance improvements with SPDY in the real-world, however, are mixed [16,34]. Inter-dependencies in web pages due to *JS* and CSS and the relative lack of power of mobile browsers implies that all the objects cannot be requested in parallel even though supported by SPDY [16,34]. Further, each server has to explicitly support server push with SPDY. Given that objects are requested across a range of domains, the gains from server push are unclear. Further, the SPDY proposal does not specify how to implement server push in the presence of proxies.

Mobile offload beyond web-browsing Many works [14, 15,19,20,28,29,31] have investigated issues around offloading code of generic applications (e.g., compute intensive face recognition applications) to the cloud, primarily to reduce computation time and save device energy. Mobile web browsing has unique issues such as multiple round-trip times associated with data transfer, and is different in that data naturally flows into mobile devices from remote servers, with cloud servers potentially on the data path from the server to the device. Recent work has looked at energy efficient transfer of video streams in LTE networks [27], but web-browsing poses issues quite different from video streaming.

4 *PARCEL* approach and design

We describe the design of *PARCEL*, our objectives behind that design, and how our design addresses the problems with cellular web browsing. We then discuss some of the practical considerations in realizing our design.

4.1 Design Considerations

Our goal with *PARCEL* was to improve user experience when browsing on cellular networks by reducing page load times and radio energy usage, a key component of total device energy consumption. To that end, our key design considerations were:

Minimize per-object HTTP request-response interactions: Traditional browsers involve per-object HTTP request-response interactions. Given the high RTTs of cellular networks, we seek to avoid HTTP request-response interaction with the browser for individual objects.

Responsive and radio energy efficient client interactions: Cloud-heavy solutions that offload *JS* execution to the proxy entail network communication with the proxy to support client interactions. This results in lowered responsiveness and increased radio energy usage [32]. A key consideration in designing *PARCEL* was to handle dynamic page changes and user interaction locally at the client to avoid network communication.

Cellular-friendly and latency-sensitive data transfer: To reduce radio energy consumption, it is desirable to bundle data to and from the client. However, in doing so, it is important to take the page load process into account, and ensure page latencies are not impacted. Our goal in designing *PARCEL* is to balance these multiple considerations.

4.2 *PARCEL* design

The *PARCEL* architecture is designed to meet the objectives listed above. *PARCEL*, as shown in Figure 4, splits the typical browser functionality between a browser installed on the mobile device and a proxy in the cellular network. Like with traditional browsers, when users enter or select a URL, the browser issues a request for this URL to the proxy. On receiving the request, the *PARCEL* proxy does necessary

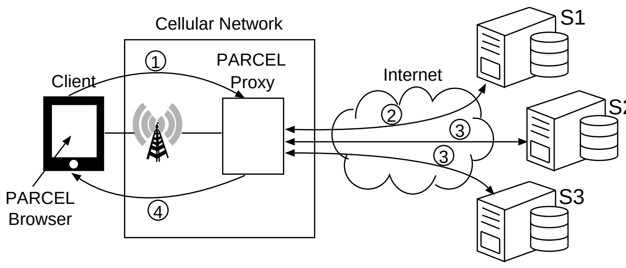


Figure 4: *PARCEL* Architecture

DNS lookups, and requests the URL from the corresponding web server, much like traditional proxies (steps (1) and (2)).

This is where *PARCEL* starts to deviate from traditional browsers and proxies. On receiving the response from the web servers, rather than just forwarding it to the client, the *PARCEL* proxy starts to parse the web page and identify all the objects that are required to successfully render the page in its entirety. This not only includes parsing the HTML page to identify the objects in that page, but also processing all *JS* since these objects may also refer to other dependencies or point to objects that are dynamically identified for this request. The proxy then goes ahead and requests these identified objects without the client requesting them (step (3) of Figure 4). In that process, the proxy may choose to apply any necessary compression and/or transformations (transcoding images, shrinking size, etc) to these objects as they arrive. It then collects all these objects and transfers them to the client.

The client, for its part, receives the main HTML page and all the objects associated with the page from the proxy. It then behaves like a traditional browser in that it parses the HTML files and identifies the objects on that page. Unlike traditional browsers, however, the *PARCEL* browser does not issue requests for these objects since the objects are proactively fetched by the proxy and made available as part of the collection transferred to the browser. The client browser uses the meta-data associated with the collection to identify the right object and use it for rendering the page. As part of this process, the client also parses all CSS files and processes *JS*. While there seems to be repetition of work at the proxy and the client, our design allows for all interactivity to be locally handled by the client browser (and therefore keep it responsive and energy efficient). That said, our design does not preclude optimizations that allow the client to leverage the work done by the proxy as part of its rendering process (like using some of the *JS* processing).

4.3 Benefits of *PARCEL* design

While conceptually simple, the *PARCEL* design addresses many of the important issues with mobile web browsing.

• By sending just the URL request to the proxy and no other requests, *PARCEL* reduces the number of round trips from the client on the higher latency cellular link, thereby reducing page load times. We envision that the proxy will be implemented on a powerful server with lots of processing power, high bandwidth and low latency to the Internet. Consequently, having the proxy identify which objects to download (through html parsing and *JS* processing) allows for fast downloads of these objects (refer Figure 3).

• By getting the browser to just send the URL, we get the client to not continuously use up radio energy. Similarly, by getting the proxy to push objects, we give it the flexibility to

	HTTP proxies [9]	SPDY proxies [5, 16]	Cloud browsers [6, 8, 36]	*PARCEL*
# of TCP connections	many	single	single	single
# of HTTP requests	per object	per object	single	single
Object identification	client	client	proxy	proxy
Interactive *JS*	client	client	proxy	client
cellular-friendly transfer	$\times$	$\times$	$\times$	$\surd$

Table 1: *PARCEL* vs. existing approaches.

bundle and schedule object transfers to the client efficiently, in a cellular friendly way.

• Compared to cloud-heavy approaches, *PARCEL* executes *JS* locally which minimizes network communications during client session, and allows for responsive and energy efficient client interactions [32]. Though, cloud-heavy approaches may lower device CPU energy consumption by executing *JS* remotely, this may be outweighed by the radio energy consumed due to client interactions (§ 8.2). *PARCEL* does not incur any more device CPU energy consumption than conventional browsers or proxies, and CPU energy consumption could potentially be lowered by allowing the client to leverage the work done by the proxy.

• The advantages of *PARCEL* above continue to hold even if a protocol such as SPDY is enabled between the client and the proxy since the performance with SPDY is limited by how quickly the (less capable) mobile client issues requests for objects in a web-page [16].

We contrast *PARCEL* with existing efforts and summarize the key differences in the approaches in Table 1.

4.4 Cellular friendly data transfer

Once the *PARCEL* proxy identifies and downloads the different objects on a specific page, it has the flexibility of transferring the different objects to the client in a manner that is cellular friendly. Figure 5 illustrates the different approaches possible with *PARCEL*.

IND: In this scheme, shown in Figure 5(b), the proxy transfers objects to the client as and when it receives the objects. Apart from reducing the energy consumption, this has the potential to significantly reduce OLT because it eliminates the need for the client to make requests and allows the client to start processing HTML and *JS* objects as they arrive.

ONLD: An alternate approach, shown in Figure 5(c), is for the proxy to fetch all objects from the web servers and transmit the data in a single batch to the client. This approach trades off increased OLT (compared to *IND*), for the potential to further reduce energy consumption. This is because, the client can go into idle mode when the proxy is downloading the data and then receive all the data as a single bundle from the proxy.

PARCEL(X): A final approach, shown in Figure 5(d) is to strike a balance between the latency energy benefits. With *PARCEL(X)*, the *PARCEL* proxy does not wait for all objects to arrive. Instead, it starts transferring data to the client when a certain threshold of data (X) becomes available or if the onload event is detected. This allows us to transfer objects quickly to the client for processing while reducing the number of state transitions of the client radio. We explore and evaluate these approaches further in § 8.

4.5 Practical challenges and solutions

In this section, we describe some of the practical challenges that need to be addressed in the realization of *PARCEL* design, and our proposed solutions.

Suppressing object requests The *PARCEL* browser starts parsing the main HTML page as soon as it is received and identifies the objects needed to display the page. If all the objects in the page are part of the received bundle, it can immediately start the rendering process. The browser, however, will not have the entire set of objects when it receives the HTML page with *IND* or *PARCEL(X)*. The fact that some objects are requested after onload can cause the bundle with *ONLD* to also not have all the objects. Finally, in rare cases, the use of *JS* can cause the object URL as determined by the *PARCEL* browser to differ from that by the proxy. In all these cases, the browser has to decide when to request the object because the missing object could well be on flight from the proxy to the browser. Our approach is for the browser to suppress making requests for any objects it has identified as needed but not already available. When the proxy determines that it has downloaded all the objects it sends a notification to the browser of such completion. If there are outstanding objects after such notification, the browser can request for these missing objects.

Determining page completion: In order to send the completion notification, the proxy has to determine when it has completed the page download (traditionally indicated by the onload event). Since this approach would not work for pages that request objects even after the onload, we resort to typical page statistics to make this determination. An analysis of the Alexa top-500 web pages indicates that the inter-arrival time of objects is less than 5 sec for 95% of the objects after onload. Based on this, our proxy implements a simple heuristic where it waits for a short time period of inactivity between the proxy and the server (after onload) and then determines the page complete. In the rare event that the proxy misses some object, the browser can still request for those missing objects.

Client properties and customization Web-pages have browser-dependent code that downloads objects or renders web page differently for different browser implementations. Similarly it could check for the type of access device (smartphone vs. laptop) and send objects tailored for the screen size. In *PARCEL*, since web servers receive requests from the proxy rather than the browser, we need to inform the proxy of the relevant attributes about the client (device) and its browser. e.g., the client sends these attributes (user-agent, mobile device screen information) to the proxy when it connects to the proxy and requests the URL. The proxy then uses this information to emulate the client while downloading the page.

Caching and cookies The *PARCEL* design allows both the browser and the proxy to cache data. However, since the browser does not request for objects other than the main URL, the proxy is unaware of the objects that are cached at the client. Handling cookies and other session objects is challenging for the same reason. One approach is for the proxy to track the object versions sent to the client, which helps avoid redundant transfer of objects. We prefer a model where these proxies are deployed as personalized proxies [12] for each user, running on virtual machines.

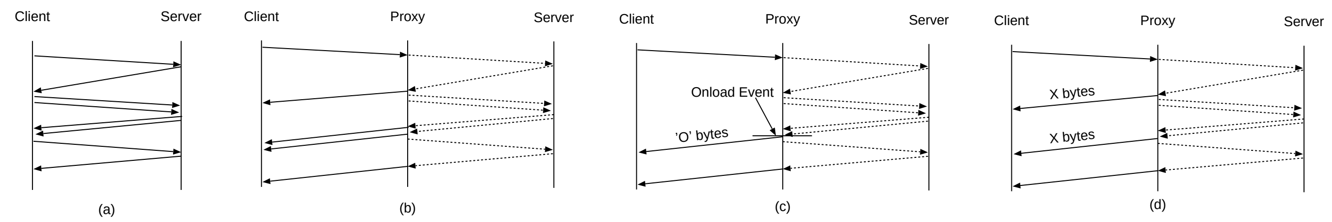


Figure 5: Download patterns of traditional browsers (a) and *PARCEL* with *IND*, *ONLD* and *PARCEL(X)* (b-d)

This allows the proxy to easily mirror the state of the objects (cache and cookies) stored at the client. While proxies could be located in Amazon EC2-like public clouds, they could also be deployed similar to middle-boxes within the cellular networks in the near future. Further, as approaches like CloudRAN [37] get more popular, cellular operators can deploy these proxies closer to the edge and have a tighter coupling with the cellular base stations for optimal energy efficiency. We hope to explore these alternatives as part of our future work.

Handling HTTPS and POST For encrypted pages, *PARCEL* falls back to the traditional way of downloading web pages since the proxy cannot parse and identify objects. However, using personalized proxies where the user trusts the proxy, HTTPS requests can potentially be handled by setting up independent secure channels between the browser and server. The *PARCEL* proxy handles POST requests by relaying them to the server as received from the client. If the POST response from the server is a HTML, the proxy processes the response as explained in § 4.2 before sending it out to the client. For HTTP responses that do not have content (e.g., HTTP 204), the proxy simply forwards them unmodified to the client.

5 Prototype implementation

In this section, we describe the prototype implementation of *PARCEL*. There are three main objectives we seek to demonstrate with our implementation: (a) efficacy of *PARCEL*, (b) real-world performance and energy improvements with *PARCEL* and, (c) how existing software can be reused to build *PARCEL*.

5.1 Proxy implementation

While many web proxy implementations exist, we cannot simply reuse these implementations as *PARCEL* requires the proxy to not only parse HTML files but also process CSS and *JS* (or in essence behave like a browser). To achieve this capability, we use an existing browser (*Mozilla Firefox*), sufficiently extended, to act as our proxy. We used the *JS* server sockets API to implement server functionality in the proxy, which on receiving a URL request from the browser, invokes Firefox to load the page and execute *JS*. Using the Firefox extension framework, we developed an extension that intercepts HTTP responses received from the web server. The interception mechanism helps in bundling and scheduling the responses to the client and in minimizing the impact of rendering of the Firefox process on the proxy's performance. Our extension includes an event listener that detects onload event at the proxy and begins data transfer to the client in the *ONLD* and *PARCEL* (X) schemes.

We use MHTML (a standard HTTP file format) to bundle and transfer data from the proxy to the client browser. MHTML is a low overhead format that allows inclusion of HTTP headers along with the object data in the bundles. Further, MHTML is supported by most modern browsers (directly or through third-party libraries), which enables them to parse and directly render bundles transferred by *PARCEL* proxy. Since browsers implement single-threaded execution of *JS*, we use asynchronous IO in our proxy implementation to decouple the proxy-server path from the cellular path when transferring objects to the client. Further, our implementation ensures that the proxy writes enough data to the kernel buffer and proceeds with execution until a callback is received when the buffer is available.

5.2 Client implementation

While the *PARCEL* client browser differs from traditional browsers in how it requests/receives objects and in the information sent to the proxy, the parsing and rendering process do not change. Hence, our client implementation reuses existing parsing and rendering functionality of browsers while modifying how the objects are requested and downloaded. While it is easier to write extensions for existing browsers to realize *PARCEL* functionality on clients (e.g., intercepting and manipulating HTTP requests/responses using HTTP Channel Observers in Firefox XUL runner framework), support for such extensions is not yet mature on mobile platforms. Hence, we build our own custom browser on top of the Webview library for Android devices. Webview allows us to integrate browser functionality into applications and uses the WebKit rendering engine to render the web pages. In our implementation, the client browser application accepts URLs from the user and sends the request to the *PARCEL* proxy over TCP. Our client also receives the MHTML bundles as part of the response from the proxy and sends them out to Webview for rendering.

6 PARCEL model and analysis

In this section, we present a simple analytical model to provide insights into the trade-offs between the various data transfer schemes presented in § 4.4. Suppose B is the aggregate object size at the time of page onload event. Let us assume that we use n equal-sized bundles and that the proxy sends out the n-th bundle as soon as the onload event occurs at the proxy (We approximate IND as a case with large n). We further assume that the $(n-1)$th bundle transfer was complete before the proxy onload event. We define $s(n)$ to be the download speed between *PARCEL* proxy and client with n bundles. We denote the OLT at the proxy as T_p, which we assume is independent of the number of bundles.

Radio energy We first focus on the radio energy at the time of client onload event. Let us assume that after each bundle transfer, the client always completes CR_{tail} and SDRX cycles. We denote the duration of each cycle to be d_c and d_s. Let us consider the aggregate duration of LDRX cycle $d_l(n)$ before the n-th bundle arrives to the client: $d_l(n) = T_p(n) - \frac{n-1}{n}\frac{B}{s(n)} - (n-1)(d_c + d_s)$. This is from simply accounting for the transmission and state transition time for $(n-1)$ bundles before the transmission of the n-th bundle. Then, we add up the radio energy for LDRX, state transition, and actual transmission to obtain the total radio energy at the time of client onload: $E(n) = p_l d_l(n) + (n-1)(p_c d_c + p_s d_s) + p_c \frac{B}{s(n)}$, where p_l, p_c, and p_s are power consumption in LDRX, CR, and SDRX states, respectively.

For ease of exposition, we assume that $s(n) = s$ for all n and that $E(n)$ is a continuous and differentiable function of n.[2] By solving $E'(n) = 0$, we can find an optimal bundle count n^* that minimizes the radio energy: $n^* = \frac{1}{\alpha}\sqrt{\frac{B}{s}}$, where $\alpha = \sqrt{((p_c - p_l)d_c + (p_s - p_l)d_s)/p_l}$. Then, it follows that the optimal bundle size b^* is:

$$b^* = \frac{B}{n^*} = \alpha\sqrt{sB}. \quad (1)$$

Eq. 1 matches intuition. First, for higher download speeds, larger bundles are more acceptable since there is less penalty in waiting longer. Second, as web pages become larger, using larger bundle sizes ensures the total number of bundles, and the associated state transition overhead is limited. Lastly, α captures relative radio state transition overhead due to radio technology and parameter settings - as transition overhead increases, it becomes important to use fewer bundles and reduce the state transition overheads.

Trade-off between energy and time By simply accounting for the transmission time of the n-th bundle after onload at the proxy, we get: $\text{OLT}(n) = T_p + \frac{1}{n}\frac{B}{s(n)}$. Assuming $s(n) = s, \forall n$, OLT(n) is a decreasing function of n, which agrees with our intuition that OLT is likely better with smaller bundles. On the other hand, from Eq. 1, performance in radio energy depends on various web page and network characteristics. For example, for a 2MB page, with download speed of 6Mbps, and $\alpha = 0.74$ (derived from LTE parameter values obtained as described in § 7), the optimal bundle size is approximately 0.9MB. In our evaluation, we experiment with different bundle sizes and explain the findings using the analysis result (§ 8.3).

7 Evaluation Methodology

Downloading web pages over a cellular network is subject to high variability due to dynamically changing web pages and radio signal fluctuation. In this section, we describe our experiment setup in which we limit the variability in both aspects and ensure fair comparison across different schemes.

7.1 Schemes for comparison

We compare *PARCEL* with a popular traditional mobile browser for Android that we call **DIR**.[3] For the comparison with DIR, we use the IND scheduling scheme for *PARCEL* to help us understand the benefit of *PARCEL* proxy in isolation. We also separately compare different scheduling schemes of *PARCEL* to understand the relative performance benefits due to bundling. Finally, to evaluate *PARCEL* under client interactivity, we contrast its energy usage with a popular cloud browser (we call **CB**) that performs all *JS* execution functionality in the cloud, as we discuss in § 8.2.

Metrics Our primary metrics of interest are *OLT* and *TLT* at the client as defined in § 2. To compute these quantities, we collect packet traces on the device for each experiment. For OLT, we compute the time between the first SYN and the last ACK for all objects required to generate the onload event at the client. For TLT, we compute the time between the first SYN and the last ACK for all objects in the trace. For each experiment, we limit the packet collection for 60 seconds, which we found sufficiently long for all objects of each page to be downloaded with all schemes. Note that we do not include rendering time, which is typically small (median rendering time is 0.436 sec) and is comparable for *PARCEL* and DIR.

Another metric that we use is the *radio energy* consumed by the LTE radio interface for different schemes. We calculate the radio energy using the open source ARO tool [18], which captures the Radio Resource Control (RRC) state transitions and the corresponding energy consumption levels by performing fine-grained simulation on the packet traces collected from the client device using a pre-computed model. We conducted new measurements to generate the model parameters specific to our phone (Samsung Galaxy S3) and experimental location (since power values are device-specific and timer values are periodically tuned by operators) by using the same methodology [18]. However, the relative power hierarchies of the different RRC states (as described in § 2.2) remain the same across all settings. In addition to using the energy model, we also report actual measurements of total device energy consumption using a power meter (See § 8.2). Unless otherwise mentioned we compare the schemes on the **median values** of all the metrics in § 8.1, 8.3 and 8.4.

7.2 Robustness to radio network variability

All experiments were conducted over a production LTE network in West Lafayette, IN, using a Samsung Galaxy S3 phone. To overcome variability in production networks, traditionally experiments are conducted by emulating the properties of the network. But given the significant variability in LTE networks, generating realistic models of latency and throughput for emulation is challenging and such models do not exist currently. Hence we chose to conduct all our experiments in live network settings. We conducted experiments at night, when the cell load is low, to eliminate variability due to other users. Next, we conducted experiments in rounds, where each round consists of back-to-back runs using different schemes for a given web page. This minimizes the effects of changing network conditions for different schemes. Finally, we logged the signal strength and cellular technology for each experiment, and only considered those rounds where all schemes experienced comparable signal strengths. We also filtered rounds where the device used 3G or handed off from LTE to 3G.

To obtain statistically significant results, we needed to collect statistics from tens of experiment rounds for a given web page. However, due to radio signal fluctuation and cellular technology hand-off, we had to filter out almost 50% of rounds. To make our experimental process tractable, we focused on a smaller subset of 34 web pages from the top 500 Alexa global pages (31 US sites), covering a wide range of

[2]An alternate option of considering $\Delta(n) = E(n+1) - E(n)$ provides a similar intuition with more complexity, which we do not show here. Note also that $E(n)$ omits some delay aspects that are constant across all n (e.g., propagation delay), which does not affect the analysis result.

[3]We anonymize the browser to keep the focus on the scientific aspects of our study.

categories like news, sports, photo streaming, business and science. The page sizes ranged from few KB to 5 MB. The median page size is 1.04 MB. To put the time in perspective, it took us multiple weeks to collect more than 20 post-filtered runs for each of the 34 web pages and the different schemes for Figure 9b.

We used the full versions of the web pages in our experiments. These are the web pages downloaded when data cards or tablets are used. Previous work has shown that mobile versions of the pages are optimized for smaller screen sizes; however, they are not significantly different from the full versions from a network communication perspective [25]. Hence, using a mobile version does not solve the problem of increased page load latency in cellular networks and we expect *PARCEL* to provide benefits even with mobile versions of the pages. The relative gains with mobile versions, however, may be different than the full versions.

7.3 Replaying pages to control page variability

Over the course of our experiments, we observed significant variability in the number of objects and aggregate download size for a given web page, even over short intervals of time. When we conducted 10 runs back-to-back during the month of February 2014, we found that 50% of the web-pages in our set had at least 0.5 as the coefficient of variation of the total number of objects. A similarly high variability was observed when considering page sizes as well.

To ensure fair comparisons in the presence of variability in web pages, a bulk of our experiments were conducted using an open source tool called web-page-replay [23] to record and replay web pages. Specifically, we first accessed real web pages from the actual web server(s) and stored the objects in a local server. Then, the same snapshot of a given page was replayed for all schemes in later experiments. Some web pages still showed variability as they had *JS* that requested different URLs (e.g., using a random number or date) over different runs. We modified the web-page-replay code to replace such occurrences with constant values to ensure the same objects were requested for all schemes.

Since both the proxy and the replay server are at the same location, we used dummynet [13] to emulate proxy to server delay. While we used 20ms as the default delay value for majority of our evaluation in the paper, we did conduct experiments to study the sensitivity to different delay values between the proxy and the server (last paragraph in § 8.3). We used the same proxy to server delay for all objects because we wanted to minimize the effect of page and network variability between the proxy and the server and for controllability of the setup. Further, web-page-replay [23] as a tool does not support capturing different latencies to different web servers for objects in the same page, though a recent tool [22] may allow emulation of heterogeneous delays in the future. For realism, we also compared the different schemes using real web servers (without using web-page-replay or dummynet), which captures heterogeneous delays from the proxy to the server. Note that while the mobile device and the proxy are in the same location, the path between the two goes through a production LTE network and is hence representative of a typical data session going through a LTE packet core and the Internet.

To eliminate variability across runs due to cached objects, all comparison experiments were conducted by flushing device cache across runs for both DIR and *PARCEL*. We also disabled proxy caching for *PARCEL* proxy. Moreover, we ignore the first run in each round and obtain the median for the remaining runs to ensure that DNS resolution does not impact our experiment results. Note that our experiments in § 8.2 do consider local device caching only because the focus of these experiments is on client interactions during a user session.

A majority of our experiments focussed on landing pages, though we conducted experiments involving a user session in § 8.2. In general, since a session consists of a sequence of webpage downloads, we expect *PARCEL's* benefits to aggregate over each page download. Even though some objects in subsequent pages of a session could potentially be cached in the device, an approach like *PARCEL* is still important since these pages too are likely to contain content that must be explicitly fetched.

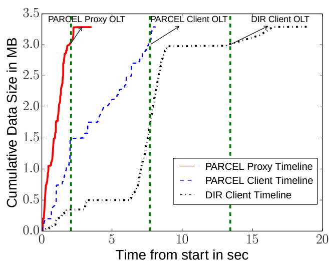
8 Results

In this section, we present results comparing *PARCEL* with DIR (§ 8.1), and CB (§ 8.2), as well as compare variants of *PARCEL* (§ 8.3). All these experiments are conducted by replaying pages using web-page-replay and the experimental approach described in § 7. We also present results with real web servers in § 8.4. Since CB does proprietary data transformations not implemented in *PARCEL*, we limit our comparisons with CB to § 8.2 where we wish to highlight the benefits of local JS execution.

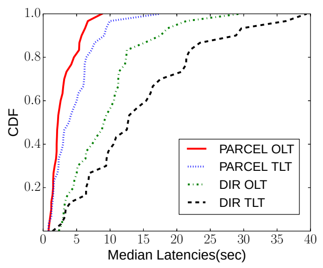
8.1 *PARCEL* vs. traditional mobile browsers

• ***Latency Reductions:*** Figure 6a shows the timeline for an example web page (*taobao.com*) highlighting the cumulative data downloaded at *PARCEL* (proxy and client), and the DIR client. From the figure, we observe that download occurs fastest at the *PARCEL* proxy, followed by the *PARCEL* client and DIR client respectively. This is because DIR incurs the long RTT of the cellular network for each object it requests, while *PARCEL* client sends a single request to the proxy. The DIR curve has a few long flat segments (no objects are downloaded – these are likely because the client was processing html and *JS* to figure out what objects to fetch next) and the gap between consecutive object downloads is long. In contrast, *PARCEL* proxy leverages the better processing power, high bandwidth and shorter RTT to the origin servers, to download the whole page quickly and to transfer the objects to the client faster. Observe from Figure 6a that the time to download all objects until onload event is 6 sec quicker at the *PARCEL* client (OLT is 7.5 sec) when compared to DIR (OLT is 13.44 sec). This result validates the intuition behind a key design element in *PARCEL*: it is desirable to reduce the number of requests from the client, and empower the *PARCEL* proxy early on in the download process to identify and download objects.

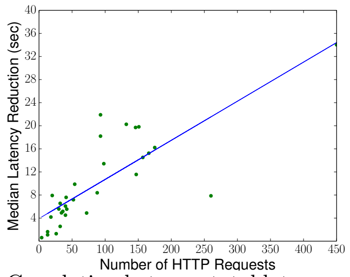
Figure 6b presents the CDF of the per-page median latencies (onload and total) across all web pages for the *PARCEL* client and DIR, observed over 40 runs with each scheme. From the figure, we see that *PARCEL* outperforms DIR consistently by a large margin. For 70% of the pages, the *PARCEL* OLT is smaller than 3 sec and the highest observed OLT is 8.8 sec. In contrast, only 10% of the pages have OLT smaller than 3 sec with DIR. Also, the 70th percentile and maximum latencies– 11 sec and 29 sec respectively are $3X$ higher than the corresponding *PARCEL* latencies. We also find that *PARCEL* reduces the OLT by > 1 sec for 90% of



(a) page download timeline with *PARCEL* and DIR for *taobao.com*



(b) Latencies with *PARCEL* and DIR for all pages.



(c) Correlation between total latency reduction and number of HTTP requests.

Figure 6: Effectiveness of *PARCEL* in reducing latency compared to DIR.

the pages, $>$ 5 sec for 60% of the pages, and reduces the total latency by $>$ 5 sec for 80% of the pages.

Figure 6c shows a scatter plot highlighting the correlation between the median total time reduction for *PARCEL* and the number of client HTTP requests (or number of objects) for each page. From the figure, we see that latency benefits with *PARCEL* increases with the increase in number of HTTP requests (correlation coefficient is 0.83). This shows that, as pages get richer and request hundreds of objects, benefits with *PARCEL* will be critical for maintaining a good user experience.

• ***Radio Energy Savings*** Figure 7a shows the RRC states over time for a single download (median run) of an example web page (landing page of *ebay.com*). The black shade represents CR, the white shades represents DRX and the bold red line shows the onload. From the figure, we see that with DIR spends a longer time in CR (active) and DRX (low power tail) states, whereas *PARCEL* reduces time spent in both CR and DRX states, saving energy in both the states. For example, the total radio energy consumed by *PARCEL* for this page is $5.63J$, which is half the energy consumed by DIR ($11.16J$). *PARCEL* benefits from the use of a single TCP connection to the proxy to download the entire page in a single transfer. In contrast, DIR uses multiple parallel TCP connections (6 per domain) to download individual objects. *PARCEL* delegates the effort to identify and fetch objects to the proxy which batches object transfers through the single TCP connection, thereby achieving higher throughput and hence reducing the time spent in CR state. Further, the implicit batching observed by the *PARCEL* proxy helps reduce the gap between consecutive objects transferred and thus reduces the time spent in DRX state. For the example page shown in Figure 7a, DIR observes 22 transitions between the CR and DRX, while *PARCEL* observes only 7 transitions between the states for the entire download.

Figure 7b shows the CDF of the per-page median radio energy consumption with *PARCEL* and DIR for all pages. From the figure, we see that *PARCEL* consistently consumes much lower energy when compared to DIR. With *PARCEL*, the energy usage was $< 4J$ for 80% of the pages and maximum of $8J$. In contrast, the DIR energy usage was $< 4J$ for only 38% of the pages with a maximum of $13J$.

Figure 7c shows for each page, (i) the median total radio energy savings with *PARCEL* for all the pages and (ii) the corresponding median CR energy savings, both expressed as a fraction of the total energy consumed with DIR. From the figure, we observe that *PARCEL* achieves significant energy savings compared to DIR across all the pages. *PARCEL* saves at least 20% of radio energy compared to DIR for 95% of the pages and the savings is at least 50% for 50% of the pages. Further, we note that the savings in the energy consumed in the CR state accounts for at least 50% of the total energy savings for 85% of the pages (the remaining savings are attributed to savings in DRX energy).

Overall our results show that *PARCEL* reduces both the incurred latency and the radio energy consumption when compared to traditional browsers by minimizing the HTTP request-response interactions and delegating the download process to the proxy.

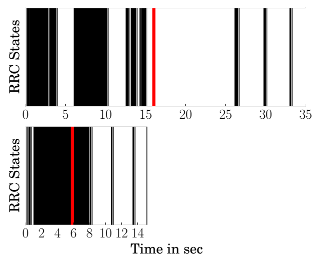
8.2 Importance of local *JS* execution

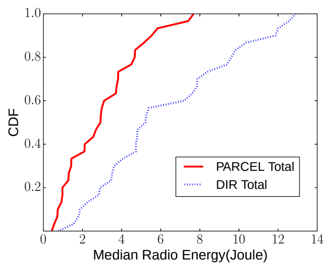
A key design decision with *PARCEL* was to execute *JS* at the client to ensure responsive and energy efficient client interactions. In this section, we illustrate the importance of this decision by conducting experiments involving the landing page of *ebay.com* and emulating an interactive session where the user clicks on a button once every minute to scroll through different product images. We compare *PARCEL* with DIR as well as **CB,** a popular commercially available cloud-based mobile browser, which executes all *JS* in the cloud and not at the client [32].[4]

Figure 8 top graph shows the cumulative radio energy consumption with *PARCEL*, DIR and CB over the entire user session. Each group of bars corresponds to a particular event in the session. The left-most group corresponds to the first download (FD), while the other groups correspond to subsequent user click events (C1-C4). Each bar shows the cumulative radio (total) energy consumed up to that event for a given scheme. The figure shows that the cumulative radio energy consumption grows significantly with each click event for CB, but remains unchanged with user interaction for *PARCEL* and DIR. This is because both DIR and *PARCEL* request the product images during the first-time download of the page, and store them locally. On a click event, *JS* runs locally on the client and displays the cached images without any network interaction. In contrast, CB issues a request to the proxy on each click event which processes the *JS* and sends back a new snapshot of the page to the client. This network communication required by CB results in radio energy consumption on each click event.

While CB incurs higher radio energy usage, it can potentially lower CPU energy usage compared to DIR and *PARCEL* by not executing *JS* in the client. Figure 8 bottom

[4]We anonymize the browser to keep the focus on the scientific aspects of our study.


(a) Time in RRC states with DIR(top) and *PARCEL* (bottom) for *ebay.com*.


(b) Total radio energy of *PARCEL* and DIR for all pages.

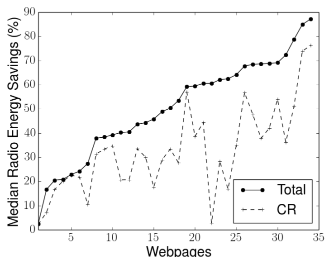

(c) Radio energy savings with *PARCEL* as a fraction compared to DIR.

Figure 7: Effectiveness of *PARCEL* in reducing total radio energy compared to DIR.

graph provides a more complete picture by showing the total device energy consumption cumulative to each session event. The total energy consumption was measured using a power meter. The screen was kept in the lowest brightness throughout the experiment. The baseline screen power (626mW) was measured and deducted from the total energy consumption. The total energy consumed by CB is indeed lower after the first download (FD), which can be attributed to running *JS* in the cloud rather than the device. However, the total energy consumption for CB increases rapidly on subsequent user clicks, and is higher than not only *PARCEL* but also DIR by the end of the session. In contrast, the cumulative total energy consumption with *PARCEL* is lower than DIR at every point in the session.

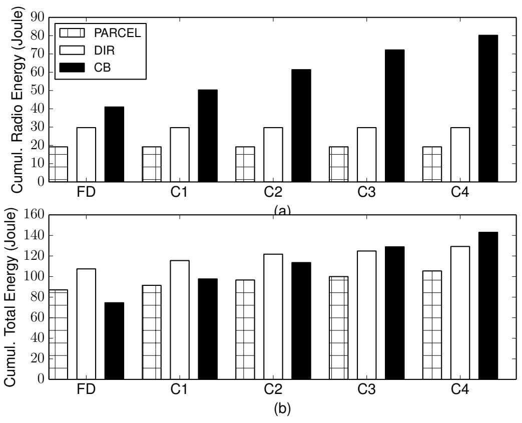


Figure 8: Cumulative radio and total device energy consumption for various schemes over a user session.

8.3 Exploring design knobs within *PARCEL*

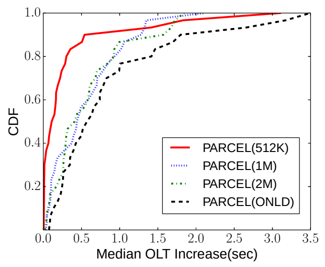
One of the design considerations with *PARCEL* is to schedule data transmission to the client by judiciously balancing user latencies and radio energy consumption (See 4.2). In this section, we experiment with different bundle sizes (512KB, 1MB, 2MB) for *PARCEL* (X) and present how they perform in terms of latency and radio energy. We use the same setup as before (§ 7) and results from 20 runs for each page and each scheme.

In Figure 9a, we present the OLT increase for different *PARCEL* bundling variants, compared to *PARCEL* (IND). As expected, all the bundling variants experience higher latency than *PARCEL* (IND), and the amount of increase in general becomes larger with larger bundles, which is consistent with our intuition and analysis in § 6. Specifically, as shown in Figures 9a, *PARCEL* (ONLD) has the highest latency (e.g., the increase is 0.57 sec). For *PARCEL* (512K), the increase is much smaller than other variants using larger bundles (e.g., the increase is 0.11 sec). We also observed similar trends in the TLT. However, we note that all *PARCEL* variants perform better than DIR.

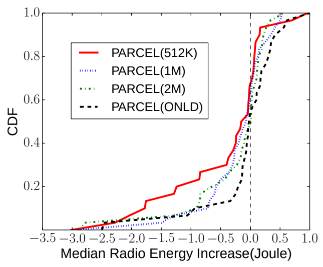
In Figure 9b, we present the total radio energy increase for *PARCEL* bundling variants compared to *PARCEL* (IND). We observe that the trend is not as clear as the latency case and there seems to be no one single bundle size that can minimize the radio energy for all web pages. Specifically, for around 60% of web pages, *PARCEL* (512K) results in less radio energy consumption than *PARCEL* (IND), while the opposite is true for the rest of the web pages. All *PARCEL* variants consume less radio energy than DIR.

We further analyze the results to understand when bundling helps in reducing the radio energy. In Figure 9c, we show a scatter plot of total page size and radio energy difference between *PARCEL* (512K) and *PARCEL* (IND). With larger web pages (e.g., $>$ 2MB), bundling decreases the radio energy consumption, compared to *PARCEL* (IND). Although not shown here due to space constraint, we observe the similar trend when using other bundle sizes. This energy saving is because bundling reduces the gaps between consecutive objects and thus incurs lower transition overhead. Also, transferring larger bundle potentially helps TCP overcome the slow-start effects and achieve increased effective throughput. With smaller pages (<2MB), the potential energy saving opportunity is inherently small, and the LTE network variability makes it difficult to identify a clear trend.

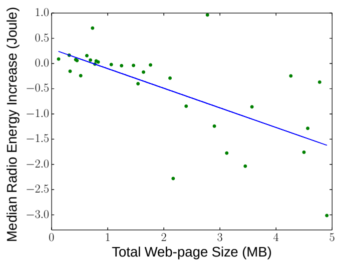
Sensitivity of bundle sizes on energy savings As expected, larger bundle sizes increase the client OLT compared to *PARCEL* (IND). The energy results however are more nuanced. For instance, smaller bundle size (512KB) seems to give more benefits than larger bundle sizes. The typical large web-pages range anywhere between 2 and 4MB and we observed download speeds ranging from 4 to 8 Mbps with median of 6 Mbps in our experiments. As shown in § 6, the optimal bundle size is around 1MB for those large web pages. However, we observe that energy saving in practice is largest with a bundle size slightly smaller than the optimal size from Eq. 1 (e.g., 512K instead of 1M). We believe that this is due in part to some of the assumptions we make (e.g., full state transition per bundle transfer, fractional n). Another reason is that in practice the observed download



(a) OLT increase



(b) Total radio energy increase



(c) Correlation between size and radio energy increase with *PARCEL* (512K)

Figure 9: Performance of *PARCEL* bundling variants, compared to *PARCEL* (IND).

speed is variable and hence larger bundles are more severely affected by sudden radio signal degradation.

Sensitivity to proxy-server delay We briefly report the results when we use dummynet to vary the delay between the *PARCEL* proxy and our web-page-replay server. In our experiments, we used two round-trip delay values: 20 and 60 ms, representing the RTTs from the client location (Purdue) to the east and west coasts. We observe that with a higher delay, *PARCEL* (ONLD) had higher latency penalty, but provided more energy savings compared to *PARCEL* (IND). Note that in general *PARCEL* (IND) will perform better if objects reach the client close to each other and cause lower state transition overhead. As the delay between proxy and server increases, the object arrivals at the proxy and with IND, at the client are likely to be more spread out, which causes higher energy consumption due to higher state transition overhead.

8.4 Experiments with real web servers

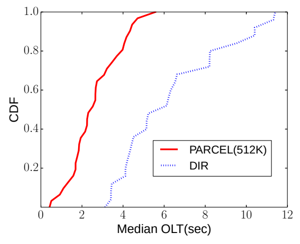


Figure 10: OLT with real web servers

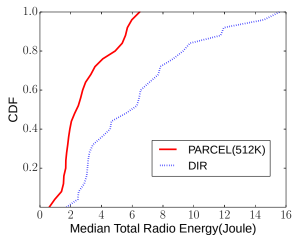


Figure 11: Radio energy with real web servers

The evaluation so far has focused on understanding the benefits of *PARCEL* in a controlled setting, where the pages are hosted on a server in Purdue. In this section, we conduct experiments with pages served from the original domain servers to study the benefits of *PARCEL* in the real world. We use results from 10 runs for each page with each scheme. In Figure 10, we observe that *PARCEL* (512K) consistently outperforms DIR in both onload latency and radio energy usage, which is consistent with our controlled experiment results. For example, the median OLT of *PARCEL* (512K) is $<$ 2.5sec, while that of DIR is around 6sec. We also observe that for 50% of the pages, the *PARCEL* (512K) OLT was one third or less of the DIR OLT. In Figure 11, we observe that *PARCEL* (512K) also reduces the total radio energy significantly compared to DIR. Specifically, with *PARCEL* (512K), the radio energy consumption of all pages is $< 6.5J$, while for around 40% of pages, DIR consumes significantly more radio energy. While we do not show the results, we also conducted experiments with *PARCEL* (IND). *PARCEL* (IND) had slightly lower OLT (0.1sec) and slightly higher radio energy ($0.5J$) than *PARCEL* (512K), but performing significantly better than DIR in both the metrics. The above results confirm the benefits of *PARCEL* in real world settings as well.

9 Conclusions

In this paper, we have shown that judiciously refactoring functionality between mobile browsers and proxies can significantly reduce page load times and radio energy usage in cellular networks. We have presented *PARCEL*, an initial attempt at such functionality refactoring. Distinct from traditional browsers, *PARCEL* moves the task of identifying and downloading objects needed to render a web-page to a well-provisioned proxy. Distinct from existing cloud-heavy browsers, *PARCEL* retains most other functionality including execution of *JS* in the client browser. Our evaluations confirm the potential of *PARCEL*. Compared to a traditional mobile web-browser, *PARCEL* reduces OLT by 49.6% and radio energy consumption by 65%. Unlike cloud-heavy browsers, *PARCEL* continues to perform well on client interactions. While our results are promising, there are many areas for future research as discussed below:

•*Combining PARCEL with page optimizations:* There are many recent efforts at webpage optimization services [7] to be used by web developers to optimize their sites. *PARCEL* can co-exist well with such optimizations, since the *PARCEL* proxy only processes the page to identify the objects and sends the objects received from the server as such to the client without modifying the HTML. Evaluating *PARCEL* in conjunction with such techniques is an interesting direction for future work.

•*Comparisons with other proxy-assisted approaches:* In this paper, we have demonstrated the benefits of *PARCEL* over the CB proxy (§ 8.2) and discussed its benefits over SPDY proxies (§ 4). A quantitative comparison with SPDY proxies is an interesting direction for future research, though we note that previous work [16] has already shown SPDY protocol alone does not outperform HTTP in the cellular setting. A fair comparison is complicated since the most noteable SPDY proxy [5] not only uses SPDY [17] to communicate with the clients, but also performs webpage optimization and data reduction [7], a feature not implemented in our *PARCEL* prototype. Integrating *PARCEL* with such page optimization techniques as described above will facilitate a fair head-to-head comparison in the future.

•*Adapting PARCEL for traffic encryption:* Today 67.2% of the top 151,509 Alexa pages still use HTTP [10] and only 7.25% of the mobile traffic in NorthAmerica is encrypted [30]. However, recent reports do suggest that the aggregated encrypted traffic is increasing throughout the globe [30]. Dealing with HTTPS traffic is an issue not only for *PARCEL* but also for many proxy-assisted solutions([5,6]) that involve web-page parsing. While our current *PARCEL* implementation lets HTTPS traffic follow the normal path without using our proxy, one approach to handling HTTPS traffic is using a trusted, per-client proxy, as we have discussed in the design section (§ 4.5). In the future, we hope to investigate potential issues around personalizing *PARCEL* proxies, and tackle questions around how they must be deployed, placed, and managed. Understanding the cost of running personalized proxies with large number of users is an interesting direction for future research.

10 Acknowledgments

We thank our shepherd David Choffnes and the anonymous reviewers for their constructive feedback and comments. We also thank Oliver Spatscheck for his valuable inputs during discussions. This work was supported in part by the National Science Foundation (NSF) under Career Award No. 0953622 and NSF Award No. 1162333. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

11 References

[1] The amazon kindle fire: Benchmarked, tested, and reviewed. http://www.tomshardware.com/reviews/amazon-kindle-fire-review,3076-7.html.

[2] Amazon silk split browser architecture. https://s3.amazonaws.com/awsdocs/AmazonSilk/latest/silk-dg.pdf.

[3] Amazon's silk browser acceleration tested: Less bandwidth consumed, but slower performance. http://tinyurl.com/84br5tc.

[4] Cisco visual networking index: Global mobile data traffic forecast update, 2013-2018. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html.

[5] Data compression proxy in android chrome. https://developer.chrome.com/multidevice/data-compression.

[6] Opera mini architecture and javascript. http://dev.opera.com/articles/view/opera-mini-and-javascript/.

[7] Pagespeed optimization service. https://developers.google.com/speed/pagespeed/.

[8] Skyfire - cloud based mobile optimization browser. http://www.skyfire.com/operator-solutions/whitepapers.

[9] Squid web proxy. Available at http://www.squid-cache.org.

[10] SSL Pulse. https://www.trustworthyinternet.org/ssl-pulse/.

[11] 3GPP TS 36.331: Radio Resource Control (RRC) (v10.3.0). 2011.

[12] R. Cáceres, L. Cox, H. Lim, A. Shakimov, and A. Varshavsky. Virtual Individual Servers as Privacy-Preserving Proxies for Mobile Devices. In *Proc. ACM MobiHeld*, 2009.

[13] M. Carbone and L. Rizzo. Dummynet Revisited. In *ACM SIGCOMM Computer Communication Review*, March 2010.

[14] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. CloneCloud: Elastic Execution between Mobile Device and Cloud. In *Proc. ACM Eurosys*, 2011.

[15] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI:Making Smartphones Last Longer with Code Offload. In *Proc. ACM MobiSys*, 2010.

[16] J. Erman, V. Gopalakrishnan, R. Jana, and K.K. Ramakrishnan. Towards a SPDY'ier mobile web? In *Proc. CoNEXT*, Dec 2013.

[17] Google. SPDY: An experimental protocol for a faster web. http://www.chromium.org/spdy/spdy-whitepaper.

[18] 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 *Proc. ACM Mobisys*, 2012.

[19] R. Kemp, N. Palmer, T. Kielmann, and H. Bal. Cuckoo: a Computation Offloading Framework for Smartphones. In *Proc. MobiCASE*, 2010.

[20] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang. ThinkAir: Dynamic resource allocation and parallel execution in cloud for mobile code offloading. In *Proc. IEEE INFOCOM*, 2012.

[21] K. Matsudaira. Making the mobile web faster. *Communications of the ACM, Vol 56. No 3.*, 2013.

[22] R. Netravali, A. Sivaraman, K. Winstein, S. Das, A. Goyal, and H. Balakrishnan. Mahimahi: A Lightweight Toolkit for Reproducible Web Measurement (Demo). In *ACM SIGCOMM 2014*, Chicago, IL, August 2014.

[23] Web-page-replay. Record and play back web pages with simulated network conditions. https://www.code.google.com/p/web-page-replay/.

[24] 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 *Proc. ACM MobiSys*, 2012.

[25] F. Qian, S. Sen, and O. Spatscheck. Characterizing Resource Usage for Mobile Web Browsing. In *Proc. ACM MobiSys*, 2014.

[26] 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 *Proc. ACM Mobisys*, 2011.

[27] S. Rajaraman, M. Siekkinen, V. Virkki, and J. Torsner. Bundling Frames to Save Energy While Streaming Video from LTE Mobile Device. In *Proc. ACM MobiArch*, 2013.

[28] A. Saarinen, M. Siekkinen, Y. Xiao, J. K. Nurminen, and M. Kemppainen. Smartdiet: Offloading Popular Apps to Save Energy(Poster). In *Proc. ACM Sigcomm*, 2012.

[29] A. Saarinen, M. Siekkinen, Y. Xiao, J. K. Nurminen, M. Kemppainen, and P. Hui. Can Offloading Save Energy for Popular Apps. In *Proc. ACM MobiArch*, 2012.

[30] Sandvine. Global internet phenomena report 1h-2014. Available at https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/1h-2014-global-internet-phenomena-report.pdf.

[31] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies. The Case for VM-based Cloudlets in Mobile Computing. *IEEE/Trans. Pervasive Computing*, 2009.

[32] 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 *Proceedings of ACM HotMobile*, 2014.

[33] S. Souders. Onload event and post-onload requests. http://www.stevesouders.com/blog/2012/10/30/qa-nav-timing-and-post-onload-requests.

[34] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. How Speedy is SPDY? In *Proc. NSDI*, April 2014.

[35] X. S. Wang, H. Shen, and D. Wetherall. Accelerating the Mobile Web with Selective Offloading. In *Proc. ACM MCC*, 2013.

[36] B. Zhao, B. C. Tak, and G. Cao. Reducing the Delay and Power Consumption of Web Browsing on Smartphones in 3G Networks. In *Proc. ICDCS*, 2011.

[37] Z. Zhu, P. Gupta, Q. Wang, S. Kalyanaraman, Y. Lin, H. Franke, and S. Sarangi. Virtual base station pool: towards a wireless network cloud for radio access networks. In *Proc. of the 8th ACM International Conference on Computing Frontiers*, 2011.