Towards a SPDY'ier Mobile Web?

Jeffrey Erman, Vijay Gopalakrishnan, *Member, IEEE*, Rittwik Jana, *Member, IEEE*, and Kadangode K. Ramakrishnan, *Fellow, IEEE*

***Abstract*—Despite its widespread adoption and popularity, the Hypertext Transfer Protocol (HTTP) suffers from fundamental performance limitations. SPDY, a recently proposed alternative to HTTP, tries to address many of the limitations of HTTP (e.g., multiple connections, setup latency). In this paper, we perform a detailed measurement study to understand the benefits of using SPDY over cellular networks. Through careful measurements conducted over 4 months, we provide a detailed analysis of the performance of HTTP and SPDY, how they interact with the various layers, and their implications on Web design. Our results show that unlike in wired and 802.11 networks, SPDY does not clearly outperform HTTP over cellular networks. We identify negative interactions between the protocols used for Web access (HTTP/SPDY over TCP) and cellular radio resource management as the underlying cause. Overall performance suffers when devices go through a cellular radio state promotion after an idle period, and the consequent increase in latency. This impacts SPDY more because of the use of a single TCP connection. We conclude that a viable solution has to account for these unique cross-layer dependencies to achieve improved Web performance over cellular networks.**

***Index Terms*—Cellular networks, mobile Web performance, SPDY, wireless protocol.**

I. Introduction

As THE speed and availability of cellular networks grow, they are rapidly becoming the access network of choice. Web access remains one of the most important uses of the mobile Internet. It is therefore critical that the performance of the cellular network be tuned optimally for mobile Web access.

The Hypertext Transfer Protocol (HTTP) is the key building block of the Web. Its simplicity and widespread support has catapulted it into being adopted as the nearly "universal" application protocol, such that it is being considered the narrow waist of the future Internet [1]. Yet, despite its success, HTTP suffers from fundamental limitations, many of which arise from the use of TCP as its transport layer protocol. It is well established that TCP works best if a session is long-lived and/or exchanges a lot of data. This is because TCP gradually ramps up load and takes time to adjust to the available network capacity. Since HTTP connections are typically short and exchange small objects, TCP does not have sufficient time to utilize the full network capacity. This is particularly exacerbated in cellular networks where high latencies (hundreds of milliseconds are not unheard of [2]) and packet loss in the radio access network is common. These factors are widely known to impair TCP performance.

SPDY [3], a recently proposed protocol, aims to address many of the inefficiencies with HTTP. SPDY uses fewer TCP connections by opening one connection per domain. Multiple data streams are multiplexed over this single TCP connection for efficiency. SPDY supports multiple outstanding requests from the client over a single connection. SPDY servers transfer higher-priority resources faster than low-priority resources. Finally, by using header compression, SPDY reduces the amount of redundant header information each time a new page is requested. Experiments show that SPDY reduces page load time by as much as 64% on wired networks and estimate as much as 23% improvement on cellular networks [4] (based on an emulation using Dummynet).

In this paper, we perform a detailed and systematic measurement study on production cellular networks to understand the real-world benefits of SPDY. We deployed a SPDY proxy that functions as an intermediary between the mobile devices and Web servers and ran detailed field measurements using 20 popular Web pages. These were performed across a 4-month span to account for the variability in the cellular network. Each of the measurements was instrumented and set up to account for, and minimize, factors that could bias the results (e.g., cellular handoffs). We use a proxy for several reasons: First, cellular operators, especially in the US, make use of proxies for Web traffic [5], [6]. This has been motivated by their desire to adapt and optimize content to be delivered to mobile devices. With a similar motivation, content providers are also deploying SPDY-based proxies [7], [8]. While it is yet to be seen which of these (or both) will be the ultimate long-term solution, we believe the use of a proxy in our measurements reflects the current technology direction. Second, SPDY is most effective when there are multiple requests over the single TCP connection, especially over the high-latency cellular link. Without a proxy, the client would have to establish new TCP connections to multiple servers (due to domain sharding) and thereby limit the benefits of SPDY. Finally, since only about 0.9% Web sites support SPDY [9], it was hard to get detailed insights without a proxy.

The main observation from our experiments is that, unlike in wired and 802.11 WiFi networks, SPDY *does not* outperform HTTP. Most importantly, we see that the interaction between TCP and the cellular network has the most impact on performance. Specifically, TCP implementations do not account for the long promotion delays encountered when a cellular radio's state transitions from idle to active after an idle period. As a

Manuscript received November 25, 2014; revised April 26, 2015 and June 24, 2015; accepted June 27, 2015; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Fahmy. Date of publication October 26, 2015; date of current version December 15, 2015.

J. Erman, V. Gopalakrishnan, and R. Jana are with AT&T Labs—Research, Bedminster, NJ 07921 USA (e-mail: erman@research.att.com; gvijay@research.att.com; rjana@research.att.com).

K. K. Ramakrishnan is with the University of California, Riverside, Riverside, CA 92521 USA (e-mail: kk@cs.ucr.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2015.2462737

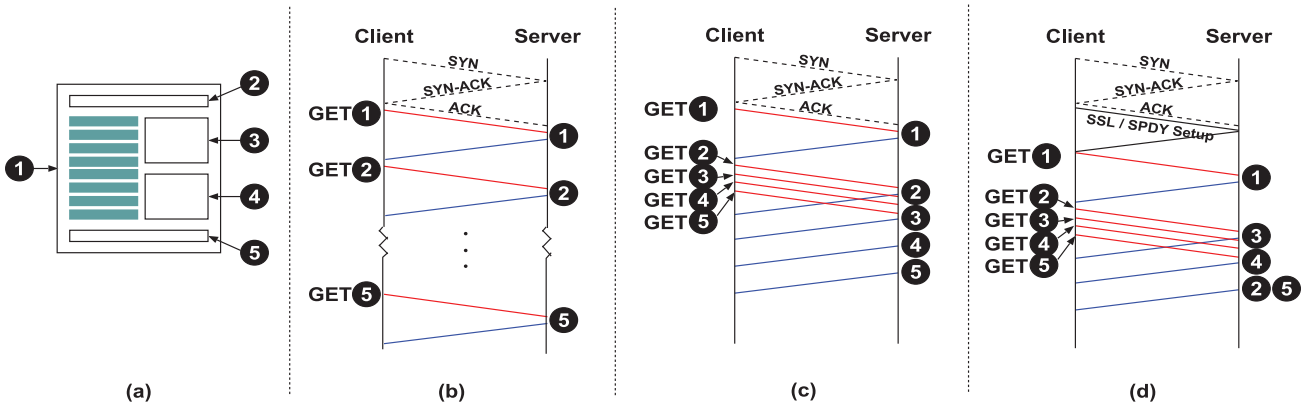

Fig. 1. This example shows how the different objects on a Web page are downloaded when using HTTP (with just persistent connections and with both persistent connections and pipelining) and SPDY. (a) Example Web page. (b) HTTP persistent connection. (c) HTTP with Pipelining. (d) SPDY.

result, the TCP round-trip time (RTT) estimate and thus the timeout value is incorrect (significantly underestimated) after an idle period, triggering spurious retransmissions and hence lower throughput. The TCP connection and the cellular radio connection for the end-device become idle because of users' Web browsing patterns (with a "think time" between pages [10]) and how Web sites exchange data. Since SPDY uses a single long-lived TCP connection in the presence of a proxy, timeouts impact page load times significantly. HTTP is less affected by this because of its use of parallel connections (isolates impact to a subset of active connections) and because the connections are short-lived (isolates impact going across Web sites). While we could similarly open separate SPDY connections for each domain, that would not allow reuse of TCP connections and hence diminishes the benefit of moving from HTTP to SPDY. Hence, we believe that a viable solution has to account for these unique cross-layer dependencies to achieve improved performance of both HTTP and SPDY over a cellular network.

The main contributions of this paper include the following.

- We conduct a systematic and detailed study over more than 4 months on the performance of HTTP and SPDY. We show that SPDY and HTTP perform similarly over cellular networks, and that using one TCP connection with SPDY makes it susceptible to negative interactions between TCP and the cellular network.
- We show that the interaction between the cellular network and TCP needs optimization. In particular, we show that cellular radio state promotion delays cause spurious TCP timeouts, causing severe performance degradation.
- We show that Web site design, where data is requested asynchronously and with gaps, can trigger cellular state changes resulting in TCP timeouts. We also show that there exist dependencies in Web pages today that prevent the browser from fully utilizing SPDY's capabilities.

II. Background

We present a brief background on how HTTP and SPDY protocols work in this section. We use the example in Fig. 1 to aid our description.

A. HTTP Protocol

The HTTP is a stateless, application-layer protocol for transmitting Web documents. It uses TCP as its underlying transport protocol. Fig. 1(a) shows an example Web page that consists of the main HTML page and four objects referred in that page. When requesting the document, a browser goes through the typical TCP 3-Way handshake as depicted in Fig. 1(b) and (c). Upon receiving the main document, the browser parses the document and identifies the next set of objects needed for displaying the page. In this example, there are four more objects that need to be downloaded.

With the original versions of HTTP, a single object was downloaded per connection. HTTP ver. 1.1 introduced the notion of persistent connections that have the ability to reuse established TCP connections for subsequent requests and the concept of pipelining. With persistence, objects are requested sequentially over a connection as shown in Fig. 1(b). Objects are not requested until the previous response has completed. However, this introduces the problem of *head-of-line* (HOL) blocking where subsequent requests get significantly delayed in waiting for the current response to come back. Browsers attempt to minimize the impact of HOL blocking by opening multiple concurrent connections to each domain (most browsers today use six parallel connections) with a limit on the number of active connections across all domains.

With pipelining, multiple HTTP requests can be sent to a server together without waiting for the corresponding responses as shown in Fig. 1(c). The client then waits for the responses to arrive in the order in which they were requested. Pipelining can improve page load times dramatically. However, since the server is required to send its responses in the same order that the requests were received, HOL blocking can still occur with pipelining. Some mobile browsers have only recently started supporting pipelining.

B. SPDY Protocol

Even though HTTP is widely adopted and used today, it suffers from several shortcomings (e.g., sequential requests, HOL blocking, short-lived connections, lack of server initiated data exchange, etc.) that impact Web performance, especially on the cellular network.

SPDY [3] is a recently proposed application-layer protocol for transporting content over the Web with the objective of minimizing latency. The protocol works by opening one TCP connection per domain (or just one connection if going via a proxy).

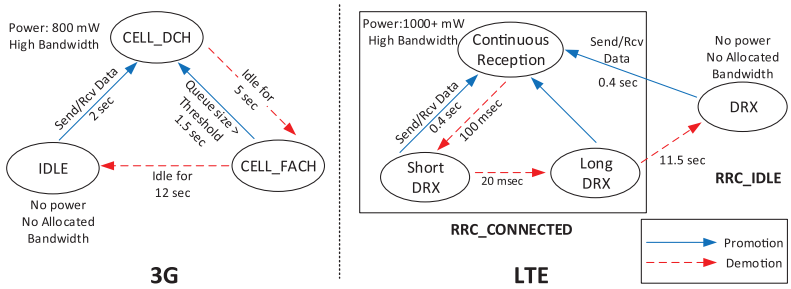

Fig. 2. RRC state machines for 3G UMTS and LTE networks.

SPDY then allows for unlimited concurrent streams over this single TCP connection. Because requests are interleaved on a single connection, the efficiency of TCP is much higher: Fewer network connections need to be made, and fewer, but more densely packed, packets are issued.

SPDY implements request priorities to get around one object request choking up the connection. This is described in Fig. 1(d). After downloading the main page, and identifying the objects on the page, the client requests all four objects in quick succession, but marks objects 3 and 4 to be of higher priority. As a result, server transfers these objects first thereby preventing the connection from being congested with noncritical resources (objects 2 and 5) when high-priority requests are pending. SPDY also allows for multiple responses to be transferred as part of the same packet [e.g., objects 2 and 5 in Fig. 1(d)] can fit in a single response packet can be served altogether. Finally, SPDY compresses request and response HTTP headers and Server-initiated data exchange. All of these optimizations have shown to yield up to 64% reduction in page load times with SPDY [3].

C. Cellular State Machines

The radio state of every device in a cellular network follows a well-defined state machine. This state machine, defined by 3GPP [11] and controlled by the radio network controller (in 3G) or the base station (in LTE), determines when a device can send or receive data. While the specifics differ between 3G and LTE, the main purpose is similar: The occupancy in these states controls the number of devices that can access the radio network at a given time. It enables the network to conserve and share available radio resources among the devices. It also allows for saving the device battery when the device does not have data to send or receive.

3G State Machine: The 3G state machine, as shown in Fig. 2, typically consists of three states: $IDLE$, Forward access channel $(FACH)$, and Dedicated channel (DCH). When the device has no data to send or receive, it stays in the $IDLE$ state. The device does not have radio resource allocated to it in $IDLE$. When it wants to send or receive data, it has to be *promoted* to the DCH mode, where the device is allocated dedicated transport channels in both the downlink and uplink directions. The delay for this promotion is typically $\sim$ 2 s. In the $FACH$, the device does not have a dedicated channel, but can transmit at a low rate. This is sufficient for applications with small amounts or intermittent data. A device can transition between DCH and $FACH$ based on data transmission activity.

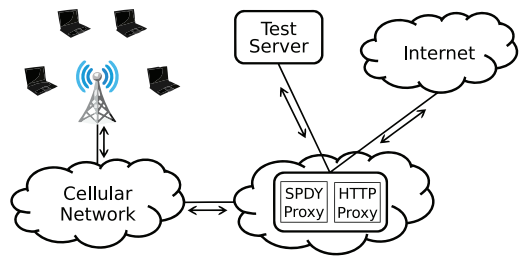

Fig. 3. Our test setup. We used laptops equipped with 3G/LTE dongles and hosted a SPDY and HTTP proxy on the same server in a public cloud.

For example, if a device is inactive for $\sim$ 5 s, it is *demoted* from DCH to $FACH$. It is further demoted to $IDLE$ if there is no data exchange for another $\sim$ 12 s. Note that state transition timer values vary across networks and implementations.

LTE State Machine: LTE employs a slightly modified state machine with two primary states: $IDLE$ and $CONNECTED$. If the device is in $IDLE$ and sends or receives a packet (regardless of size), a state promotion from $IDLE$ to $CONNECTED$ occurs in about 400 ms. LTE makes use of three substates within $CONNECTED$. Once promoted, the device enters Continuous Reception state where it uses considerable power (about 1000 mW) but can send and receive data at high bandwidth. If there is a period of inactivity (e.g., for 100 ms), the device enters the short Discontinuous Reception $(Short\_DRX)$ state . If data arrives, the radio returns to the Continuous Reception state in $\sim$ 400 ms. If not, the device enters the long Discontinuous Reception $(Long\_DRX)$ state. In the $Long\_DRX$ state, the device prepares to switch to the $IDLE$ state, but is still using high power and waiting for data. If data does arrive within $\sim$ 11.5 s, the radio returns to the Continuous Reception state; otherwise it switches to the low-power $(<$ 15 mW$)IDLE$ state. Thus, compared to 3G, LTE has significantly shorter promotion delays.

III. Experimental Setup

We conducted detailed experiments comparing the performance of HTTP and SPDY on the 3G network of a commercial US cellular provider over a 4-month period.

Fig. 3 provides an overview of our test setup. Clients in our setup connect over the cellular network using HTTP or SPDY to proxies that support that corresponding protocol. These proxies then use persistent HTTP to connect to the different Web servers and fetch requested objects. We run a SPDY and an HTTP proxy on the same machine for a fair comparison. We use a proxy

as an intermediary for several reasons: 1) We necessarily could not compare SPDY and HTTP directly. There are relatively few Web sites that support SPDY. When a server and client can negotiate SPDY, they always default to using SPDY and cannot be forced to use HTTP. Thus, when comparing HTTP and SPDY, we would be evaluating connections to different servers that could affect our results (depending on their load, number of objects served, etc). 2) Most cellular operators in the US already use HTTP proxies to improve Web performance. Running a SPDY proxy would allow operators to support SPDY over the cellular network even if the Web sites do not.[1] 3) Most importantly, without a proxy, the client would establish new TCP connections to SPDY servers in different domains and not reuse existing connections unless the request is to the same domain. This diminishes the intended benefit of SPDY and negates the need to move from HTTP to SPDY.

Test Devices: We use laptops running Windows 7 and equipped with 3G (UMTS) USB cards as our client devices. We ran experiments with multiple laptops simultaneously accessing the test Web sites to study the effect of multiple users loading the network. There are several reasons we use a laptop for our experiments. First, tablets and cellular-equipped laptops are on the rise. These devices request the regular Web pages unlike smartphones. Second, and more importantly, we wanted to eliminate the effects of a slow processor as that could affect our results. For example, studies [13] have shown that HTML, Javascript, and CSS processing and rendering can delay the request of required objects and significantly affect the overall page load time. Finally, it has been observed [14] that having a slow processor increases the number of zero window advertisements, which significantly affects throughput.

Test Client: We used a default installation of the Google Chrome browser (ver. 23.0) as the test client, as it supported traversing a SPDY proxy. Depending on the experiment, we *explicitly* configured Chrome to use either the HTTP or the SPDY proxy. When using an HTTP proxy, Chrome opens up to 6 parallel TCP connections to the proxy per domain, with a maximum of 32 active TCP connections across all domains. With SPDY, Chrome opens one SSL-encrypted TCP connection and reuses this connection to fetch Web objects. The connection is kept persistent and requests for different Web sites reuse the connection.

Test Location: Cellular experiments are sensitive to a lot of factors, such as signal strength, location of the device in a cell, the cell tower's backhaul capacity, load on the cell tower, etc. For example, a device at a cell edge may frequently get handed off between towers, thereby contributing to added delays. To mitigate such effects, we identified a cell tower that had sufficient backhaul capacity and had minimal interference from other cell sites. For most of our experiments, we chose a physical location with an unobstructed view of the tower and received a strong signal (between -47 and -52 dBm). We configured the 3G modem to remain connected to that base station at that sector on a particular channel frequency and used a diagnostic tool to monitor the channel on that sector.

[1]While the end-end encryption in SPDY makes the use of transparent proxies impossible, one could still deploy forward proxies, just as some content providers are doing [7], [8]. HTTP/2, however, does not mandate encryption [12].

TABLE I
CHARACTERISTICS OF THE DIFFERENT WEB SITES USED AS PART OF OUR EXPERIMENTS. SINCE THE CONTENT OF WEB SITES CHANGES OVER TIME, THE NUMBERS FOR EACH METRIC ARE AVERAGED ACROSS RUNS

Website	Total Objs	Avg. Size (KB)	Avg. No. of Domains	Avg. Text Objs	Avg. JS/ CSS	Avg. Imgs/ Other
Finance	134.8	626.9	37.6	28.6	41.3	64.9
Entertainment	160.6	2197.3	36.3	16.5	28.0	116.1
Shopping	143.8	1563.1	15.8	13.3	36.8	93.7
Portal	121.6	963.3	27.5	9.6	18.3	93.7
Technology	45.2	602.8	3.0	2.0	18.0	25.2
ISP	163.4	1594.5	13.2	13.2	36.4	113.8
News	115.8	1130.6	28.5	9.1	49.5	57.2
News	157.7	1184.5	27.3	29.6	28.3	99.8
Shopping	5.1	56.2	2.0	3.1	2.0	0.0
Auction	59.3	719.7	17.9	6.8	7.0	45.5
Online Radio	122.1	1489.1	17.9	24.1	21.0	77.0
Photo Sharing	29.4	688.0	4.0	2.3	10.0	17.1
Technology	63.4	895.1	9.0	4.1	15.0	44.3
Baseball	167.8	1130.5	12.5	19.5	94.0	54.3
News	323	1722.7	84.7	73.4	73.6	176.0
Football	267.1	2311.0	75.0	60.3	56.9	149.9
News	218.5	4691.3	37.0	19.0	56.3	143.2
Photo Sharing	33.6	1664.8	9.1	3.3	6.7	23.6
Online Radio	68.7	2908.9	15.5	5.2	23.8	39.7
Weather	163.2	1653.8	48.7	19.7	45.3	98.2

Proxies Used: We used a virtual machine running Linux in a compute cloud on the east coast of US to host our proxies. At the time of our experiments, there were no proxy implementations that supported both HTTP and SPDY. Hence, we chose implementations that are purported to be widely used and the most competent implementations for the corresponding protocols. We used Squid (ver. 3.1) [15] as our HTTP proxy. Squid supports persistent connections to both the client and the server. However, it only supports a rudimentary form of pipelining. For this reason, we did not run experiments of HTTP with pipelining turned on. Our comparisons are restricted to HTTP with multiple persistent connections. For SPDY, we used a SPDY server built by Google and made available as part of the Chromium source tree. This server was used in the comparison [3] of SPDY and HTTP and has since had extensions built in to support proxying. We ran `tcpdump` to capture network-level packet traces and `tcp-probe` kernel module to capture TCP congestion window values from the proxy to the mobile device.

Web Pages Requested: We identified the top Web sites visited by mobile users to run our tests (in the top Alexa sites). Of these, we eliminated Web sites that are primarily landing pages (e.g., login pages) and picked the remaining 20 most requested pages. These 20 Web pages have a good mix of news Web sites, online shopping and auction sites, photo and video sharing, as well as professionally developed Web sites of large corporations. We preferred the "full" site instead of the mobile versions, keeping in mind the increasing proliferation of tablets and large screen smartphones. These Web sites contain anywhere from 5 to 323 objects, including the home page. The objects in these sites were spread across 3–84 domains. Each Web site had HTML pages, Javascript objects, CSS and images. We list relevant details of these Web sites in Table I.

Test Execution: We used a custom client that talks to Chrome via the remote debugging interface and got Chrome to load the

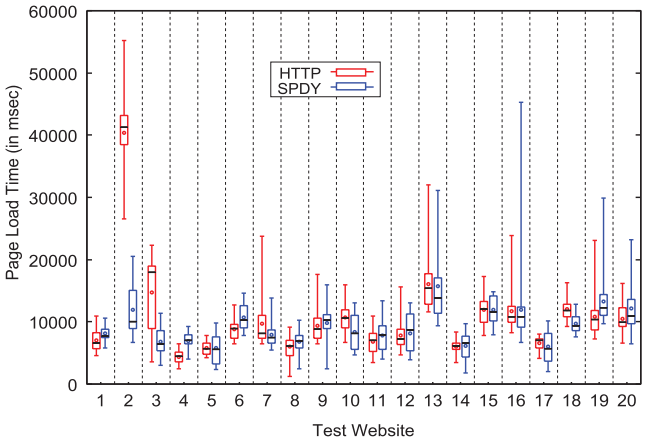

Fig. 4. Page load time for different Web sites with HTTP and SPDY over a 3G network. The figure shows there is very little performance difference between HTTP and SPDY when using 3G.

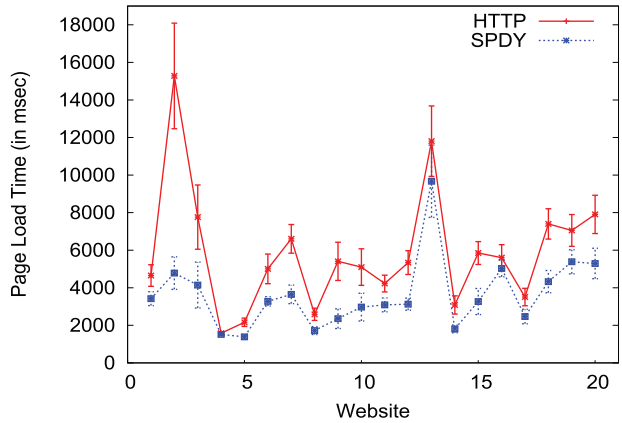

Fig. 5. Average page load time with HTTP and SPDYY over an 802.11g/Broadband network. The whiskers represent 95% confidence interval. The figure shows that SPDY significantly outperforms HTTP in this setting.

test Web pages. We generated a random order in which to visit the 20 Web sites and used that same order across all experiments. Each Web site was requested 60 s apart. The page may take much shorter time to load; in that case, the system would be idle until the 60-s window elapsed. We chose 60 s both to allow for Web pages to load completely and to reflect a nominal think time that users take between requests. We clear the browser cache before requesting each page.

We used page load time as the main metric to monitor performance. Page load time is defined as the time it takes the browser to download and process all the objects associated with a Web page. Most browsers fire a Javascript event (`onLoad()`) when the page is loaded. The remote debugging interface provided us the time to download the different objects in a Web page. We alternated our test runs between HTTP and SPDY to ensure that temporal factors do not affect our results. We ran each experiment multiple times during the typically quiet periods (e.g., 12 AM–6 AM) to mitigate effects of other users using the base station.

IV. Experimental Results

We first compare the performance of SPDY and HTTP using data collected from a week's worth of experiments. Since there was a lot of variability in the page load times, we use a box plot to present the results in Fig. 4. The x-axis shows the different Web sites we tested; the y-axis is the page load time in milliseconds. For each Web site, the (red) box on the left shows the page load times for HTTP, while the (blue) box on the right shows the times for SPDY. The box plot gives the standard metrics: the 25th percentile, the 75th percentile, and the black notch in the box is the median value. The top and bottom of the whiskers show the maximum and minimum values, respectively. Finally, the circle in these boxes shows the mean page load time across all the runs.

The results from Fig. 4, interestingly, do not show a convincing winner between HTTP and SPDY. For some sites, the page load time with SPDY is lower (e.g., 3, 7), while for others, HTTP performs better (e.g., 1, 4). However, for a large number of sites there is not a significant difference.[2] This is in sharp contrast to existing results on SPDY, where it has been shown to have between 27%–60% improvement [3]. Importantly, previous results have shown an average of 23% reduction over emulated cellular networks [4].

Performance Over 802.11 Wireless Networks: As a first step in explaining the result in Fig. 4, we wanted to ensure that the result was not an artifact of our test setup or the proxies used. Hence, we ran the same experiments using the same setup, but over an 802.11g wireless network connected to the Internet via a typical residential broadband connection (15 Mb/s down/2 Mb/s up).

Fig. 5 shows the average page load times and the 95% confidence intervals. Like previous results [3], this result also shows that SPDY performs better than HTTP consistently with page load time improvements ranging from 4% for Web site 4 to 56% for Web site 9 (ignoring Web site 2). Since the only difference between the two tests is the access network, we conclude that our results in Fig. 4 are a consequence of how the protocols operate over the cellular network.

V. Understanding the Cross-Layer Interactions

We look at the different components of the application and the protocols that can affect performance. In the process, we observe that there are significant interdependencies between the different layers (from browser behavior and Web page design, to TCP protocol implementations, to the intricacies of the cellular network) that affect overall performance.

A. Object Download Times

The first result we study is the breakdown of the page load time. Recall that, by default, the page load time is the time it takes the browser to process and download all the objects required for the Web page. Hence, we look into the average download time of objects on a given page. We split the download time of the object into four steps: 1) the *initialization* step that includes the time from when the browser realizes that it requires the object to when it actually requests the object; 2) the *send* step that includes the time to actually send the request over the network; 3) the *wait* time that is the time between sending the

[2]Using tracing tools, we found that the browser would occasionally stall when processing a Javascript file on this site. These stalls happened more often with HTTP, thereby increasing page load times. We saw this behavior with WiFi as well.

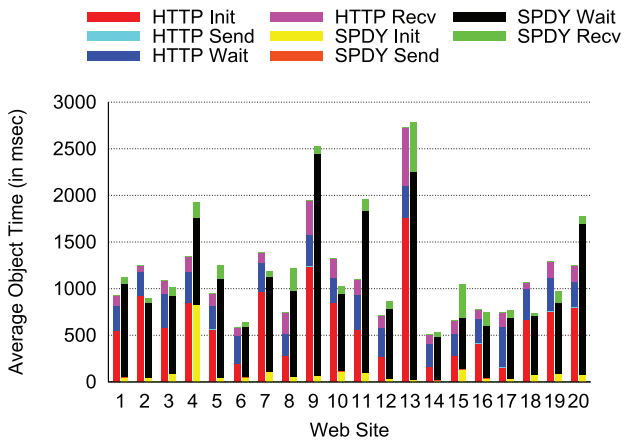

Fig. 6. Contribution of intermediate steps to average download times of objects over 3G. SPDY offers significant gains in terms of connection initiation overhead, but long wait times negate those gains.

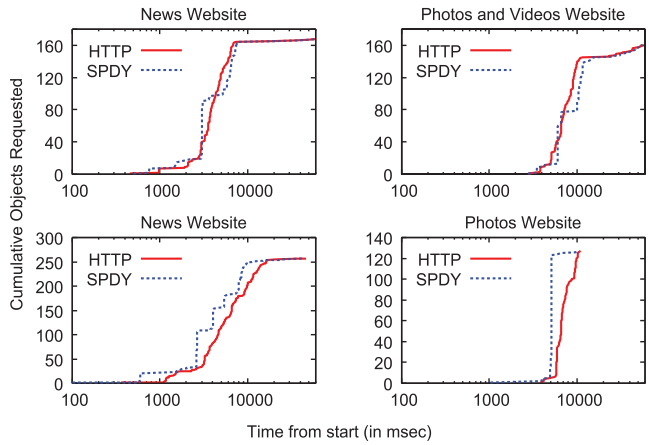

Fig. 7. Object request patterns for HTTP and SPDY for different Web sites. HTTP tends to request objects later than SPDY, but usually catches up due to object interdependencies slowing down SPDY.

request until the first byte of response; and finally 4) the *receive* time that is the time to receive the object.

We plot the average time for these steps for the different Web sites in Fig. 6. First, we see that the trends for average object download time are quite similar to that of page load times (in Fig. 4). This is not surprising since page load time is dependent on the object download times. Next, we see that the send time is negligible for both HTTP and SPDY, indicating that the request is sent very quickly. Almost all HTTP requests fit in one TCP packet. Similarly, almost all SPDY requests also fit in a single TCP packet, even when the browser bundles multiple SPDY requests in one packet. Third, we see that receive times with HTTP and SPDY are similar, with SPDY resulting in slightly better average receive times. We see that the initialization time is much higher with HTTP because the browser has to either open a new TCP connection to download the object (and adds the delay of a TCP handshake), or wait until it can reuse an existing connection.

SPDY incurs very little initialization time because the connection is preestablished. On the other hand, it incurs a significant wait time. Importantly, this wait time is significantly higher than the initialization time for HTTP. This negates any advantages SPDY gains by reusing connections and avoiding connection setup. The wait times for SPDY are much greater because multiple requests are sent together or in close succession to the proxy. This increases delay as the proxy catches up in serving the requests to the client. Fig. 8 discussed in Section V-B shows this behavior.

B. Web Page Design and Object Requests

We now look at when different objects for a Web site are requested by the browser. One of the performance enhancements SPDY allows is for all objects to be requested in parallel without waiting for the response of outstanding objects. In contrast, HTTP has only one outstanding request per TCP connection unless pipelining is enabled.

We plot the request time (i.e., the time the browser sends out a request) for both HTTP and SPDY for four Web sites (due to space considerations) in Fig. 7. Two of these are news Web sites and two contain a number of photos and videos. SPDY, unlike what was expected, does not actually request all the objects at the same time. Instead for three of the four Web sites, SPDY requests objects in steps. Even for the one Web site where all the objects are requested in quick succession, we observe a delay between the first request and the subsequent requests. HTTP, on the other hand, requests objects continuously over time. The number of objects it downloads in parallel depends on the number of TCP connections the browser opens to each domain and across all domains.

We attribute this sequence of object requests to Web page design and how the browsers process them to identify constituent objects. Javascript and CSS files introduce interdependencies by requesting other objects. Table I highlights that Web sites make heavy use of JavaScript or CSS and contain anywhere from 2 to 73 different scripts and stylesheets. The browser does not identify all objects until these files are downloaded and processed. Moreover, browsers process some of these files (e.g., Javascripts) sequentially as these can change the layout of the page. This results in further delays. The overall impact on page load time depends on the number of such files in a Web page and the interdependencies in them.

To validate our assertion that SPDY is not requesting all the objects at once because of these interdependencies and also to understand better the higher wait time of objects, we built two test Web pages that consist of only a main HTML page and images that we placed on a test server (see Fig. 3). There were a total of 50 objects that needed to be downloaded as part of the Web page. We controlled the effect of domains by testing the two extremes: in one Web page, all the objects came from different domains, while in the second extreme all the objects came from the same domain. Fig. 8 shows the results of these two tests. Since there are no interdependencies in the Web page, we see that the browser almost immediately identifies all the objects that need to be downloaded after downloading the main HTML page (shown using red dots). SPDY then requests all the images on the page in quick succession (shown in green dots) in both cases. HTTP on the other hand, is affected by these extremes. When all objects are on different domains, the browser opens one connection to each domain up to a maximum number of connection (32 in the case of Chrome). When all the objects are

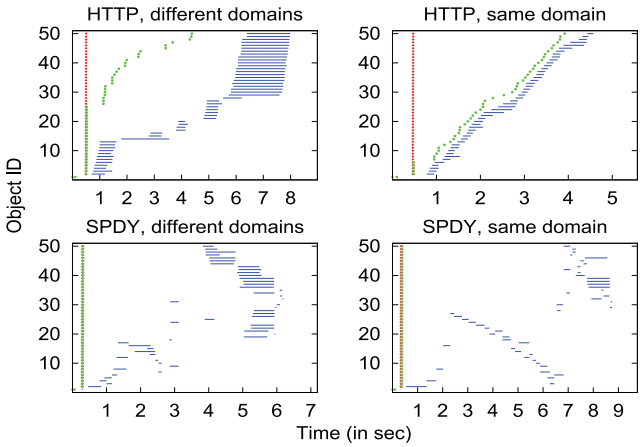

Fig. 8. Object request and download with HTTP and SPDY with test Web pages. Since these pages have no dependencies, SPDY requests all the objects right away, whereas HTTP is dependent on the number of TCP connections.

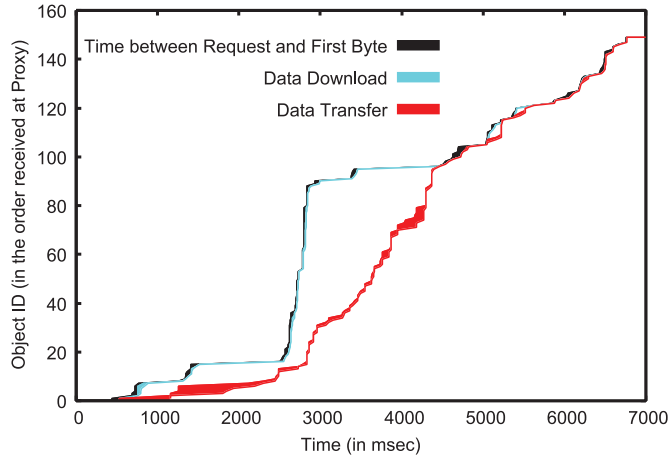

Fig. 9. This plot shows the time to first byte (black area) at the SPDY proxy, time to download the object from the Web server (cyan area) and the time to transfer the object to the client (red area). We see that the proxy quickly downloads the requested object from the Web server, but has to queue it locally for some time before it is able to transfer it to the client.

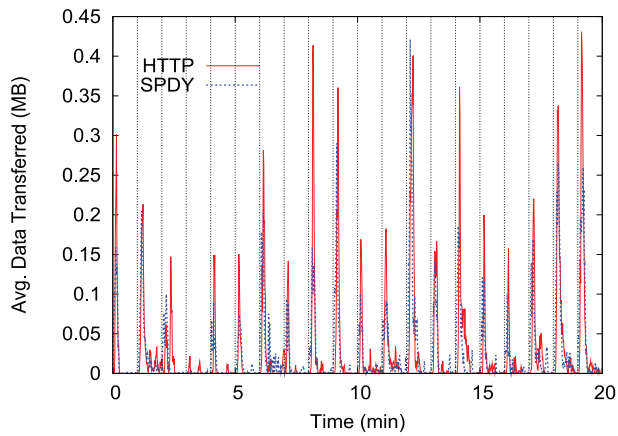

Fig. 10. Average data transferred from proxy to device every second. HTTP seems to transfer more data to the client, sometimes by as much as 100%.

on the same domain, browsers limit the number of concurrent connections (6 in the case of Chrome) but reuse the connections.

Note that while the requests for SPDY are sent out earlier (green dots) than HTTP, SPDY has much more significant delay until the first byte of data is sent back to the client (start of blue horizontal line). Moreover, we also observe especially in the different domain case, that if multiple objects are downloaded in parallel the time to receive the objects (length of blue line) is increased. We find in this experiment that removing all the interdependencies for SPDY does not significantly improve the performance. In our tests, HTTP had an average page load time of 5.29 and 6.80 s with single versus multiple domains, respectively. Conversely, SPDY averages 7.22 and 8.38 s with single or multiple domain tests. Consequently, prioritization alone does not address SPDY's performance in cellular networks.

C. Eliminating Server-Proxy Link Bottleneck

Figs. 7 and 8 show that while today's Web pages do not take full advantage of SPDY's capabilities, that is not a reason for the lack of performance improvements with SPDY in cellular networks. So as the next step, we focus on the proxy and see if the proxy-server link is a bottleneck.

In Fig. 9, we plot the sequence of steps at the proxy for a random Web site from one randomly chosen sample execution with SPDY. The figure shows the objects in the order of requests by the client. There are three regions in the plot for each object. The black region shows the time between when the object was requested at the proxy to when the proxy receives the first byte of response from the Web server. The next region, shown in cyan, represents the time it takes the proxy to download the object from the Web server, starting from the first byte that it receives. Finally, the red region represents the time it takes the proxy to transfer the object back to the client. It is clear from Fig. 9 that the link between the Web server and proxy is not the bottleneck. We see that in most cases, the time between when the proxy receives the request from the client to when it has the first byte of data from the Web server is very short (average of 14 ms with a max of 46 ms). The time to download the data, at an average of 4 ms, is also quite short. Despite having the data, however, we observe that the proxy is unable to send the data quickly to the client device. There is a significant delay between when the data was downloaded to the proxy to when it begins to send the data to the client.

We make an interesting observation here: With HTTP, the client does not request objects and queue them until the pending objects are downloaded. If individual object downloads take a while, the overall download process is also affected. SPDY, instead, requests all the objects in quick succession. While this is beneficial when there is sufficient capacity on the proxy-to-client link, SPDY does not benefit when that link is a bottleneck. Instead, it essentially moves queuing from the client to the proxy.

D. Throughput Between Client and Proxy

The previous result showed that the proxy was not able to transfer objects to the client quickly, resulting in long wait times for SPDY. Here, we study the average throughput achieved by SPDY and HTTP during the course of our experiments. Since each Web site is requested exactly 1 min apart, in Fig. 10 we align the start times of each experiment, bin the data transferred by SPDY and HTTP each second, and compute the average across all the runs.

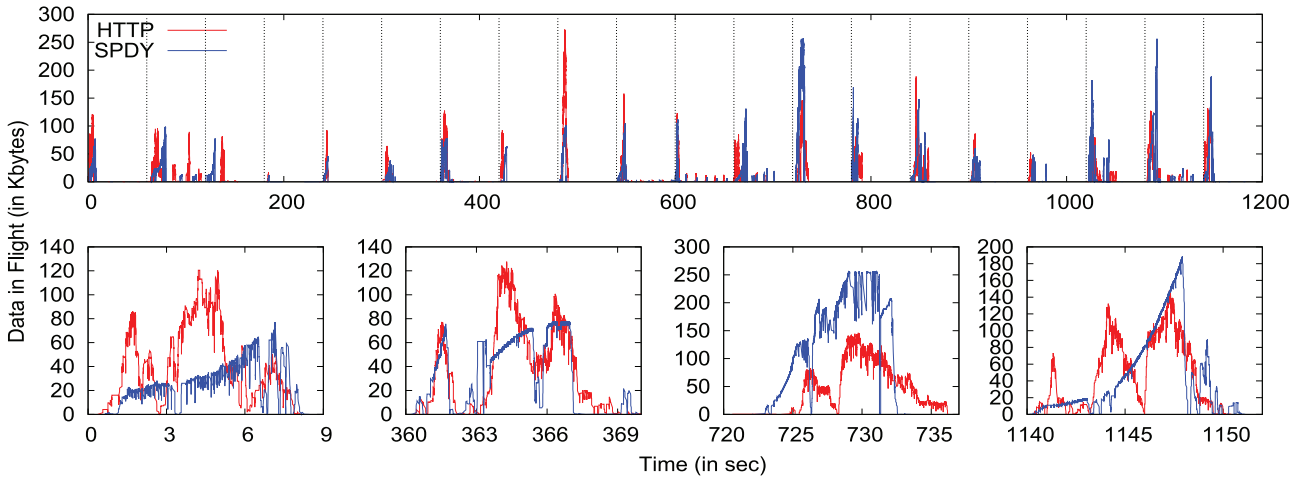

Fig. 11. Number of unacknowledged bytes for a random run with HTTP and SPDY.

The figure shows the average amount of data that was transferred during that second. The vertical lines seen every minute indicate the time when a Web page was requested. We see from the graph that HTTP, on average, achieves higher data transfers than SPDY. The difference sometimes is as high as 100%. This is a surprising result because, in theory, the network capacity between the client and the proxy is the same in both cases. The only difference is that HTTP uses multiple connections, each of which shares the available bandwidth, while with SPDY the single connection uses the entire capacity. Hence, we would expect the throughput to be similar, yet they are not. Since network utilization is determined by how TCP adapts to available capacity, we shift our attention to how TCP behaves in the cellular network.

E. Understanding TCP Performance

To understand the cause for the lower average throughput with SPDY, we look at how TCP performs with SPDY and with HTTP. We start by looking at the outstanding bytes in flight between the proxy and the client device with HTTP and SPDY. The number of bytes in flight is defined as the number of bytes the proxy has sent to the client that are awaiting acknowledgment. We plot the data from one random run of the experiment in Fig. 11.

Fig. 11 shows that there are instances where HTTP has more unacknowledged bytes, and other instances where SPDY wins. When we looked at the correlation between page load times and the number of unacknowledged bytes, we found that whenever the outstanding bytes is higher, it results in lower page load times. To illustrate this, we zoom into four Web sites (1, 7, 13, and 20) from the same run and plot them in the lower half of Fig. 11. For the first two Web sites, HTTP has more unacknowledged data, and hence the page load times were lower (by more than 1 s), whereas for 13 and 20, SPDY has more outstanding data and hence lower page load times (faster by 10 and 2 s, respectively). We see that the trend applied for the rest of the Web sites and other runs. In addition, we see in Web sites 1 and 20 that the growth in outstanding bytes (i.e., the growth of throughput) is quite slow for SPDY. We have already established in Fig. 9 that the proxy is not starved for data. Hence, the factors limiting the amount of data transferred could either be the sender's congestion window or the receiver's receive window.

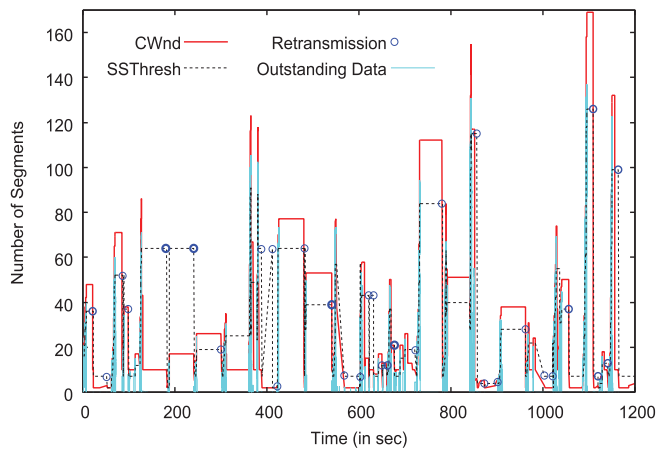

Fig. 12. `cwnd`, `ssthresh`, and outstanding data for one run of SPDY. The figure also shows times at which there are retransmissions.

Congestion Window Growth: We processed the packet capture data and extracted the receive window (`rwin`) advertised by the client. From the packet capture data, it was pretty clear that `rwin` was not the bottleneck for these experimental runs. So instead we focused on the proxy's congestion window and its behavior. To get the congestion window, we needed to tap into the Linux kernel and ran a kernel module (`tcp_probe`) that reports the congestion window (`cwnd`) and slow-start threshold (`ssthresh`) for each TCP connection.

Fig. 12 shows the congestion window, `ssthresh`, the amount of outstanding data, and the occurrence of retransmissions during the course of one random run with SPDY. First, we see that in all cases, the `cwnd` provides the ceiling on the outstanding data, indicating that it is the limiting factor in the amount of data transferred. Next, we see that both the `cwnd` and the `ssthresh` fluctuate throughout the run. Under ideal conditions, we would expect them to initially grow and then stabilize to a reasonable value. Finally, we see many retransmissions (blue circles) throughout the duration of the run (in our plot, the fatter the circle, the greater the number of retransmissions.)

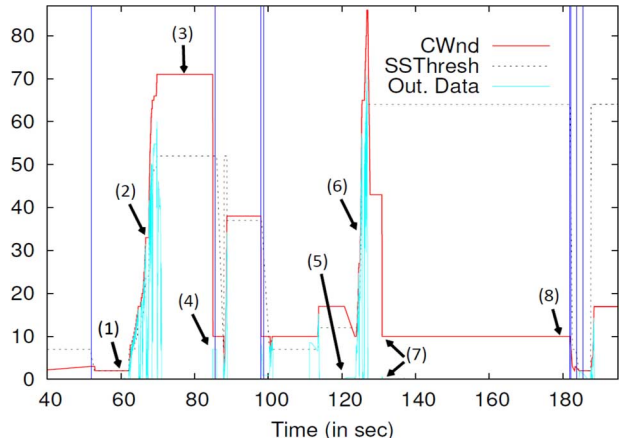

Fig. 13. `cwnd`, `ssthresh`, and outstanding data for three consecutive Web sites.

To gain a better understanding, we zoom into the interval between 40 and 190 s in Fig. 13. This represents the period when the client issues requests to Web sites 2–4. The vertical dashed line represents time instances where there are retransmissions. From Fig. 13, step (1), when accessing Web site 2 at time 60, we see that both the `cwnd` and `ssthresh` are small. This is a result of multiple retransmissions happening in the time interval 0–60 s (refer Fig. 12). In step (2), from 60 to 70 s, both the `cwnd` and `ssthresh` grow as data is transferred. Since the `cwnd` is higher than the `ssthresh`, TCP stays in congestion avoidance and does not grow as rapidly as it would in "slow-start." The pattern of growth during the congestion avoidance phase is also particular to TCP-Cubic (because it first probes and then has an exponential growth).

After about 70 s, in step (3), there is not any data to transfer and the connection goes idle until about 85 s. This is the key moment for performance loss: At step (4), when the proxy tries to send data, multiple effects are triggered. First, since the connection has been idle, a TCP parameter (`tcp_slow_start_after_idle`) is triggered. Intuitively, this parameter captures that fact that network bandwidth could have changed during the idle period, and hence it makes sense to discard all the estimates of the available bandwidth. As a result of this parameter, TCP reduces the `cwnd` to the default initial value of 10. Note that the `ssthresh` and retransmission timeout (RTO) values are left unmodified; as a result, the connection goes through slow start until `cwnd` grows beyond the `ssthresh`.

As described in Section II, the cellular radio transitions between idle and active states to conserve energy and share the radio resources. Devices transfer the most data when they are in the active (or DCH) state. They transition to idle after a small period of inactivity. When going from idle to active, the state machine imposes a *promotion* delay, which is typically around 2 s [16]. This promotion delay results in a period in which TCP does not receive any acknowledgments either. Since TCP's RTO value is not reset after an idle period, and this RTO value is much smaller than the promotion delay, it results in a TCP timeout and subsequent retransmissions (refer Fig. 12). As a consequence, `cwnd` is reduced, and the `ssthresh` is set to a value based on the `cwnd` (the specific values depend on the flavor of TCP). TCP then enters slow start, and `cwnd` and `ssthresh` grow back quickly to their previous values (again this depends on the version of TCP, and in this case depends on the behavior of TCP-Cubic). As a result of an idle and subsequent retransmission, a similar process repeats itself twice, at 90 and 120 s with the `cwnd` and `ssthresh`. Interestingly, at 110 s, we do not see retransmissions even though there was an idle period. We attribute this to the fact that the RTO value is large enough due to previous timeouts to accommodate the increased round-trip time after the idle time.

When Web site 3 is requested in step (5) at time 120, the `cwnd` and `ssthresh` grow as data is transferred [step (6)]. The Web site also transfers small amounts of data at around 130 s in step (7), after a short idle period. That causes TCP to reduce its `cwnd` to 10. However the idle period is short enough that the cellular network does not go idle. As a result, there are no retransmissions, and the `ssthresh` stays at 65 segments. The `cwnd` remains at 10 as no data were transferred after that time. When Web site 4 is requested in step (8) at 180 s, however, the `ssthresh` falls dramatically because there is a retransmission (TCP as well as the cellular network become idle). Moreover, there are multiple retransmissions as the RTT estimates no longer hold.

Understanding Retransmissions: One of the reasons for both SPDY and HTTP's performance issues is the occurrence of TCP retransmissions. Retransmissions result in the collapse of TCP congestion window, which in turn hurts throughput. We analyze the occurrence of retransmissions and its cause in this section.

There are on average 117.3 retransmissions for HTTP and 67.3 for SPDY. We observed in this section that most of the TCP retransmissions were spurious due to an overly tight RTO value. Upon close inspection of one HTTP run, we found all (442) retransmissions were in fact spurious. On a per-connection basis, HTTP has fewer retransmissions (2.9) since there are 42.6 concurrent TCP connections open on average. Thus, the 67.3 retransmits for SPDY results in much lower throughput. We also note from our traces that the retransmissions are bursty in nature and typically affect a few (usually one) TCP connections. Fig. 14 shows that even though HTTP has a higher number of retransmissions, when one connection's throughput is compromised, other TCP connections continue to perform unaffected. Since HTTP uses a "late binding" of requests to connections (by allowing only one outstanding request per connection), it is able to avoid affected connections and maintain utilization of the path between the proxy and the end-device. On the other hand, since SPDY opens only one TCP connection, all these retransmissions affect its throughput.

F. Validating the Impact of Cellular State Machine

In this experiment, we analyze the performance improvement gained by the device staying in the DCH state. Since there is a delay between each Web site request, we run a continual ping process that transfers a small amount of data every few seconds. We choose a payload that is small enough to not interfere with our experiments, but large enough that the state machine keeps the device in DCH mode.

Fig. 15 shows the CDF of the page load times for the different Web sites across the different runs. Unsurprisingly, the result

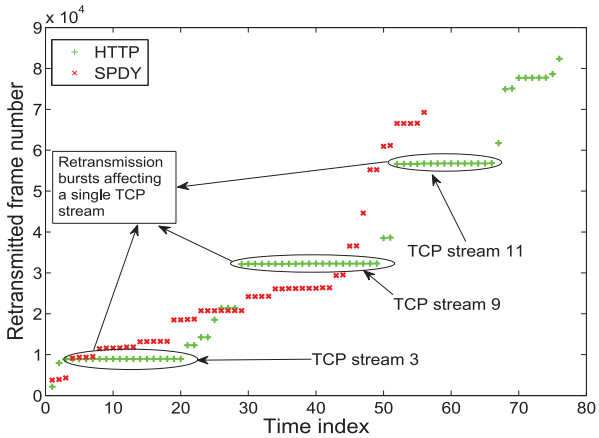

Fig. 14. Retransmission with HTTP and SPDY. We see that there are bursts of retransmissions affecting a single TCP connection. Since SPDY uses only one TCP connection, its throughput is impacted when there are retransmissions.

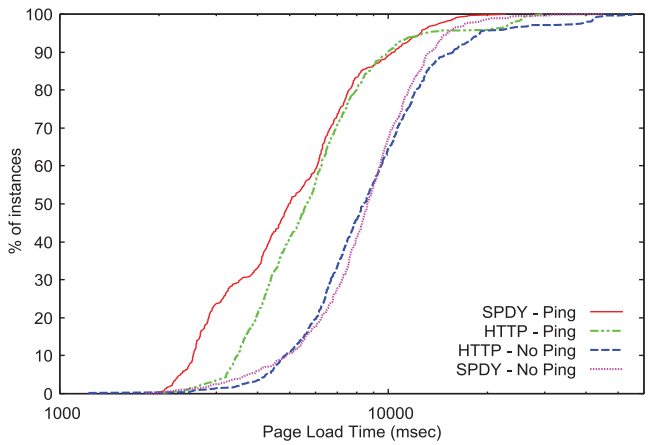

Fig. 15. Validating the impact of cellular RRC state machine. When the radio is continuously active, SPDY performs better than HTTP because it does not incur retransmissions.

shows that having the cellular network in DCH mode through continuous background ping messages significantly improves the page load time of both HTTP and SPDY. For example, more than 80% of the instances load in less than 8 s when the device sends continual ping messages, but only between 40% (SPDY) and 45% (HTTP) complete loading without the ping messages. Moreover, SPDY performs better than HTTP for about 60% of the instances with the ping messages. We also looked into the number of retransmissions with and without ping messages; not surprisingly, we observed that the number of retransmissions reduced by $\sim 91\%$ for HTTP and $\sim 96\%$ for SPDY indicating that TCP RTT estimation is no longer impacted by the cellular state machine. While this result is promising, it is not practical to keep the device in DCH state as it wastes cellular resources and drains device battery. Hence, mechanisms need to be built into TCP that account for the cellular state machine.

G. Does the Problem Disappear With LTE?

We analyze the performance of HTTP and SPDY over LTE in this section. LTE adopts an improved RRC state machine with a significantly smaller promotion delay. On the other hand, LTE also has lower round-trip times compared to 3G, which has the corresponding effect of having much smaller RTO values.

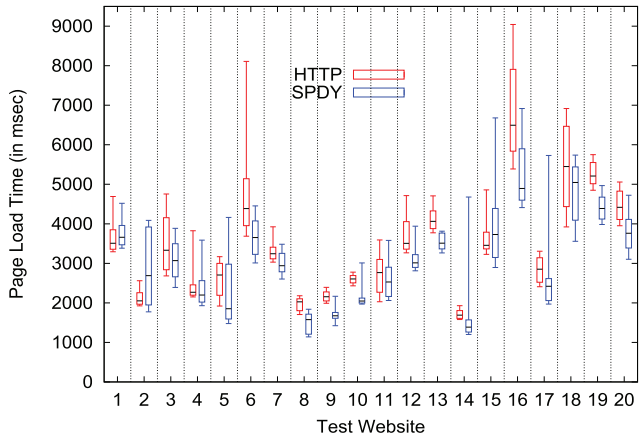

Fig. 16. Page load time of HTTP and SPDY over LTE. We see here that SPDY performs marginally better than HTTP.

We perform the same experiments using the same setup as in the previous 3G experiments, but connect to an LTE network with LTE USB laptop cards. Fig. 16 shows the box plot of page load times for HTTP and SPDY over LTE. As expected, we see that both HTTP and SPDY have considerably smaller page load times compared to 3G. We also see that HTTP performs just as well as SPDY, if not better, for the initial few pages. However, SPDY's performance is better than HTTP after the initial set of Web pages. We attribute this to the fact that LTE's RRC state machine addresses many of the limitations present in the 3G state machine, thereby allowing TCP's congestion window to grow to larger values and thus allowing SPDY to transfer data more quickly. We also looked at the retransmission data for HTTP and SPDY—the number of retransmissions reduced significantly with an average of 8.9 and 7.52 retransmissions per experiment with HTTP and SPDY (as opposed to 117 and 63 with 3G), respectively.

While the modified state machine of LTE results in better performance, we also wanted to see if it eliminated the issue of retransmission as a result of the state promotion delay. We focus on a short duration of a particular, randomly selected, run with SPDY in Fig. 17. The figure shows the congestion window of the TCP connection (in red), the amount of data in flight (in cyan), and the times when there are retransmissions (in black). The thicker retransmission lines indicate multiple retransmissions. We see from the figure that retransmissions occur after an idle period in LTE also. For example, at around 600 s, the proxy tries to send data to the device after an idle period; timeouts occur after the transmission of data, leading to retransmissions; and the congestion window collapses. This result leads us to believe that the problem persists even with LTE, albeit less frequently than with 3G.

H. Effect of Other Users and Load

In addition to experiments described above, we also performed experiments to understand the impact of multiple users in a congested cell. The experiments were conducted with four laptops simultaneously accessing the test Web pages (with each laptop starting in a staggered manner, 10 s after another). We also added congestion to the cell by having four other laptops continuously download data in the background. We verified the

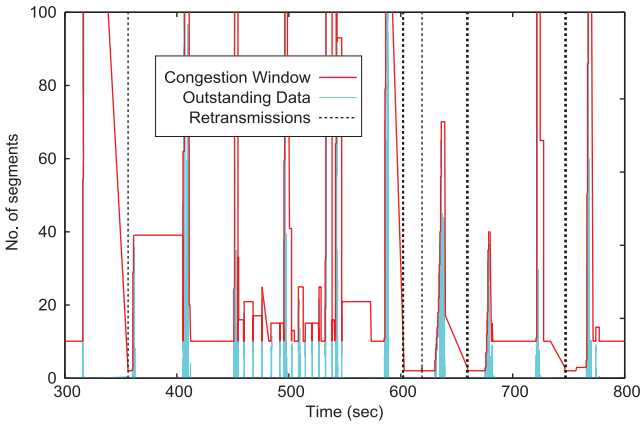

Fig. 17. SPDY's congestion window and retransmissions over LTE. We see that SPDY still incurs retransmissions albeit significantly lower than with 3G.

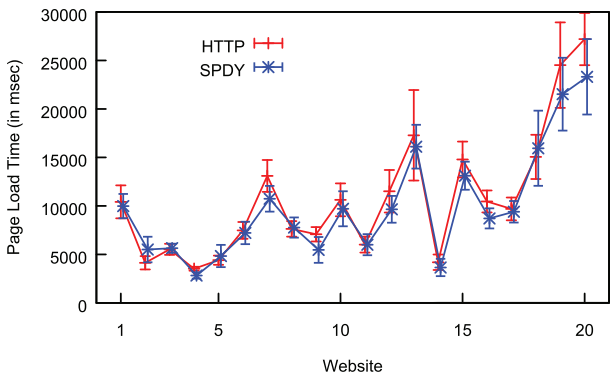

Fig. 18. Page load time over 3G with multiple users and background load.

effect of congestion by observing that the throughput of these four "background load" laptops go down proportionally to the data downloaded by the test laptops.

We show the results of this experiment in Fig. 18. Similar to the previous single-user experiments, there is very little difference between the page load times using HTTP and SPDY. Note that the absolute values are different from the single user experiments; we attribute it to the fact that the measurement was done at a different time period (day and time). We also conducted the same experiment with multiple users, but no background load. While we do not report the results here due to lack of space, we observed those trends to be similar. These results lead us to believe that the fundamental problem remains even with multiple users, under load in a congested cell.

I. Summary and Discussion

We see from these results how the interaction between the different layers affects performance. First, we see Web sites sending and/or requesting data periodically (ads, tracking cookies, Web analytics, page refreshes, etc.). We also observe that a key factor affecting performance is the independent reaction of the transport protocol (i.e., TCP) and the cellular network to inferred network conditions.

TCP implementations reset their `cwnd` statistics after an idle period as the network capacity might have changed. That in itself would not be a problem in wired networks as the `cwnd` will grow back up quickly. However, in conjunction with the cellular network's idle-to-active promotion delay, it results in unintended consequences. Spurious retransmissions occurring due to the promotion delay cause the `ssthresh` to fall to the `cwnd` value. As a result, when TCP tries to recover, it goes through slow start for a short duration, and then switches to congestion avoidance, even for small number of segments. From a TCP standpoint, this defeats the design intent where short transfers that do not have the potential of causing congestion (and loss) should be able to rapidly acquire bandwidth, thus reducing transfer time. This difficulty of transport protocols "shutting down" after an idle period at just the time when applications wake up and seek to transfer data (and therefore requiring higher throughput) is not new and has been observed before [17]. However, the process is further exacerbated in cellular networks with the existence of a large promotion delay. These interactions thus degrade performance, including causing multiple (spurious) retransmissions that have significant undesirable impacts on the individual TCP connection behavior.

Our results also point to a *limitation in TCP implementations*. Existing implementations retain information about the latency profile (i.e., RTT estimates) after an idle period. With the cellular state machine, however, the *latency profile changes* after an idle period. Since the estimates are inaccurate, it results in spurious retransmissions. We notice that LTE, despite having an improved state machine, is still susceptible to retransmissions after an idle period. When we keep the device in active mode continuously, we transform the cellular network to behave more like a traditional wired (and also a WiFi) network in terms of latency profile. Consequently, we see results similar to the ones seen over wired networks.

VI. Potential Improvements

Having identified the interactions between TCP and the cellular network as the root cause of the problem, in this section, we propose steps that can minimize their impact.

A. Using Multiple TCP Connections

The observation that using a single TCP connection causes SPDY to suffer because of retransmissions suggests a need to explore the use of multiple TCP connections. We explore this option by having the browser use 20 SPDY connections to a single proxy process listening on 20 different ports.[3] However, the use of multiple TCP connections *did not* help in improving the page load times for SPDY. This is primarily because, with SPDY, requests are issued to each connection up front. As a result, if a connection encounters retransmissions, pending objects requested on that connection are delayed. What is required is a late binding of the response to an "available" TCP connection (meaning that it has an open congestion window and can transmit data packets from the proxy to the client at that instant) and avoiding a connection that is currently suffering from the effects of spurious timeouts and retransmissions. Such a late binding would allow the response to come back on any available TCP connection, even if the request was sent out on a dif-

[3]We make use of a proxy auto config (PAC) file that dynamically maps the proxy address and one of the 20 ports to each object requested.

ferent connection. This takes advantage of SPDY's capability to send the requests out in a "burst," and allows the responses to be delivered to the client as they arrive back, avoiding any "head-of-the-line blocking."

B. TCP Implementation Optimizations

1) Resetting RTT Estimate After Idle: There is a fundamental need to decay the estimate of the available capacity of a TCP connection once it goes idle. The typical choice made today by implementations is to just reset `cwnd` to the initial value. The RTT estimate, however, is left untouched by implementations. The RTT estimate drives the RTO value and hence controls when a packet is retransmitted. Not resetting the RTT estimate may be acceptable in networks that have mostly "stable" latency characteristics (e.g., a wired or WiFi network), but as we see in our observations with the cellular network, this leads to substantially degraded performance. The cellular network has vastly varying RTT values. In particular, the idle to active transition (promotion) can take a few seconds. Since the previous RTT estimate derived when the cellular connection was active may have been of the order of tens or hundreds of milliseconds, there is a high probability of a spurious timeout and retransmission of packets after the idle period. Thus, the interaction of TCP with the RRC state machine of the cellular network has to be properly factored in to achieve the best performance. Our recommended approach is to reset the RTT estimate as well, to the initial default value (of multiple seconds). This causes the RTO value to be larger than the promotion delay for the 3G cellular network, thus avoiding spurious timeouts and unnecessary retransmissions. This, in turn, allows the `cwnd` to grow rapidly, ultimately reducing page load times.

2) Benefit of Slow Start After Idle?: One approach we also considered was whether avoiding "slow start after idle" would improve performance. We disabled slow start by setting `tcp_slow_start_after_idle` to 0 and studied the improvement in page load time. Fig. 19 plots the relative difference between the average page load time of the different Web sites with and without slow start. A negative value on the y-axis indicates that disabling slow start after idle is beneficial, while a positive value indicates that performing slow start is beneficial. We see that the benefits vary across different Web sites. Our packet traces indicate that the amount of outstanding data (and hence throughput) is quite similar in both the cases. The number of retransmitted packets seem similar under good conditions, but disabling the parameter runs the risk of having lots of retransmissions under congestion or poor channel conditions since the `cwnd` value is inaccurate after an idle period. In some instances, `cwnd` grows so large with the parameter disabled, that the receive window becomes the bottleneck and negates the benefit of a large congestion window at the sender.

3) Impact of TCP Variants: We replaced TCP Cubic with TCP Reno to see if modifying the TCP variant has any positive impact on performance. We find in Table II that there is little to distinguish between Reno and Cubic for both HTTP and SPDY over 3G. We see that the average page load time across all the runs of all pages is better with Cubic. Average throughput is quite similar with Reno and Cubic, with SPDY achieving the highest value with Cubic. While this seemingly

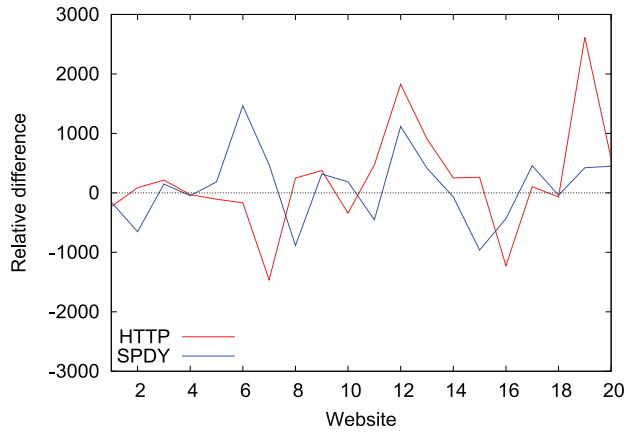

Fig. 19. Difference in page load times with and without TCP "slow start after idle." Disabling slow start after idle does not seem to improve page load time.

TABLE II
COMPARISON OF HTTP AND SPDY TO DIFFERENT TCP VARIANTS (RENO AND CUBIC). THE SPECIFIC VARIANT OF TCP DOES NOT HELP DIFFERENTIATE SPDY FROM HTTP

	Reno		Cubic	
	HTTP	SPDY	HTTP	SPDY
Avg. Page Load (msec)	9690.84	9899.95	9352.58	8671.09
Avg. Throughput (KBps)	121.88	119.55	115.36	129.79
Max. Throughput (KBps)	1024.74	528.88	889.33	876.98
Avg. `cwnd` (# segments)	10.45	24.16	10.59	52.11
Max. `cwnd` (# segments)	22	48	22	197

contradicts the result in Fig. 10, note that this result is the average across all times (ignoring idle times), while the result in Fig. 10 considers the average at that 1-s instant. Indeed, the maximum throughput result confirms this: HTTP with Cubic achieves a higher throughput than SPDY with Cubic. SPDY with Reno does not grow the congestion window as much as SPDY with Cubic. This probably results in SPDY with Reno having the worst page load time across the combinations.

4) Cache TCP Statistics?: The Linux implementation of TCP caches statistics such as the slow start threshold and round-trip times by default and reuses them when a new connection is established. If the previous connection had statistics that are not currently accurate, then the new connection is negatively impacted. Note that since SPDY uses only one connection, the only time these statistics come into play is when the connection is established. It can potentially impact HTTP, however, because HTTP opens a number of connections over the course of the experiments. We conducted experiments where we disabled caching. Interestingly, we find from our results that both HTTP and SPDY experience reduced page load times. For example, for 50% of the runs, the improvement was about 35%. However, there was very little to distinguish between HTTP and SPDY.

VII. RELATED WORK

There have been several efforts that are related to our work. These can be classified broadly into work on SPDY, work on radio resource management, and work on TCP optimizations.

SPDY: Researchers at Google originally showed the performance bottlenecks associated with Web delivery due to

the use of HTTP. They pinpointed the limitations of TCP as the root-cause of these bottlenecks and proposed SPDY as an approach to overcome the bottlenecks [3]. However, many of their results were based on wired network performance. More recently, Welsh *et al.* [4] used dummynet to emulate cellular network-like characteristics to show that SPDY improves mobile Web performance by as much as 23%. However, as we show in this paper, the real-world results with SPDY are not as optimistic and that there are important cross-layer interactions that are not captured in such emulated settings. Finally, independent of our work, Wang *et al.* [18] study the impacts of using SPDY across different dimensions. Like our paper, they also conclude that SPDY does not definitively outperform HTTP and that the outcome depends on multiple factors. They also show that object interdependencies in Web pages restrict the capabilities that SPDY offers. Unlike our paper, however, they do not focus on the mobile Web performance or on trying to understand the cross-layer aspects that affect the performance of these protocols.

Radio Resource Management: The use of cellular state machines has prompted several attempts to improve general application performance over cellular networks (e.g., [16] and [19]). Specifically, TOP and TailTheft study efficient ways of utilizing radio resources by optimizing timers for state promotions and demotions. However, despite the knowledge about state machines, their impact on TCP performance has not yet been well understood. Erman *et al.* [20] study the use of caching at different levels (e.g., nodeB, RNC) of a 3G cellular network to reduce download latency of popular Web content. Aucinas *et al.* [21] study the impact of mobile applications that provide continuous online presence. They find that these apps are "chatty" in nature. Such chatty communication would mitigate the promotion delay problem (like we show in Fig. 15), but as the study shows, such apps introduce other problems (like draining device battery and unnecessarily consuming precious radio resources).

TCP Optimizations: With regards to TCP, several proposals have tried to tune TCP parameters to improve its performance [22] and address issues like HOL blocking and multihoming. Recently, Google proposed in an IETF RFC 3390 [23] to increase the TCP initial congestion window to 10 segments to show how Web applications will benefit from such a policy. As a rebuttal, Gettys [24] demonstrated that changing the initial TCP congestion window can indeed be very harmful to other real-time applications that share the broadband link and attributed this problem to one of "buffer bloat." As a result. Gettys proposed the use of HTTP pipelining to provide improved TCP congestion behavior. Ramjee *et al.* [25] recognize how challenging it can be to optimize TCP performance over 3G networks exhibiting significant delay and rate variations. They use an ACK regulator to manage the release of ACKs to the TCP source so as to prevent undesired buffer overflow. Our work inspects in detail how SPDY and HTTP behave due to TCP over cellular networks. In this paper, we investigate in detail how congestion window growth affects download performance for HTTP and SPDY in cellular networks. In particular, we demonstrate how idle-to-active transition at different protocol layers results in unintended performance degradations. We show that a spurious timeout is caused by the fact that TCP stays with its original estimate for the RTT despite a promotion delay when transitioning from idle to active states. The RTO estimate derived over multiple round trips during the active period of a TCP connection is not only invalid during this promotion period, and results in significant performance impact. Note that the problem of spurious retransmissions due to a variable-delay channel is well known, and solutions have been implemented to mitigate its effects (e.g., F-RTO [26]). However, studies [27], [28] show that spurious timeouts do not drastically impact TCP performance in practice. Moreover, they seem to be infrequent events in UMTS networks with the ratio between spurious timeouts and other congestion recovery events experienced by TCP flows being low [28]. This could be because delay variations become critical whenever they are on the order of seconds [29]. The problem identified in this paper falls in this realm. Here, the timeout occurs because of a long, but predictable, cellular state promotion delay. Hence, we suggest using conservative ways to manage the RTO estimate whenever the device transitions from idle to active.

VIII. Conclusion

Mobile Web performance is one of the most important measures of users' satisfaction with their cellular data service. We systematically study two of the most prominent Web access protocols used today, HTTP and SPDY, through field measurements on a production cellular network. In cellular networks, there are interactions across protocol layers that limit the performance of both SPDY as well as HTTP. As a result, in contrast to existing studies on wired and WiFi networks, there is no clear performance improvement with SPDY in cellular networks. In particular, we show that TCP performance is significantly impacted when a connection comes out of idle state. Given the high promotion delay when a cellular end-device goes from idle to active, retaining TCP's RTT estimate across this transition results in spurious timeouts and corresponding retransmissions. This particularly punishes SPDY, which depends on the single TCP connection. This connection is hit with the spurious retransmissions, and hence all the cascading effects of TCP's congestion control mechanisms like lowering `cwnd`, etc. This ultimately reduces throughput and increases page load times. We propose a holistic approach to considering all the TCP implementation features and parameters to improve mobile Web performance and thereby fully exploit SPDY's advertised capabilities.

References

[1] L. Popa, A. Ghodsi, and I. Stoica, "HTTP as the narrow waist of the future Internet," in *Proc. Hotnets-IX*, 2010, pp. 6:1–6:6.
[2] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic forecasts achieve high throughput and low delay over cellular networks," in *Proc. USENIX NSDI*, Apr. 2013, pp. 459–472.
[3] Google, "SPDY: An experimental protocol for a faster Web," [Online]. Available: http://www.chromium.org/spdy/spdy-whitepaper
[4] M. Welsh, B. Greenstein, and M. Piatek, "SPDY performance on mobile networks," Apr. 2012 [Online]. Available: https://developers.google.com/speed/articles/spdy-for-mobile
[5] F. Zarinni, A. Chakraborty, V. Sekar, S. R. Das, and P. Gill, "A first look at performance in mobile virtual network operators," in *Proc. ACM IMC*, 2014, pp. 165–172.
[6] X. Xu *et al.*, "Investigating transparent Web proxies in cellular networks," in *Proc. PAM*, 2015, pp. 262–276.

[7] Google, "Chrome data compression proxy," [Online]. Available: https://developer.chrome.com/multidevice/data-compression
[8] V. Agababov *et al.*, "Flywheel: Google's data compression proxy for the mobile Web," in *Proc. USENIX NSDI*, 2015, pp. 367–380.
[9] "Web technology surveys," Jun. 2013 [Online]. Available: http://w3techs.com/technologies/details/ce-spdy/all/all
[10] S. U. Khaunte and J. O. Limb, "Statistical characterization of a World Wide Web browsing session Georgia Institute of Technology, Tech. Rep., 1997.
[11] 3GPP, "3GPP TS 36.331: Radio Resource Control (RRC)," 2014 [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/36331.htm
[12] "Hypertext Transfer Protocol version 2 (HTTP/2)," 2015 [Online]. Available: https://http2.github.io/faq/
[13] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "Demystifying page load performance with WProf," in *Proc. USENIX NSDI*, Apr. 2013, pp. 473–486.
[14] S. Sanadhya and R. Sivakumar, "Adaptive flow control for TCP on mobile phones," in *Proc. IEEE INFOCOM*, 2011, pp. 2912–2920.
[15] "Squid caching proxy," [Online]. Available: http://www.squid-cache.org
[16] F. Qian *et al.*, "TOP: Tail optimization protocol for cellular radio resource allocation," in *Proc. IEEE ICNP*, 2010, pp. 285–294.
[17] L. Kalampoukas, A. Varma, K. K. Ramakrishnan, and K. Fendick, "Another examination of the use-it-or-lose-it function on TCP traffic," presented at the ATM Forum/96–0230 TM Working Group, 1996.
[18] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How speedy is SPDY?," in *Proc. USENIX NSDI*, Apr. 2014, pp. 387–399.
[19] H. Liu, Y. Zhang, and Y. Zhou, "Tailtheft: Leveraging the wasted time for saving energy in cellular communications," in *Proc. MobiArch*, 2011, pp. 31–36.
[20] J. Erman *et al.*, "To cache or not to cache: The 3G case," *IEEE Internet Comput.*, vol. 15, no. 2, pp. 27–34, Mar.–Apr. 2011.
[21] A. Aucinas *et al.*, "Staying online while mobile: The hidden costs," in *Proc. ACM CoNEXT*, 2013, pp. 315–320.
[22] J. Stone and R. Stewart, "Stream Control Transmission Protocol (SCTP) checksum change," Sep. 2002 [Online]. Available: http://tools.ietf.org/html/rfc3309.html
[23] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis, "Increasing TCP's initial window," Feb. 2013 [Online]. Available: http://tools.ietf.org/html/draft-ietf-tcpm-initcwnd-08.html
[24] J. Gettys, "IW10 considered harmful," Aug. 2011 [Online]. Available: http://tools.ietf.org/html/draft-gettys-iw10-considered-harmful-00.html
[25] M. C. Chan and R. Ramjee, "TCP/IP performance over 3G wireless links with rate and delay variation," in *Proc. ACM MobiCom*, Atlanta, GA, USA, 2002, pp. 71–82.
[26] P. Sarolahti, M. Kojo, and K. Raatikainen, "F-RTO: An enhanced recovery algorithm for TCP retransmission timeouts," *Comput. Commun. Rev.*, vol. 33, no. 2, pp. 51–63, Apr. 2003.
[27] M. Allman and J. Griner, "TCP behavior in networks with dynamic propagation delay," in *Proc. IEEE GLOBECOM*, 2000, vol. 2, pp. 1103–1108.
[28] F. Vacirca, T. Ziegler, and E. Hasenleithner, "An algorithm to detect TCP spurious timeouts and its application to operational UMTS/GPRS networks," *Comput. Netw.*, vol. 50, no. 16, pp. 2981–3001, 2006.
[29] M. Scharf, M. Necker, and B. Gloss, "The sensitivity of TCP to sudden delay variations in mobile networks," *Networking*, vol. 3042, pp. 76–87, 2004.

Jeffrey Erman received the B.S. degree from the University of Regina, Regina, SK, Canada, in 2005, and the M.S. degree from the University of Calgary, Calgary, AB, Canada, in 2007, both in computer science.

He is a Principal Inventive Scientist with AT&T Labs—Research, Bedminster, NJ, USA. His research focuses are in IP traffic measurement, network traffic classification, cross-layer protocol analysis, network management, and performance analysis.

Vijay Gopalakrishnan (M'09) received the M.S. and Ph.D. degrees in computer science from the University of Maryland, College Park, MD, USA, in 2003 and 2006, respectively.

He has been with AT&T since 2006 and has worked on innovative solutions in the space of network management, content delivery, and the mobile Web. He is currently a Director with the Network and Service Quality Management Center, AT&T Labs—Research, Bedminster, NJ, USA, leading a team of researchers focused on systems challenges in the architecture, protocols and management of networks.

Rittwik Jana (M'02) received the Bachelor of Engineering in Electrical and Electronics degree from the University of Adelaide, Adelaide, Australia, in 1994, and the Ph.D. degree in telecommunications engineering from the Australian National University, Canberra, Australia, in 1999.

He is a Lead Inventive Scientist with AT&T Labs—Research, Bedminster, NJ, USA. His research interests span Internet technologies, networked video streaming, cellular networks and systems, and intelligent service composition using VNFs.

Kadangode K. Ramakrishnan (S'76–A'83–M'03–SM'04–F'05) received the M.S. degree in automation from the Indian Institute of Science, Bangalore, India, in 1978, and the M.S. and Ph.D. degrees in computer science from the University of Maryland, College Park, MD, USA, in 1981 and 1983, respectively.

He is a Professor with the Computer Science and Engineering Department of the University of California, Riverside, CA, USA. From 1994 until 2013, he was with AT&T, most recently a Distinguished Member of Technical Staff with AT&T Labs—Research, Florham Park, NJ, USA. Prior to 1994, he was a Technical Director and Consulting Engineer in Networking with Digital Equipment Corporation, Littleton, MA, USA. Between 2000 and 2002, he was with TeraOptic Networks, Inc., Sunnyvale, CA, USA, as Founder and Vice President. He has published more than 200 papers and has 145 patents issued in his name.

Dr. Ramakrishnan is also an AT&T Fellow, recognized in 2006 for his work on congestion control, traffic management and VPN services, and for fundamental contributions on communication networks with a lasting impact on AT&T and the industry. He has been on the editorial board of several journals and has served as the TPC Chair and General Chair for several networking conferences. He received an AT&T Technology Medal in 2013 for his work on mobile video delivery strategy and optimization. His work on the "DECbit" congestion avoidance protocol received the ACM SIGCOMM Test of Time Paper Award in 2006.