# Can HTTP/2 Really Help Web Performance on Smartphones?

Yi Liu, Yun Ma, ‡Xuanzhe Liu, and Gang Huang

*Key Lab of High-Confidence Software Technology (Peking University), Ministry of Education, Beijing China*

{*liuyi14, mayun, liuxuanzhe, hg*}*@pku.edu.cn*

***Abstract*—HTTP/2 is the next-generation Web protocol based on Google's SPDY protocol, and attempts to solve the shortcomings and inflexibilities of HTTP/1.x. As smartphones become the main access channel for Web services, we are curious if HTTP/2 can really help the performance of Web browsing. In this paper, we conduct a measurement study on the performance of HTTP/2 and HTTPS to reveal the mystery of HTTP/2. We clone the Alexa top 200 websites into our own server, and revisit them through HTTP/2-enabled proxy, and HTTPS-enabled proxy, respectively. We compare HTTP/2 and HTTPS as a transport protocol to transfer Web objects to identify the factors that may affect HTTP/2, including Round-Trip Time (RTT), bandwidth, loss rate, number of objects on a page, and objects sizes. We find that HTTP/2 hurts with high packet loss, but helps many small objects. The computation and dependencies of fetching Web objects reduce the performance improvement of HTTP/2, and sometimes can even hurt the performance of page loading. At last, we test the server push feature of HTTP/2 to leverage the performance.**

***Index Terms*—[1] Mobile; HTTP/2; SPDY; HTTP/1.x**

## I. Introduction

With the popularity of mobile devices such as smartphones and tablets, people are used to accessing various Web services via their devices, e.g., online shopping, news reading, and Web search [18]. From service computing perspective, mobile devices are playing the more and more important role as Web services consumers.

As most Internet-scale Web services are RESTful ones that are delivered with Web contents, the HTTP plays as the major protocol (HTTP/1.1 is the defacto standard since 1999), and has served the Web well for more than 15 years. However, as the Web services grow increasingly complex to provide more contents and functionalities, the shortcomings and inflexibilities of HTTP become more and more urgent to solve [21] [7], e.g., the sluggish page load, insecure content, redundant transfer, etc.

HTTP/2 is the next evolution of HTTP, which is maintained by the *IETF HTTP Working Group*. Based on Google's SPDY protocol, HTTP/2 attempts to outcome the shortcomings of HTTP, and focuses on performance, e.g., end-user perceived latency, network, and server resource usage. HTTP/2 benefits from multiplexing and concurrency, stream dependencies, header compressions, and server push. After more than two years' discussions, 17 drafts, and 30 implements, IETF HTTP Working Group for publication as standards-track RFCs approved the HTTP/2 and associated HPACK specifications in February of 2015.

In this paper, we mainly dive into the uncovered question, i.e., whether HTTP/2 help Web services performance on smartphones. Prior work shows that SPDY can either help or sometimes hurt the page load time (PLT) of Web pages when browsing real Web pages. We are curious if HTTP/2 really helps for mobile devices with limited capabilities and worse network. Even though HTTP/2 could work without SSL as the standard defines, both Chrome and FireFox do require the SSL, so we compare HTTP/2 with HTTPS rather than HTTP in this paper. At last, we test the server push feature of HTTP/2.

Both HTTP/2 and HTTP/1.x can be affected by many factors external to the protocols, including the network parameters (e.g. packet loss, bandwidth, and RTT), and the features of Web pages (e.g. object size, objects number). Users often use mobile browsers to visit Web services presented as Web pages, and we need to consider the dependencies of computation and fetching Web objects in such situation. We divide our experiments into two parts. First, we compare HTTP/2 and HTTPS as a transport protocol to transfer Web objects to identify the factors that could affect HTTP2, including RTT, bandwidth, loss rate, number of objects on a page, and objects' sizes. We find that HTTP/2 hurts with high packet loss, but helps with many small objects. Considering the real Web pages loading in mobile browser, we load the Alexa top 200 websites through HTTP/2-enabled proxy, and HTTPS-enabled proxy, respectively. The computation and dependencies of loading Web pages reduce the performance improvement of HTTP/2, and sometimes even hurt the performance of page loading.

More specifically, this paper tries to answer the following four research questions:

- **RQ1: Can HTTP/2 help RESTful Web service performance on smartphones?**
- **RQ2: How could those factors affect HTTP/2's performance?**
- **RQ3: Why does HTTP/2 help or hurt under certain conditions?**
- **RQ4: How does HTTP/2 perform when loading real Web pages in mobile browser?**

The remainder of the paper is organized as follows. Section II introduces background of HTTP/2. Section III introduces the measurement methodology and how we conduct our ex-

[1]‡: corresponding author: liuxuanzhe@pku.edu.cn

periments. Section IV presents the results and analysis of our experiments. Section V presents the related work, and Section VI concludes this paper.

## II. Background

In this section, we demonstrate the shortcomings and inflexibilities of HTTP/1.1, and describe the new features and improvements of HTTP/2.

### A. Shortcomings and inflexibility of HTTP/1.1

HTTP/1.1 is the defacto standard of HTTP, and has served the Web for more than fifteen years since the standardization. However, the Web has changed a lot to make it outdated. According to HTTP Archive [2], a Web page is becoming more and more complex. It may take more than 90 requests over 35 TCP connections to 16 different hosts to load a Web application, which may transfer about 1.9MB data on average. However, HTTP/1.1 practically only allows one outstanding request per TCP connection, and HTTP/1.1 is very latency sensitive, and has had enough of head of line blocking problem [1].

Although Web developers have come up with many best practices like domain sharding, spriting, and inlining and concatenation of resources, these techniques have their own shortcommings [21].

### B. HTTP/2

HTTP/2 derives from SPDY, and addresses those shortcomings of HTTP/1.1. HTTP/2 maintains a single, multiplexed connection, replacing the multiple connections per domain that browsers opened up in HTTP/1.x. HTTP/2 compresses header data and sends it in a concise, binary format, rather than the plain text format used previously.

Frame is the smallest unit of communication in HTTP/2, containing a frame header, which at minimum identifies the stream to which the frame belongs. All HTTP/2 communicationis performed within a connection that can carry any number of bidirectional streams, each of which is a bidirectional flow of bytes. In turn, each stream communicates in messages, which consist of one or multiple frames, each of which may be interleaved and then reassembled via the embedded stream identifier in the header of each individual frame. There are four key features of HTTP/2:

- **One connection per origin.** One connection per origin significantly reduces the associated overhead: fewer sockets to manage along the connection path, smaller memory footprint, and better connection throughput. It also may lead to other benefits, such as better compression through use of a single compression context, less time in slow-start, faster congestion and loss recovery.
- **Request Prioritization.** HTTP message can be split into many individual frames, the exact order in which the frames are interleaved and delivered can be optimized to further improve the performance. The browser can immediately dispatch each request the moment the resource is discovered, specify the priority of each stream, and let the server determine the optimal response delivery. This eliminates unnecessary request queuing latency and allows us to make the most efficient use of each connection
- **Header Compression.** Each HTTP transfer carries a set of headers that describe the transferred resource and its properties. In HTTP/1.x, this metadata is always sent as plain text and adds anywhere from 500 to 800 bytes of overhead per request, and kilobytes more if HTTP cookies are required. Instead of retransmitting the same data on each request and response, HTTP/2 uses "header table" on both client and server to track and store previously sent key-value pairs and incrementally updates them.
- **Server Push.** In additional to the response for the original request, the server can push additional resources to the client, which can reduce number of requests.

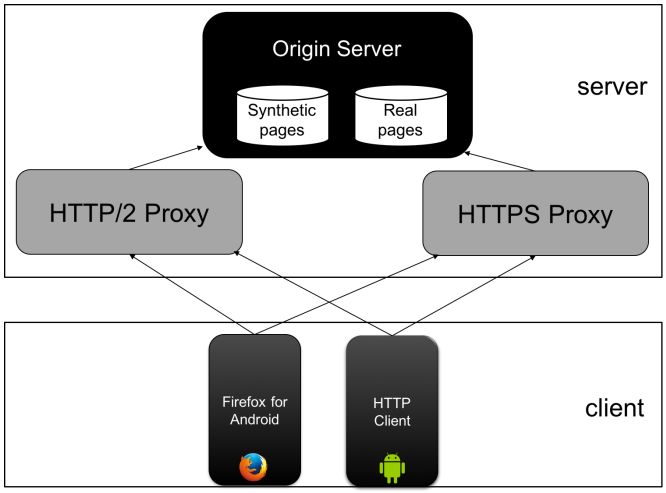



Figure 1: Experiment Setup

## III. Measurement Methodology

Prior works show that SPDY can either help or sometimes hurt the page load time when browsing real Web pages [24], [16], [13]. We are curious if HTTP/2 really helps for mobile devices with limited capabilities and worse network conditions. Even though HTTP/2 could work without SSL as the standard defines, both Chrome and Firefox do require the SSL, so we compare HTTP/2 with HTTPS instead of plain HTTP in this paper.

We divide our experiments into two parts. First, we compare HTTP/2 and HTTPS as a transport protocol to transfer Web objects to identify the factors that may affect HTTP/2, including Round Trip Time (RTT), bandwidth, packet loss rate, number of objects on a page, and objects sizes. We need to address that loading a Web page is not as simple as fetching all resources in parallel. For example, while loading a page, browsers usually do not download any images until JavaScript and CSS files are fetched and processed. CSS files may import other CSS files, and the browser can not know about that in advance. Some scripts generate new resources for the browsers to fetch. [23] shows that dependencies between

network activities (fetching Web objects) and computation (HTML parsing, JavaScript execution) have significant impacts on page loading time, so we compare page load time (PLT) when loading Web pages using real browsers with either HTTP/2 or HTTPS. PLT is calculated from initiation (when you click on a page link or type in a Web address) to completion (when the page is fully loaded in the browser). We listen the *onLoad*[2] event emitted by browser to get the PLT. When considering to load the real Web pages, we clone the home pages of the Alexa top 200 websites into our own local server, and revisit them through HTTP/2-enabled proxy and HTTPS-enabled proxy with FireFox for Android, respectively.

### A. Experiment setup

TABLE I: Factors may affect HTTP/2 and HTTPS performance.

| Factor | Range | High |
|---|---|---|
| **RTT** | 20ms, 100ms, 200ms | $\geq 100ms$ |
| **bandwidth** | 1Mbps, 10Mbps | $\geq 10Mbps$ |
| **pkt loss** | 0, 0.005, 0.01, 0.02 | $\geq 0.01$ |
| **object size** | 100B, 1K, 10K, 100K, 1M | $\geq 1K$ |
| **# of object** | 2, 8, 16, 32, 64, 128, 512 | $\geq 64$ |

In this part, we introduce the infrastructural platforms and how we conduct our experiments as depicted in 1. In our experiments, we need the same server implementation to provide HTTPS and HTTP/2 stacks or else our results would be biased. We set up a Nginx proxy [4], which can transfer Web objects and pages through both HTTPS protocol and HTTP/2 protocol. For experiment with synthetic pages, we develop a client in Android platform to fetch Web objects instead of using real browsers. When considering dependencies between network activities and computations in real browsers, we use FireFox for Android platform, which supports both HTTPS and HTTP/2. Thus, we need to keep the consistency across experiments, so we clone the home pages of the Alexa top 200 websites into our local server.

**Server.** We use ThinkPad S3 as our server, which is a 64-bit machine with 1.9GHz 4 core CPU and 8GB memory and Ubuntu 14.04. We could switch the transfer protocol between HTTPS and HTTP/2 on Nginx server. We turn off gzip encoding of Nginx server to keep the exact size of Web objects and Web pages.

**Client.** We have mentioned we divide our experiments into two parts before. When we compare HTTP/2 and HTTPS as a transport protocol, we develop our own HTTP/2 and HTTPS client with OkHttp [6]. When considering the dependencies between network activities and computation in real browser, we load Web pages in Firefox 43.0 for Android. The mobile device is Samsung Galaxy Note 2 with 2 GB RAM and Android 4.4.4 OS.

**Settings.** In this paper, we consider five factors external to HTTP that may affect performance, including network conditions (e.g. packet loss, bandwidth, and RTT), and features of Web pages (e.g. object size, number of objects). Table I shows the factor settings in our experiments. We use *Traffic Control (TC)* of Linux kernel to set up various network conditions. The RTT values include 20ms (good WiFi), 100ms, and 200ms (3G). The bandwidth (bw) values include 1Mbps (3G) and 10Mbps (WiFi). We vary random packet loss rates from 0% to 2% [12]. We also consider a wide range of Web object size (100B-1MB) and number of objects (2-512) when we synthesize Web pages. We define a threshold for each factor, so that we can classify each setting as being high or low for our further analysis.

**Work flow.** First, we compare HTTPS and HTTP/2 as a transfer protocol, and analyze the effects of factors external to the protocols, including network conditions (e.g. packet loss, bandwidth, and RTT), and features of Web pages (e.g. object size, objects number). We synthesize Web pages with different sizes and numbers of Web objects. To experiment with real effect of browsers, we clone the home pages of the Alexa top 200 websites into our local server.

## IV. Performance Evaluation

### A. Experiment with Synthetic pages

When experimenting with synthetic pages, we consider a wide range of parameter settings as showed in Table I. We need to exclude the impact of complex process of loading a Web page, so we develop our own Android client based on OkHttp to fetch all the resources from these synthetic pages with pre-specified object size and number of objects. We switch transfer protocol of Nginx proxy between HTTPS and HTTP/2, so that our client can work with HTTPS and HTTP/2, respectively.

*1) Methodology:* Prior work [24] has analyzed when SPDY help or hurt, we wonder if HTTP/2 do have similiar features as SPDY since HTTP/2 derives from SPDY. To understand the conditions under which HTTP/2 helps or hurts, we try to build a predictive model based on decision tree analysis. In the analysis, each configuration is a combination of values for all factors listed in Table I. For each configuration, we add an additional variable *s*, which is the PLT of HTTP/2 divided by that of HTTPS. We run the decision tree with ID3 algorithm [3] to predict the factor settings under which HTTP/2 outperforms HTTPS ($s \leq 0.95$) and under which HTTPS outperforms HTTP/2 ($s \geq 1.05$). The ID3 algorithm begins with the original set *S* as the root node. On each iteration of the algorithm, it iterates through every unused factor (including RTT, bw, packet loss rate, object size, and object number in our data set) of the set *S* and calculates information gain *IG(A)* of that attribute. It then selects the attribute which has the largest information gain value. The set *S* is then split by the selected attribute to produce subsets of the data. The algorithm continues to recurse on each subset, considering only attributes never selected before. The decision tree analysis generates the likelihood that a configuration works better under HTTP/2 (or HTTPS). If this likelihood is over 0.75, we mark the branch as HTTP/2 (or HTTPS); otherwise, we say that HTTP/2 and HTTPs have comparable performances, and mark the branch as 'Other'.

[2]Page Load Time, https://developer.chrome.com/devtools/docs/network

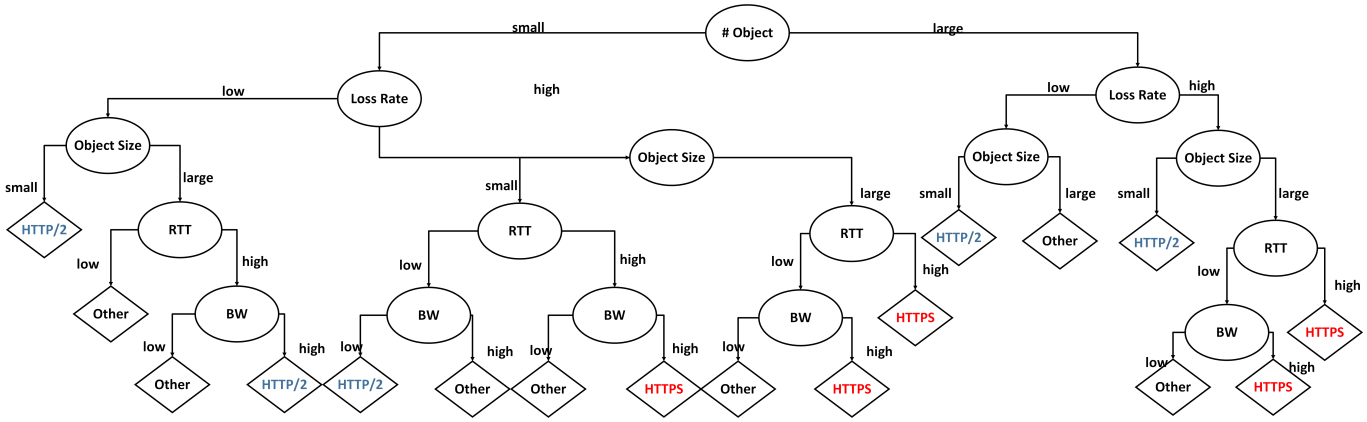



Figure 2: The decision tree that tells when HTTP/2 or HTTPS helps. A leaf pointing means HTTP/2 (HTTPS) helps; a leaf pointing to Other means performances of HTTP/2 and HTTPS are comparable.

*2) Result Analysis:* We first try to answer **RQ1: Can HTTP/2 help RESTful Web service performance on smartphones**. We find that HTTP/2 does improve performance under certain conditions, but hurts under other conditions.

Then, we have answer to **RQ2: How could those factors affect HTTP/2's performance**. The decision tree shows that HTTP/2 hurts when packet loss is high. However, HTTP/2 helps under a number of conditions, such as many small objects, small objects under low packet loss rate, and so on. The decision tree also depicts the relative importance of contributing factors. Intuitively, factors close to the root of the decision tree affect HTTP/2' performance more than those near the leaves. This is because the decision tree places the important factors near the root to reduce the number of branches. We find that object number, packet loss rate, and object size are the most important factors in predicting HTTP/2' performance. However, RTT and bandwidth play a less important role as shown in Figure 2.

Figure 3 depicts three performance trending graphs of HTTP/2 and HTTPS with a specific setting (rtt = 200ms, bw = 10Mbps, loss = 0, iw = 10, object_size = 10K, object_number = 64). Figure 3(a) shows that HTTP/2 performs better than HTTPS under such condition even if packet loss rate arise at a small range. Figure 3(b) shows that HTTP/2 performs better, but hurts when the number of objects arises to 512. Figure 3(c) depicts that HTTP/2 helps when object size is not too big, but hurts when object size arises to 1M.

Last, we move to **RQ3: Why does HTTP/2 help or hurt under certain conditions**. HTTP/2 helps when transferring many small objects. TCP implements congestion control by counting outstanding packets not bytes. Thus, sending a few small objects with HTTP will promptly use up the congestion window, though outstanding bytes are far below the window limit. In contrast, A single HTTP/2 connection can contain multiple concurrently-open streams, with either endpoint interleaving frames from multiple streams.

HTTP/2 performs better than HTTPS when RTT goes up. HTTP/2 benefits from having a single connection and stream multiplexing. One connection per origin significantly reduces the associated overhead: fewer sockets to manage along the connection path, smaller memory footprint, better connection throughput, less time in slow-start, faster congestion and loss recovery. As the RTT goes up, each HTTPS request cost more time on establishing connection with one RTT on TCP handshaking and two RTTs on negotiating SSL setup.

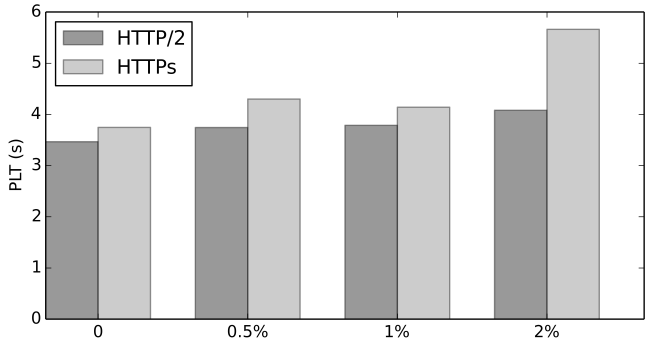
HTTP/2 is more vulnerable to packet loss because its feature of one TCP connection per origin. A single connection hurts under high packet loss because it aggressively reduces the congestion window compared to HTTPS which reduces the congestion window on only one of its parallel connections. When this single connection suffers from packet loss, all streams running over this unique TCP connection are negatively impacted.

### *B. Experiment with Real Web Pages*

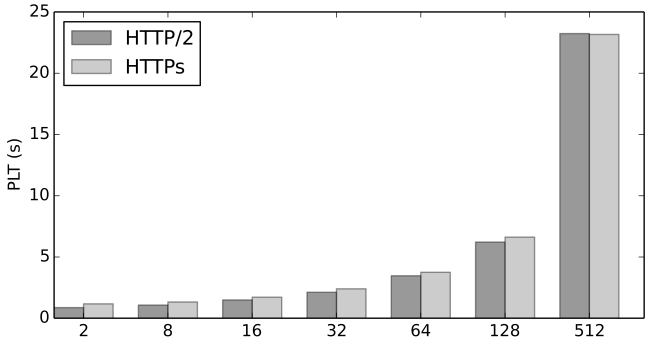
In this section, we compare the performance of HTTP/2 and HTTPS when loading real Web pages in mobile browsers, and try to answer **RQ4: How does HTTP/2 perform when loading real Web pages in mobile browser**. We clone Alexa Top 200 websites into our local server, and revisit these Web pages through HTTP/2 proxy and HTTPS proxy using Firefox for Android, respectively. We also vary other network settings including packet loss rate, bandwidth, and RTT to analyze the effect of network activities and computations in real browser.

We show the PLT of HTTP/2 and HTTPS of 20 test websites with both HTTP/2 and HTTPS in Figure 4. Since the page load time varies under different parameters settings, we use a box plot to present the results. For each test Web page, the left (red) box depicts the page load time with HTTP/2, while the right (blue) one depicts the page load time with HTTPS. Figure 4 shows that HTTP/2 performs better for some Web pages, while HTTPS wins for other pages.

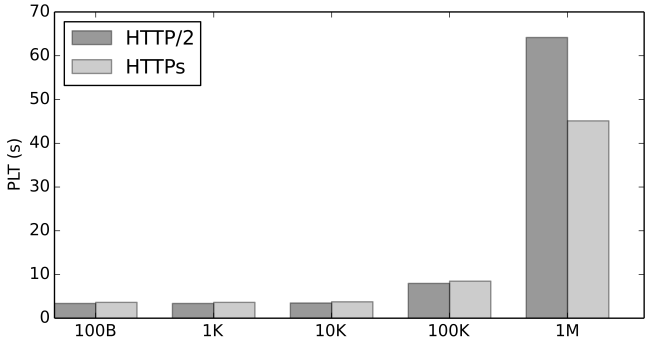
First, we characterize these real Web pages to further analyze why HTTP/2 helps or hurts. Figure 5 depicts the feature of the Alexa top 200 Web pages. These websites contain a wide range number of Web objects from 6 to 526. The smallest size of these Web pages is 16.35KB, but the largest one could be 21.5MB with lots of resources, including




(a) Packet loss rate




(b) # of objects




(c) Object size

Figure 3: Performance trends for three factors with a setting: rtt=200ms, bw=10Mbps, loss=0, obj size=10K, obj number=64.

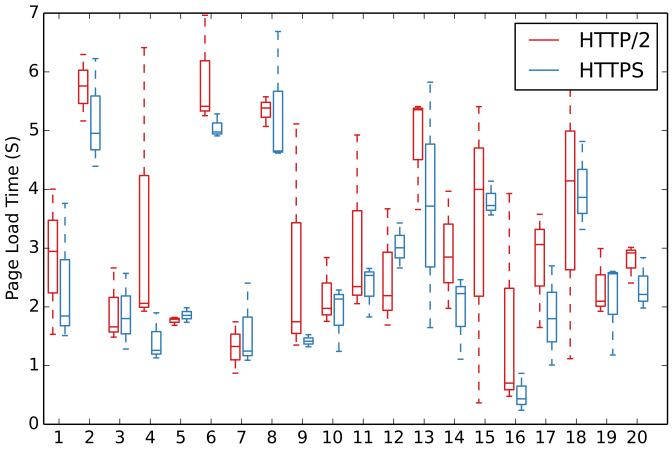
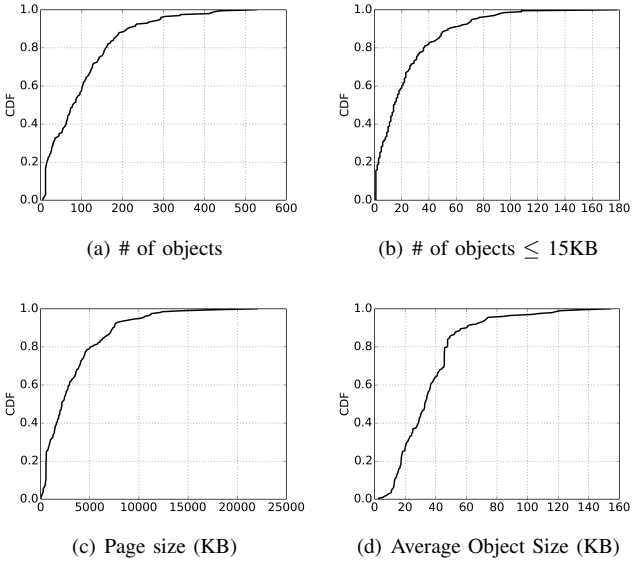



Figure 4: PLT of 20 test websites with both HTTP/2 and HTTPS




(a) # of objects

(b) # of objects $\leq$ 15KB

(c) Page size (KB)

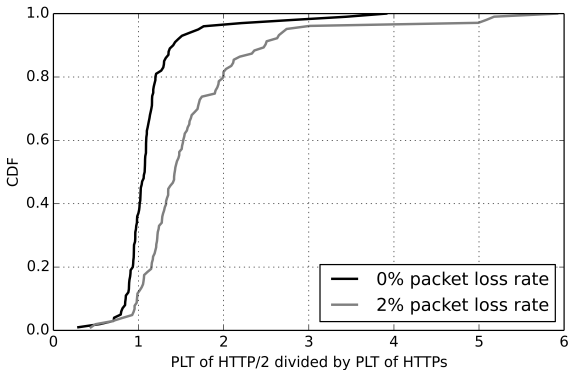
(d) Average Object Size (KB)

Figure 5: Features of top 200 Alexa Web pages

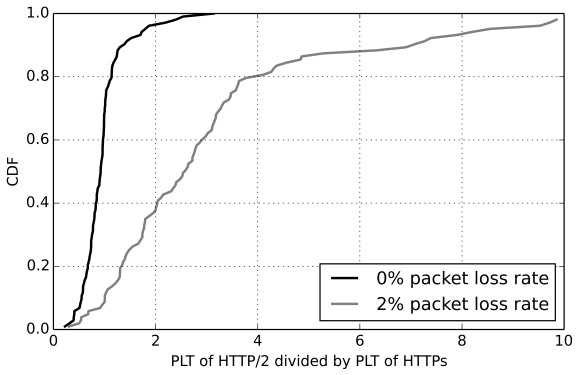
images, JavaScript files, CSS files etc. The median number of Web objects is 79, and the median Web page size is 2162KB. The median Web object size of all resources collected from all these Web pages is 8.22KB. We could find more Web objects and bigger page size as Web pages become more and more complex [24].

Overall, HTTP/2 helps 8.7% $\sim$ 68% of Web pages across different network conditions. HTTP/2 performs better under the condition of high latency and low packet loss. We could see that complex processes of real page loading in mobile browser, including network activities, computations, and their dependencies, do affect the performance improvement of HTTP/2. Due to the limited capabilities of computations, mobile browsers may spend more time on computation activities, including parsing HTML, parsing JavaScript/CSS, and interpreting JavaScript/CSS etc. For HTTP/2, the improvement of transferring Web objects could blur because of the above problems.

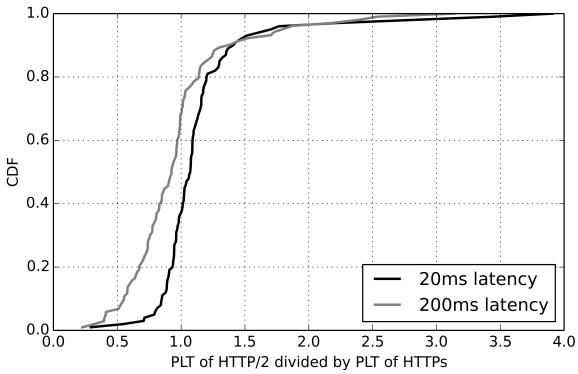
HTTP/2 helps likely due to the following reasons. First, HTTP/2 almost eliminates retransmissions. Second, HTTP/2 benefits from one connection per origin and stream multiplexing. As RTT grows up, it costs increasing time for HTTPS to establish connections for each resources, which costs one RTT on TCP handshaking and two on negotiating SSL setup. Third, we could find that more than 60% of the pages have large number of objects, and 40% of the pages have more than 20 small objects as shown in Figure 5(a) and Figure 5(b). Since HTTP/2 helps with small objects (based on the decision tree analysis) and acts well with large object number, it is




(a) RTT = 20ms, bw = 10Mbps




(b) RTT = 200ms, bw = 10Mbps




(c) packet loss rate = 0%, bw = 10Mbps

Figure 6: HTTP/2 performance across Alexa Top 200 pages.

not surprising that HTTP/2 has lower PLT for this set of experiments.

Figure 6 depicts the performance of HTTP/2 across Alexa Top 200 Web pages under conditions of 10Mbps bandwidth. We depict some cumulative distributions of performance under different network conditions. As shown in Figure 6(a) and Figure 6(b), we could see that HTTP/2 performs worse as the packet loss rate grows. We need to address the awkward situation of 2% packet loss rate, under which HTTP/2 is completely defeated by HTTPS as depicted in Figure 6(a)

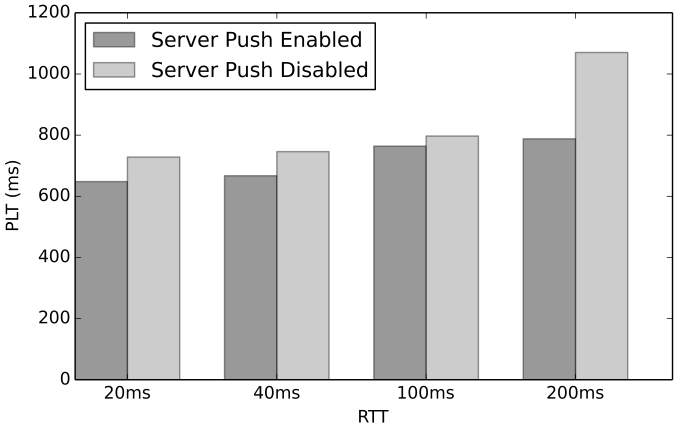



Figure 7: PLT with and without server push when using HTTP/2

and Figure 6(b). HTTP/2 opens only one TCP connection, under which high packet loss and retransmissions affect its throughput. However, we could find that HTTP/2 react more positively to high latency than HTTPS as shown in Figure 6(c). As RTT grows up, it costs increasing time for HTTPS to establish connections for each resources, while HTTP/2 benefits from single connection per origin.

### C. Server Push

Server push is a new feature of HTTP/2. When browser loads a Web page in the era of HTTP/1.x, it first initiates a request to fetch the main HTML page. Then, it will initiate consequential requests to fetch other resources, including JavaScript files, CSS files, images when parsing HTML. However, browsers have to initiate a new request for each resource with HTTP/1.x. The key idea of server push is that we could infer further needed resources when loading a page. In additional to the response for the original request, the server can push additional resources to the client.

We wonder if server push could improve the performance of page loading. Unfortunately, the Nginx server we used in our previous experiments does not support the new feature yet. We switch to *node-http2* [5] to test the new feature. We compare the page load time with and without server push when using HTTP/2.

We just synthesize a Web page consisting of a hundred of small pictures. When testing the effect of server push, we just implement a naive approach by pushing all pictures to browsers. Browsers need to issue extra a hundred of requests to fetch all the pictures without server push, but just one when server push is enabled. Despite our naive approach, we could see server push does improve the performance under different latencies as shown in Figure 7. We find an improvement 4.14% $\sim$ 26.35% in page load times. We also need to address that server push performs better under high latency. Server push helps because it could reduce the number of requests, and eliminate the latency from parsing HTML/JavaScript/CSS to recognizing new resource.

### D. Findings and implication

HTTP/2 attempts to overcome those shortcomings of HTTP/1.1, and speed the Web pages loading. As a transfer protocol, HTTP/2 does help under certain conditions, such as many small objects, low packet loss rate etc. However, we need address following problems:

- Modern Web pages are becoming more and more complex, containing more Web objects with growing sizes. Websites with large Web objects may have fewer benefits from HTTP/2.
- Loading a Web page is not as simple as fetching all resources in parallel. Loading a Web page in modern browsers is a complex work, including parsing HTML, parsing JavaScript/CSS, and interpreting JavaScript/CSS etc. The complicated dependencies between network activities and computation may blur the bright spot of HTTP/2.
- Due to the limited resources of mobile devices, computation activities may dominate in page loading process. As the weight of network activities reduces, HTTP/2 may just help marginally.
- Mobile devices may suffer from poor network conditions with high packet loss, under which HTTP/2 does not work well. Many users are used to visiting Web services using mobile devices with 3G/4G network, which may not benefit from HTTP/2.
- Web developers have come up with many best practices like domain sharding, spriting, and inlining and concatenation of resources to speed HTTP(S). However, these optimization of HTTP(S) may be anti-patterns for HTTP/2. For example, bundling multiple resources into a single response was a critical optimization for HTTP/1.x. However, bundled resources may delay execution until the entire response is downloaded.

In our experiments, we do find that the benefits from HTTP/2 when loading real Web pages using mobile browsers, especially under certain network conditions. However, HTTP/2 hurts under other network conditions, and the performance could be affected by the characteristics of Web pages. We give some implications as shown in Table II. For website developers, they should adjust their expectations that it may not speed their Web pages. It may be more practical to optimize their websites to reduce Web object sizes, improve cache configures etc. Of course, we should not be too negative, since mobile devices are becoming more and more powerful and related technologies of HTTP/2 is advancing. If developers want to deploy HTTP/2, they may need to reconstruct their websites with abandon of those best practices for HTTP/1.x. Meanwhile, developers should pay close attention to new features of HTTP/2, including server push and priority. Server push could reduce the number of requests, and eliminate the latency from parsing HTML/JavaScript/CSS to recognizing new resources.

## V. Related Work

Web service publishers aim to provide competitive services, and utilize all kinds of technologies to improve services' performance. In this paper, we focus on how HTTP/2 could reduce users perceived latency, which is a important factor of QoS. HTTP/2 derives from SPDY and aims to improve the performance of HTTP, so we focus on prior studies on improvement of HTTP and works about HTTP/2 and SPDY in this section.

**Quality of Service (QoS):** QoS is always attractive research topic of Web service. Ma et al. [19] investigate how to measure Web service QoS precisely from both subjective and objective aspects. Nacer et al. [8] and Chen et al. [9] present service selection and service recommendation, respectively. Elshater et al. [15] utilizes design pattern to improve performance in terms of average response time and throughput. However, the performance of services could be easily affected by network conditions and complexity of services. It is important to speed the delivery of response content to services consumers more than services processing.

**HTTP:** Radhakrishnan et al. [20] describes the TCP Fast Open protocol, a new mechanism that enables data exchange during TCP's initial handshake. Flach et al. [17] presents the design of novel loss recovery mechanisms for TCP that judiciously use redundant transmissions to minimize timeout-driven recovery.

**SPDY:** Erman et al. [16] finds that SPDY performed poorly while interacting with radios due to a large body of unnecessary retransmissions. Wang et al. [24] swipe a complete parameter space including network parameters, TCP settings, and Web page characteristics to learn which factors affect the performance of SPDY. Thus, they also dive into the analysis of the effect of dependencies of network activities and computation activities in real browsers. However, we are more curious how HTTP/2 performs in mobile devices with limited capacities and poor network conditions. El-khatib et al. [13] [14] also analyze the effect of network and infrastructural variables on SPDY's performance. Prior works on SPDY do encourage us to reveal the mysterious mask of HTTP/2.

**HTTP/2:** Chowdhury et al. [10] focus on the energy efficiency of HTTP/2, and shows that HTTP/2 exhibits better performance as the RTT goes up. However, we are more curious if HTTP/2 could ireduce page load time. Hugues et al. [11] evaluate the influence of HTTP/2' new features on the Web performance. They notice generally speedier page load times with HTTP/2, thanks to the multiplexing and compression features enabled by the protocol. The limitation is that they only test 15 websites. We test the Alexa Top 200 websites in our experiment. Varvello et al. [22] presents a measurement platform to monitor both adoption and performance of HTTP/2. They find 13,000 websites already announcing H2 support, out of which 600 websites (mostly hosted by Google and Twitter) deliver content over HTTP/2. Although HTTP/2 announces to provide better performance, developers may still be in doubt.

## VI. Conclusion

In this paper, we have divided our experiments into two parts. First, we compared HTTP/2 and HTTPS as a transport protocol to identify the factors that may affect HTTP/2,

TABLE II: Summary of Findings and Implications

| Findings | Implications |
|---|---|
| HTTP/2 could both decrease and increase page load times under certain network conditions and page characteristics. For example, HTTP/2 helps Web pages with many Web objects when packet loss rate is low. | Developers should consider the adoption of HTTP/2 thoroughly. It will not speed their websites with few Web objects. It is necessary to characterize websites before turning to HTTP/2. |
| Servers could push objects to clients to save round trips with server push, which could also eliminate the latency from parsing HTML/JavaScript/CSS to recognizing new resource. | If necessary, developers could take full advantage of such new features of HTTP/2 to improve performances of Web pages. |
| Best practices like domain sharding, spriting, and inlining and concatenation of resources do help in the era of HTTP/1.x. However, those best practices may hurt when using HTTP/2. | If developers decide to turn to HTTP/2, they need to reconstruct their websites to eliminate effects of those development conventions in the era of HTTP/1.x. |

including RTT, bandwidth, loss rate, number of objects on a page, and objects sizes. We found that HTTP/2 hurt with high packet loss, but helped with many small objects. Considering loading real Web pages in mobile browser, we cloned the top 200 websites into our local server, and revisited them through HTTP/2-enabled proxy, and HTTPS-enabled proxy, respectively. The complex process of loading Web pages, including network activities, computations, and their dependencies, reduced the benefits from HTTP/2. HTTP/2 benefited from its use of a single TCP connection and multiplexed streams. However, the benefits from a single TCP connection per origin could be easily overwhelmed with high packet loss because all streams running over this unique TCP connection are negatively impacted. At last, we conducted a simple experiment to test the server push feature of HTTP/2, and we got a positive feedback that server push could help. For future work, we plan to conduct more in-depth experiments to analyze how HTTP/2 helps or hurts under certain conditions. Thus, we will test more features of HTTP/2 including priority and header compression, and their practical applications.

## ACKNOWLEDGEMENT

This work is supported by the High-Tech Research and Development Program of China under Grant No.2015AA01A202, the Natural Science Foundation of China (Grant No. 61370020, 61421091,61528201).

## REFERENCES

[1] Head of line blocking. https://en.wikipedia.org/wiki/Head-of-line_blocking.
[2] HTTP archive trend. http://httparchive.org/trends.php.
[3] ID3 algorithm. https://en.wikipedia.org/wiki/ID3_algorithm.
[4] Nginx, an HTTP and reverse proxy server. http://nginx.org/.
[5] node-http2: a node module of HTTP/2. https://github.com/molnarg/node-http2.
[6] Okhttp. http://square.github.io/okhttp/.
[7] Why revise HTTP. http://http2.github.io/faq/#why-revise-http.
[8] A. Ahmed-Nacer, K. Bessai, S. Youcef, and C. Godart. A multi-criteria based approach for web service selection using quality of service (qos). In *Proceedings of the 2015 IEEE International Conference on Services Computing, SCC 2015*, pages 570–577, 2015.
[9] M. Chen, Y. Ma, B. Hu, and L. Zhang. A ranking-oriented hybrid approach to qos-aware web service recommendation. In *Proceedings of the 2015 IEEE International Conference on Services Computing, SCC 2015*, pages 578–585, 2015.
[10] S. A. Chowdhury, V. Sapra, and A. Hindle. Is HTTP/2 more energy efficient than HTTP/1.1 for mobile users? *PeerJ PrePrints*, 3:e1280, 2015.
[11] H. de Saxcé, I. Oprescu, and Y. Chen. Is HTTP/2 really faster than HTTP/1.1? In *Proceedings of the 2015 IEEE Conference on Computer Communications Workshops, INFOCOM Workshops 2015*, pages 293–299, 2015.
[12] N. Dukkipati, M. Mathis, Y. Cheng, and M. Ghobadi. Proportional rate reduction for TCP. In *Proceedings of the 11th ACM SIGCOMM Internet Measurement Conference, IMC 2011*, pages 155–170, 2011.
[13] Y. El-khatib, G. Tyson, and M. Welzl. Can SPDY really make the web faster? In *Proceedings of the 2014 IFIP Networking Conference, IFIP 2014*, pages 1–9, 2014.
[14] Y. El-khatib, G. Tyson, and M. Welzl. The effect of network and infrastructural variables on SPDY's performance. *CoRR*, abs/1401.6508, 2014.
[15] Y. Elshater, P. Martin, and E. Hassanein. Using design patterns to improve web service performance. In *Proceedings of the 2015 IEEE International Conference on Services Computing, SCC 2015*, pages 746–749, 2015.
[16] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan. Towards a SPDY'ier mobile web? In *Proceedings of the conference on emerging Networking Experiments and Technologies, CoNEXT 2013*, pages 303–314, 2013.
[17] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan. Reducing web latency: the virtue of gentle aggression. In *Proceedings of the ACM SIGCOMM 2013 Conference, SIGCOMM 2013*, pages 159–170, 2013.
[18] Y. Liu, X. Liu, Y. Ma, Y. Liu, Z. Zheng, G. Huang, and M. B. Blake. Characterizing restful web services usage on smartphones: A tale of native apps and web apps. In *Proceedings of 2015 IEEE International Conference on Web Services, ICWS 2015*, pages 337–344, 2015.
[19] Y. Ma, S. Wang, Q. Sun, H. Zou, and F. Yang. Web services qos measure based on subjective and objective weight. In *Proceedings of the 2013 IEEE International Conference on Services Computing, SCC 2013*, pages 543–550, 2013.
[20] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan. TCP fast open. In *Proceedings of the 2011 Conference on Emerging Networking Experiments and Technologies, Co-NEXT 2011*, page 21, 2011.
[21] D. Stenberg. HTTP2 explained. *Computer Communication Review*, 44(3):120–128, 2014.
[22] M. Varvello, K. Schomp, D. Naylor, J. Blackburn, A. Finamore, and K. Papagiannaki. To HTTP/2, or not to HTTP/2, that is the question. *CoRR*, abs/1507.06562, 2015.
[23] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. Demystifying page load performance with wprof. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013*, pages 473–485, 2013.
[24] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. How speedy is SPDY? In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014*, pages 387–399, 2014.