DOI: 10.3969/j. issn. 1673-5188. 2016. S1. 006
http://www.cnki.net/kcms/detail/34.1294.TN.20161228.1105.002.html, published online December 28, 2016

An Optimization of HTTP/2 for Mobile Applications

DONG Zhenjiang[1], SHUANG Kai[2], CAI Yanan[2], WANG Wei[1], and LI Congbing[1]

(1. Cloud Computing & IT Research Institute, ZTE Corporation, Nanjing 210012, China;
2. State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract

In recent years, Hyper Text Transfer Protocol (HTTP) spreads quickly and steadily in the usage of mobile applications as a common web protocol, so that the mobile applications can also benefit from HTTP/2, which is the new version of HTTP based on SPDY developed by Google to speed up the Internet transmission speed. HTTP/2 enables a more efficient use of network resources and a reduced perception of latency by introducing header field compression and allowing multiple concurrent exchanges on the same connection. However, what HTTP/2 focuses on is visiting websites through a browser, and mobile applications are not considered much. In this paper, firstly, mobile applications are classified based on the data flow characteristics. Based on the classification, we propose an optimization of HTTP/2 for mobile applications, called HTTP/2-Advance, which uses multiple Transmission Control Protocol (TCP) connections to multiplex HTTP requests and responses. Then we build a tiny system which simulates actual requests and responses between mobile applications and servers. We figure out the best choice of the number of multiple TCP connections for mobile applications, and compare the performance of HTTP, HTTP/2 and HTTP/2-Advance in both simulated and in-situ experiments in our system.

Keywords

HTTP/2; HTTP optimization; multiple connection; header compression

This work was supported in part by ZTE Industry-Academia-Research Cooperation Funds.

1 Introduction

In recent years, with the growing popularity of smart phones and the rapid prosperous development of mobile Internet, mobile applications present an explosive growth. According to the data provided by Google Play Store, there were more than 370,000 applications available for the Android pads and phones in 2011, and the number rose sharply to 1.5 million in 2014 [1]. Meanwhile, because of an increasing trend on binding mobile applications with rendering engine and a widely acceptance on emerging frameworks like Hybrid [2], Hyper Text Transfer Protocol (HTTP) spreads quickly and steadily in the usage of mobile applications as a common web protocol. However, since the Internet has changed dramatically since 1990 when HTTP was first published, HTTP cannot hold the needs of users gradually.

The optimization of HTTP is imminent. HTTP 1.0 serially builds on Transmission Control Protocol (TCP) connection for each embedded objectives. When requesting a web page with a browser, the TCP 3-way/4-way handshake and slow start significantly increase the latency. To optimize this issue, HTTP1.1 uses multiple, persistent TCP connections to "keep-alive". However, it is still a serial prototype in essence and the multiple connections strategy may cause more bandwidth usage. HTTP/2 [3], [4] is the second major version of HTTP to make web faster. It is based on SPDY [5], which is developed by Google. According to W3Techs, 1.2% of all websites support HTTP/2 in August 2015 to speed up the Internet transmission speed [6].

HTTP/2 enables a more efficient use of network resources and a reduced perception of latency by introducing header field compression and allowing multiple concurrent exchanges on the same connection [3]. This means less competition with other flows and longer-lived connections, which in turn leads to better utilization of available network capacity. As a result, the web browser loads pages more quickly. However, for the mobile, it is much different. The browser is just a very small part of the mobile applications that use HTTP. Besides, the unpredictable network circumstance and data flow characteristics brought about by different applications have great impact on the Internet transmission speed. For example, applications like UC browser have intermittent network connections with diverse connection time and data flow throughout the day depending on the usage patterns, while other applications such as Sina Weibo have persistent network connections to transfer the signaling and user requested resources with light data flow. In this paper, we proposed an optimization of HTTP/2 for the mobile applications, which enables a more efficient way to use HTTP/2 in mobile application scenarios. The main contributions of this paper are: 1) Mobile applications are classified based on the data flow characteristics; 2) an optimization of HTTP/2 for mobile applications, called HTTP/2-Advance, is proposed; 3) a tiny system is built to simulate the actual re-

quests and responses between applications and servers, and 4) the performance of this optimization is tested and analyzed in different categories of applications and different network conditions in both simulated and in-situ experiments.

2 Background and Related Works

2.1 Background

HTTP/2 allows the interleaving of request and response message exchanges on the same connection associated with its own stream. The streams are largely independent of each other; in this way, a blocked or stalled request or response does not prevent any progress on other streams. It can reuse the TCP connections to reduce the number of TCP connections and the latency.

An HTTP header has some same information for a single session, for instance, host and user-agent. HTTP/2 provides the header compression strategy to reduce the unnecessary redundant information, and then decrease the TCP bytes and application layer latency. It is preferable to use no more than two levels of subheadings (primary and secondary), and the levels should be carefully differentiated. Ideally we try not to go below a second level subheading. It should be noted that there is no period after the final number.

2.2 Related Works

Some works have been done in PC since SPDY (which HTTP/2 is based on) was published. The performance of SPDY and HTTP in a variety of settings in publicly available software was tested in [7]. Web page load time under SPDY and HTTP was systematically compared in [8]. A comprehensive evaluation of SPDY's performance was experimented in [9] to find the potential benefits of SPDY. However, all of them focus on the PC side, and are limited by the existing SPDY deployments provided by some famous web service providers.

Some works have been done in the mobile side. J. Khalid et al. [10] proposed two adopting mechanisms to dynamically adjust the overall performance. However, this solution did not evaluate them in the mobile circumstance. J. Erman et al. [11] provided a detailed analysis of the performance of both HTTP and SPDY, and of their interaction with various layers as well. However, only the web pages on mobile browsers in cellular networks have been analyzed. G. Mineki et al. [12] proposed a SPDY accelerator that could considerably accelerate the web access speed by combining the SPDY protocol and cache system, but it only evaluates the performance of web pages, too. So far, none of the current works have examined the performance of SPDY on mobile applications.

In this paper, an optimization of HTTP/2 for mobile applications is proposed, which is called HTTP/2-Advance. Both the simulated experiment and in-situ experiment are also conducted to compare the performance of HTTP, HTTP/2 and HTTP/2-Advance.

3 Application Taxonomy

The taxonomy is based on the data flow characteristics of an application during its usage time. According to the Android operating system, only one app can be shown to the user on the screen, while other applications are only allowed to run in background. The reason we consider app usage time rather than user behavior of multiple applications is that every factor of the user behavior towards multiple applications, such as location, time, user needs, and certain context can be divided into the data flow characteristic of application and the network condition during the app usage time. The data flow characteristic will be introduced in this section, and the network condition influence will be discussed in next section.

Mobile applications are divided into four categories in this paper, and for each category, one Chinese popular app is selected as the test object. The categories are shown in **Table 1**.

The data flow characteristics of these applications are shown as follows.

3.1 Intermittent Connection with Random Data Flow

The application has intermittent connections with diverse connection time, and data flow throughout the day is difficult to predict since it largely depends on the usage pattern. The mobile browser UC Browser is one representative that occupies 41.7% of the Chinese mobile browser market [17].

3.2 Intermittent Connection with Light Data Flow

The application uses intermittent connections to download the resources with diverse connection time. Data flow is generated only when the application is downloading resources. After downloading, the application hardly consumes network resources. One example is Southern Weekly, an online application made by a traditional media company.

3.3 Persistent Connection with Heavy Data Flow

Xunlei Kankan is a popular online video application. Its data flow is heavy, and may occupy the independent channel resources of persistent connections for a long time.

3.4 Persistent Connection with Light Data Flow

The application has persistent network connections to transfer constantly switching of state signaling and download user

▼**Table 1. Application taxonomy based on data flow characteristic**

Application taxonomy	Popular application
Intermittent connection with random data flow	UC Browser [13]
Intermittent connection with light data flow	Southern Weekly [14]
Persistent connection with heavy data flow	Xunlei Kankan [15]
Persistent connection with light data flow	Sina Weibo [16]

requested resources, but the data flow is light. Sina Weibo, the most popular social network app in China, is such an application.

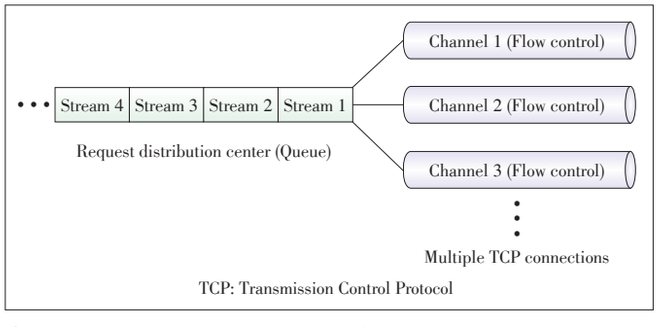
4 Optimization of HTTP/2

HTTP/2 allows interleaving of request and response messages exchange on the same connection associated with its own stream. The client sends an HTTP request on a new stream using a previously unused stream identifier. The server sends an HTTP response on the same stream as the request [6]. HTTP/2 uses multiplexing connection, and since using a single connection, it does not monopolize network resources. This means less competition with other flows and longer-lived connections, which in turn leads to better utilization of available network capacity. As a result, the web browser will load pages more quickly.

However, what this strategy focuses on is visiting websites through a PC browser, and mobile applications are not considered. According to Android operating system, only one application can be shown to the user on the screen, so it is reasonable to assume that there is only one application being able to access to network at the same time. Different from the PC scenario, the network resources can be monopolized by one mobile application. In this case, only using a single TCP connection to interleave HTTP requests and responses is not the best choice, because it cannot make full use of network capacity. Therefore, we propose an optimization strategy of HTTP/2 called HTTP/2-Advance, which uses multiple TCP connections to multiplex HTTP requests and responses. That means that the web elements are loaded in parallel over several TCP connections. The multiplexing strategy (**Fig. 1**) is as follows:

1) The multiple TCP connections between the client and the server are declared as Channel 1, Channel 2, … , Channel *N*, and these channels are indiscriminate with each other from the HTTP request-response stream perspective.
2) Each HTTP request-response stream is allocated a unique stream ID and put into the request distribution center, which is an orderly queue in essence, by the generation time of each request.
3) The request distribution center distributes the HTTP request-response streams to the multiple TCP connection channels based on the round-robin strategy. Meanwhile, a flow-control [3] scheme ensures that the streams on the same connection do not destructively interfere with each other so that blocked streams are prevented.

The multiple TCP connections provide a parallel data transmission channel, but simultaneously bring more bandwidth consumption and more HTTP/2 multiplexing connection processing time when using our optimization. This is not a "the more, the better" scenario and there should be a tradeoff for each application category. If the application just has light data flow with intermittent connections to transfer, the benefit brought from multiple TCP connections may not cover the process time for that. On the other hand, if the data flow is heavy, the multiple TCP connections benefit the transmission speed more. To figure out how many multiple TCP connections are best for each application category, some experiments have been done, which will be introduced in the next section.

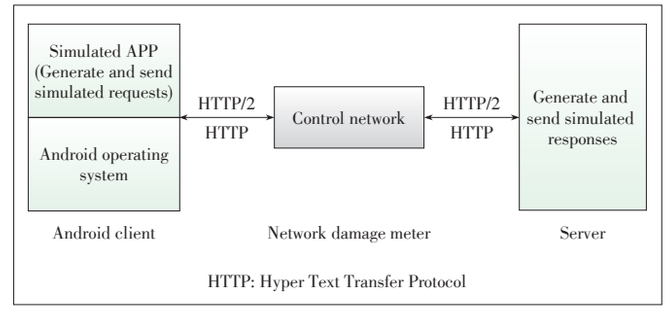


▲**Figure 1. The multiplexing strategy of HTTP-Advance. The HTTP request-response streams are put into the request distribution center by the sequence of requesting time, and distributed to the multiple channels based on round-robin.**

5 Experiments and Evaluation

5.1 Experimental Setup

Since HTTP/2 is a new protocol, almost none of the mobile applications have been built with a HTTP/2 protocol stack, as well as the application servers in China. To simulate actual requests and responses between the applications and servers, we build a tiny system. This system is made up of an Android client with a simulated application, a server, and a network damage meter [18] between the client and the server. The simulated application can generate and send both simulated HTTP and HTTP/2 requests. The server is used to generate and send simulated responses for both HTTP and HTTP/2 requests in each protocol stack. The number of multiplexing connections are alternative between the client and the server. The network damage meter controls the network condition, such like the latency and the packet loss rate. The structure of the system is shown in **Fig. 2**. The Android phone used as the client is Nex-



▲**Figure 2. The simulation system architecture. The client is connected to the server via a network damage meter that controls the network condition.**

us 5 with a 2.3 GHz CPU, 2G RAM, and Android 4.4 OS. The server is a Dell T5810 computer with 3.5 GHz CPU and 32G RAM.

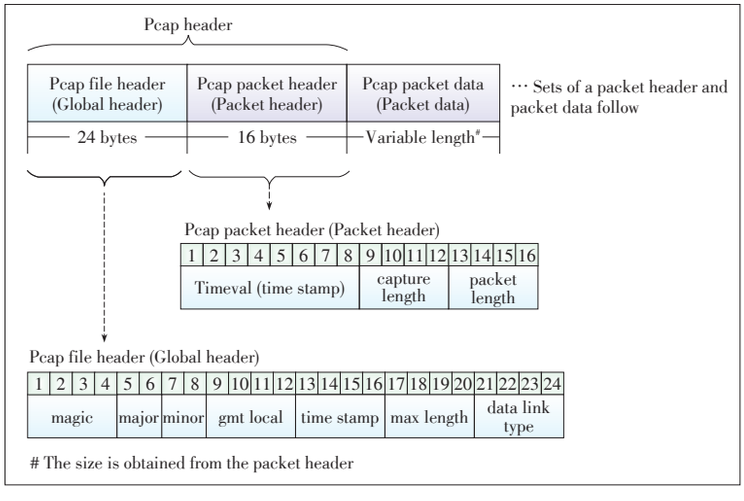
For rigorous controlled trials, simulated user requests based on the actual requests are generated and sent from the Android client to the server many times to compare the performance of HTTP, HTTP/2 and HTTP/2-Advance statistically. Simulated responses based on the actual responses are generated by the server to respond with the client requests. At the same time, network condition is controlled by the parameter configuration on the network damage meter. In order to produce authentic simulated requests and responses:

Firstly, for each application category, application requests are classified into several behavior categories based on our user research, and the packets of actual requests and responses of each behavior category are captured by the Shark [19]. The Shark then generates .pacp files (**Fig. 3**). The pcap packet data stores the captured packet, and pcap packet header stores the time stamp and length of the captured packet. By analyzing the pcap files, we get the time stamp and length from the pcap packet header, and requests and responses information from pcap packet data.

Secondly, the behavior configuration files are generated by analyzing the pcap files of each behavior category. By parsing the pacp file, each realistic request-response is analyzed to get the data of index, request time, dependent resources, GET/POST, URL, header length and body length. Based on these data, the simulated application and server can simulate the actual requests and responses.

Thirdly, the behavior configuration files are grouped to generate the group file. The grouping is based on our user research of user habits when using the application.

Lastly, the simulated application and server send requests and responses based on the group file and the behavior configuration files to guarantee the similarity of the traffic characteristic between the simulated packets and the actual ones.



▲**Figure 3. Pcap file format. Pcap packet data stores the captured packet, and the pcap packet header stores the time stamp and length of the captured packet.**

With the example of Sina Weibo, requests are firstly classified into five behavior categories, that is, login, refresh, read, post, and repost/comment, and actual requests and responses packets of each behavior are then captured by the Shark. Secondly, the behavior configuration files are generated as login file, refresh file, read file, post file, and repost/comment file. Thirdly, based on the research of user habits when using Sina Weibo, the group file is generated by grouping the behavior configuration files. Lastly, the simulated packets based on the group and configuration files are sent by the client and the server.

The network condition is parameterized by the latency and the packet loss rate. The latency and packet loss rate parameters based on the mobile access network like 2G, 3G, 4G and Wi-Fi/802.11 are considered to simulate different network conditions [20]. They are parameterized as (0 ms, 0%), (100 ms, 0%), (300 ms, 0%), (0 ms, 2%), (100 ms, 2%), and (300 ms, 2%). At the same time, the application layer latency $L_{app\ layer}$ in (1), application layer throughput $TP_{app\ layer}$ in (2) and (3), TCP total bytes B_{TCP}, and the total number of TCP connections $\#_{TCP\ conn}$ are recorded to evaluate the performance of HTTP, HTTP/2 and HTTP/2-Advance in different scenarios. The calculation methods of the comparison standards are as follows:

$$L_{app\ layer} = t_{last\ 200\ ok} - t_{1st\ GET/POST} \quad (1)$$

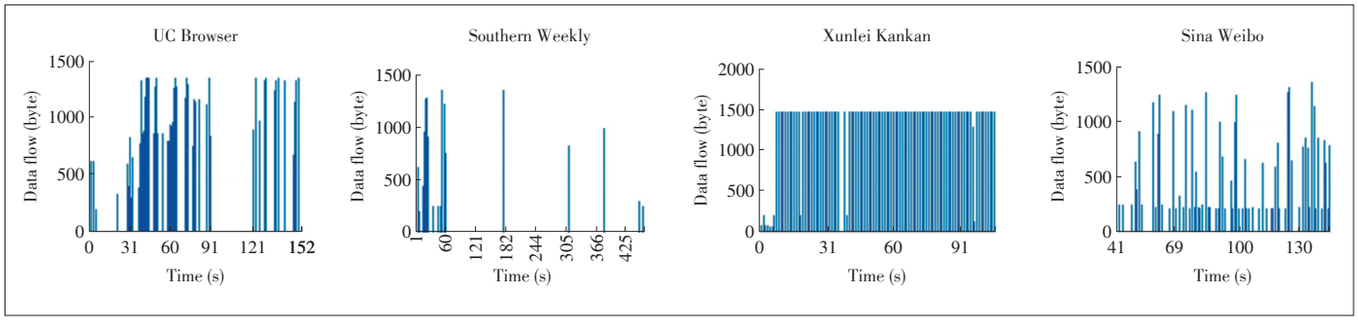
$$TP_{app\ layer} = B_{TCP\ payload} / L_{app\ layer} \quad (2)$$

$$B_{TCP\ payload} = B_{TCP} - B_{TCP\ header} - B_{conn\ control} \quad (3)$$

Since HTTP and HTTP/2 are application layer protocols, the application layer latency $L_{app\ layer}$ and the application layer throughput $TP_{app\ layer}$ are the most important indicators to measure the optimization proportion. The application layer latency is a main standard of user experience, and the application layer throughput reflects network efficiency and usage. The application layer latency $L_{app\ layer}$ equals to the HTTP session duration, from the first sent GET/POST request timestamp $t_{1st\ GET/POST}$ to the last received 200 OK response timestamp $t_{last\ 200\ ok}$. The application layer throughput $TP_{app\ layer}$ equals to the TCP payload bytes $B_{TCP\ payload}$ divided by the application layer latency $L_{app\ layer}$. The TCP payload bytes $B_{TCP\ payload}$ as in (3), equals to the TCP total bytes B_{TCP} decrement the TCP header bytes $B_{TCP\ header}$ and the TCP connection establishment/close control bytes $B_{conn\ control}$ (such as SYN+ACK).

5.2 Data Flow Characteristics of Application Categories

As described in Section 3, applications are divided into four categories based on the data flow characteristic. Before the test, data flow characteristics of each applica-



▲Figure 4. Data flow characteristics of application categories. Different application categories have different data flow characteristics, which influences the performance.

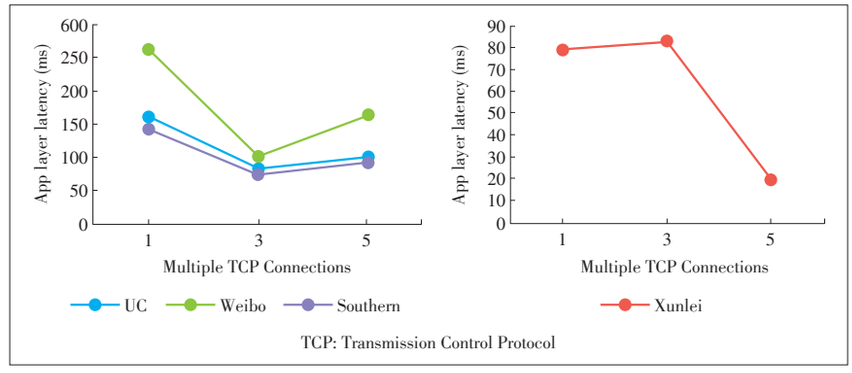
tion category are measured to have more in-depth understanding of the application category. The characteristics are shown in **Fig. 4**, from which we can find that, UC Browser has intermittent network connections with diverse connection time, and the data flow is random. Southern Weekly uses intermittent connections to download with diverse connection time and light data flow. Xunlei Kankan has persistent connection with heavy data flow all the time, and Sina Weibo has persistent connection like Xunlei Kankan, but the data flow is light.

We also find that most of these applications in China use HTTP1.0 (short connection) to request the resources. For every HTTP request-response, the applications serially established multiple TCP connections to improve the resources loading time. Therefore, in our later simulation and experiment, HTTP1.0 (short connection) will be used to evaluate the performance of HTTP, HTTP/2 and HTTP/2-Advance.

5.3 Simulation Results

5.3.1 Number of Connections

The performance of HTTP/2-Advance, using multiple TCP connections to multiplex HTTP requests and responses, is tested and analyzed to figure out how many multiple TCP connections are best for each application category. Application layer latency are measured in single connection, three multiple TCP connections, and five multiple TCP connections (**Fig. 5**). UC Browser and Southern Weekly donnot always have a packet to transfer, and the packets are not big since the web page optimization for mobile phone displaying. Sina Weibo has constantly state switch signaling and heartbeat packet, but the packet is also small. UC Browser, Sina Weibo and Southern Weekly perform best at three multiple TCP connections. Xunlei Kankan has very different data flow characteristics from others. Because its data flow is always heavy and occupying the independent channel resources for a long time, it is not strange that the latency is drastically reduced when three multiple TCP connections is changed into five. In a total, the HTTP/2-Advance enables a more efficient way to use HTTP/2 in mobile applications, and multiplexing three multiple TCP connections is feasible in most cases.



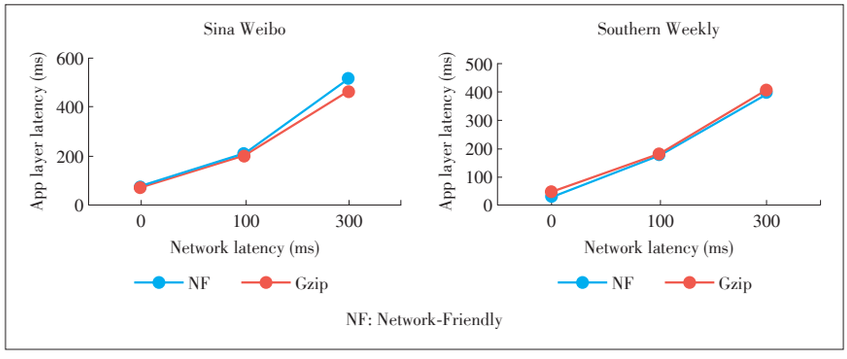
▲Figure 5. The number of multiple TCP connections for each category. More connections increase connection bandwidth and processing time, and a tradeoff is necessary when using HTTP/2-Advance.

5.3.2 Header Compression Strategy

The header compression strategy can reduce redundant information in the HTTP header and further the TCP bytes and application layer latency. The application layer latency is measured in two compression strategies that are Network-Friendly [21] and Gzip [22]. As shown in **Fig. 6**, the difference of header compression strategies has little influence on the application layer latency. The HTTP header occupies a major part of the request packet, but in the response packet, the HTTP header is usually much smaller than HTTP body content. Therefore, in a pair of request-response, the header size of the whole HTTP packets is so small that different header compression strategies have little influence on latency. We finally choose Gzip for our later evaluation.

5.3.3 Evaluation of HTTP/2-Advance

The performance of HTTP, HTTP/2 and HTTP/2-Advance in different network condition are tested and compared with three TCP connections as 3 and Gzip header compression. The results are shown in **Figs. 7** and **8**.



▲Figure 6. Application layer latency in different compression strategy. Difference of compression strategy has little influence on app layer latency.

HTTP/2-Advance performs better than HTTP/2, especially when the network condition is good. The average HTTP/2-Advance optimization proportion of HTTP on application layer latency is 41.67%. The network resources can be better used by multiple TCP connections when the network condition is good, however, these multiple TCP connections will be in conflict while the network condition becomes bad. Even though the advantage of HTTP/2-Advance decreases, HTTP/2-Advance still preforms better.

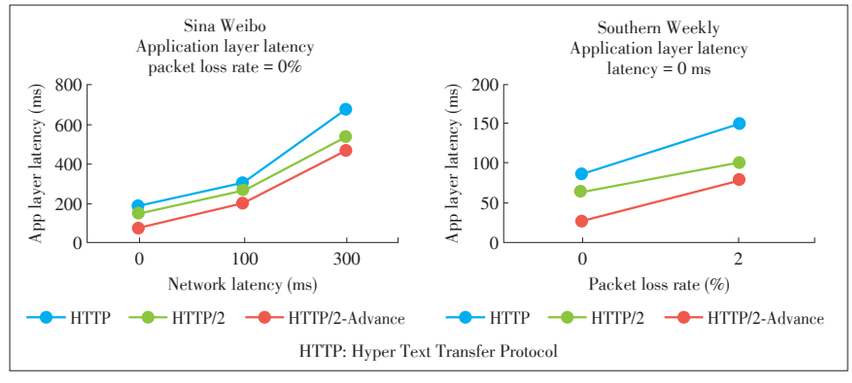
The application layer latency increases when the latency or the packet loss rate increases no matter HTTP, HTTP/2 or HTTP/2-Advance is used. The network latency directly affects the data transmission latency. A larger latency will increase the data transferring time and then increase the application layer request completion time with other conditions remaining unchanged. If latency is a fixed number, a higher packet loss rate stands a higher occurrence of incomplete data, and this relation also exists in retransmission scenario that will cause the increase of the application layer latency.

The application layer throughput declines when latency increases. As $TP_{app\ layer}$ and $B_{TCP\ payload}$ hardly change when using HTTP, HTTP/2 or HTTP/2-Advance, it is obvious that the application layer throughput only depends on the application layer latency, and is inversely proportional to the application layer latency in theory (Fig. 8).

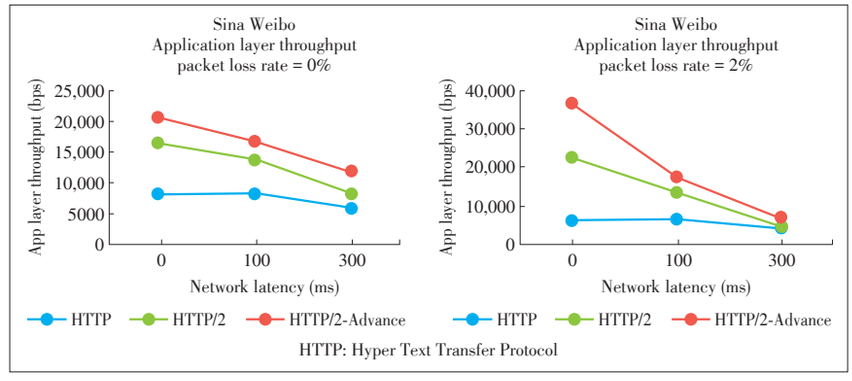
The main reason of the reduced application layer latency using HTTP/2-Advance is that HTTP/2-Advance uses multiplexing connection, which reduces the total number of TCP connections and the cost of TCP connection establishment/close 3-way/4-way handshakes. During the test, the number of total TCP connections and the TCP bytes are recorded (**Tables 2** and **3**). The average reducing bytes of Southern Weekly is 800.44 bytes per connection, and that of UC Browser is 663.92 bytes per connection. Due to the multiplexing connection strategy, the HTTP packet can reuse the previous established TCP connections. As we all know, the establishment of a TCP connection needs three-way handshake, and the close of the connection needs four-way handshake. Theoretically, the connection establishment/close control bytes $B_{conn\ control}$ that is required by TCP establishment and close handshakes are in (4), where n means the total number of TCP connections $\#_{TCP\ conn}$.

$$B_{conn\ control}=\sum_{i=1}^{n}\left(\sum_{j=1}^{3}establish_{i,j}+\sum_{k=1}^{4}close_{i,k}\right)\approx (74+74+66+66+66+60+60)*n= =466n\left(n=\#_{TCPconn}\right) \quad (4)$$

The actual data shown in Tables 2 and 3 are consistent with the magnitude of the theoretical data. Due to the unpredictable factors such as retransmission and HTTP/2 header compression strategy, a little more bytes reduction than the theoretical



▲Figure 7. HTTP/2-Advance preforms better than HTTP/2 and HTTP according to the application layer latency.



▲Figure 8. HTTP/2 and HTTP/2-Advance are both optimize the app layer throughput, and the app layer throughput is inversely proportional to the app layer latency in theory.

▼Table 2. TCP bytes and the total number of TCP connections—Southern Weekly

	Latency	0 ms		100 ms		300 ms	
Packet loss rate is 0%	Protocol	HTTP	HTTP/2-Advance	HTTP	HTTP/2-Advance	HTTP	HTTP/2-Advance
	B_{TCP} (bytes)	186,650	115,712	167,230	100,034	269,855	138,575
	$\#_{TCP\ conn}$	120	3	105	4	146	3
Packet loss rate is 2%	Protocol	HTTP	HTTP/2-Advance	HTTP	HTTP/2-Advance	HTTP	HTTP/2-Advance
	B_{TCP} (bytes)	224,105	144,908	215,676	107,392	280,634	152,482
	$\#_{TCP\ conn}$	121	3	106	3	146	3

HTTP: Hyper Text Transfer Protocol TCP: Transmission Control Protocol

▼Table 3. TCP bytes and the total number of TCP connections—UC Browser

	Latency	0 ms		100 ms		300 ms	
Packet loss rate is 0%	Protocol	HTTP	HTTP/2-Advance	HTTP	HTTP/2-Advance	HTTP	HTTP/2-Advance
	B_{TCP} (bytes)	260,353	165,797	322,514	194,722	292,336	172,530
	$\#_{TCP\ conn}$	186	3	220	4	192	4
Packet loss rate is 2%	Protocol	HTTP	HTTP/2-Advance	HTTP	HTTP/2-Advance	HTTP	HTTP/2-Advance
	B_{TCP} (bytes)	293,155	181,261	365,463	194,722	328,536	181,378
	$\#_{TCP\ conn}$	187	3	219	4	192	3

HTTP: Hyper Text Transfer Protocol TCP: Transmission Control Protocol

data is understandable. Furthermore, besides Southern Weekly and UC Browser, the other application categories also show the same relation.

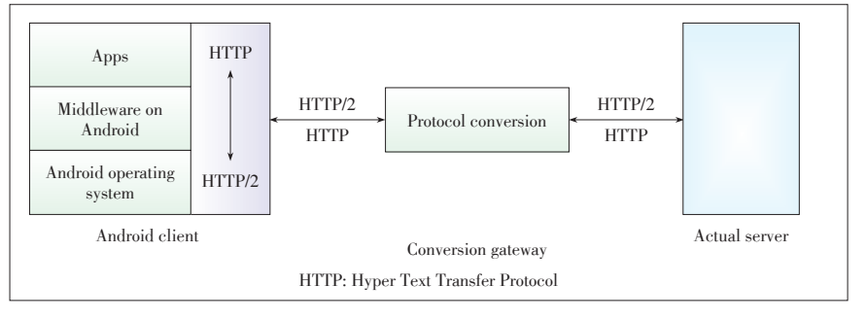
In a total, HTTP/2-Advance performs better than HTTP/2, especially when the network condition is good. HTTP/2-Advance optimizes the application layer latency and throughput and it does reduce the TCP connections and TCP bytes thanks to the multiplexing connection and the header compression strategy.

5.4 In-Situ Experiment

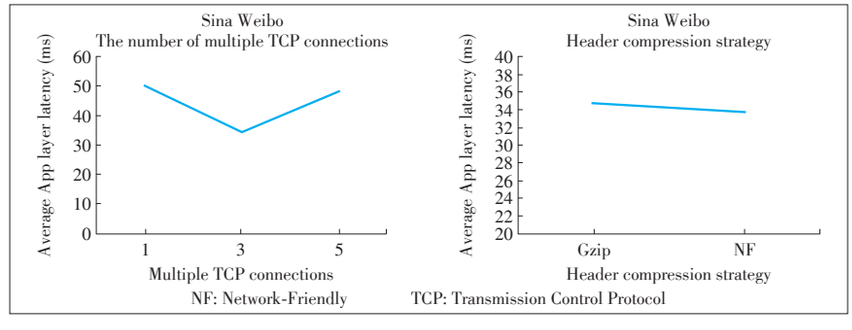
A tiny system is built to test the performance of HTTP-Advance in actual situation (**Fig. 9**). An Android middleware is developed for changing HTTP stack to HTTP/2-Advance stack on the client side. The middleware runs "under" the applications and allows the real applications to send HTTP/2-Advance packet indirectly. Placed in between the actual server and client, a protocol conversion gateway is used to convert the protocol HTTP and HTTP/2-Advance. The server is an actual application server that responds requests from the client. The tests of Sina Weibo and UC Browser were done by 20 persons at different time and different locations. The test results are shown in **Fig. 10** and **Table 4**, from which we can find that the in-situ experiment has the same conclusion with the simulated experiment. HTTP/2-Advance performs better than HTTP/2 when using three multiple TCP connections to multiplex HTTP requests and responses, and the average application layer latency has minute difference between Gzip and Network-Friendly header compression strategies. When the multiple TCP connections is three and the Gzip header compression is usedi, the average HTTP/2-Advance optimization proportion of Sina Weibo is 25.86% and that of UC Browser is 48.39%.

6 Conclusions

In this paper, popular mobile applications are classified by the data flow characteristics. Then, an optimization of HTTP/2 for mobile applications, called HTTP/2-Advance, is proposed for the scenario of mobile applications. Different applications are also simulated and tested in several network circumstances parameterized by the latency and the packet loss rate. By changing the header compressions strategies and the network conditions, the comparisons of HTTP, HTTP/2 and HTTP/2-Advance in different scenarios



▲Figure 9. The actual experiment circumstance. The client is connected to the server via a conversion gateway that converts HTTP/2 and HTTP.



▲Figure 10. The in-situ experiment on the number of multiple TCP connections influence and header compression influence. The in-situ result has the same conclusion with the simulated experiment.

▼Table 4. Application layer latency in in-situ experiment

	Protocols	Mobile applications	
		Sina Weibo	UC Browser
Average app layer latency (ms)	HTTP	57.6983	55.37302
	HTTP/2-Advance	42.7775	28.57672
Average optimization proportion		25.86%	48.39%

HTTP: Hyper Text Transfer Protocol TCP: Transmission Control Protocol

are tested and analyzed. The results indicate that the HTTP/2-Advance enables a more efficient way than HTTP/2 in mobile applications, and it performs better than HTTP/2, especially when the network condition is good. Overall, multiplexing three TCP connections is feasible for most mobile applications.

References

[1] Google Play. (2016). *Google Play for Android Applications* [Online]. Available: http://play.google.com
[2] *Hybird* [Online]. Available: http://www.hybird.org
[3] M. Belshe, M. Thomson, and R. Peon. (2015, May). *Hypertext Transfer Protocol Version 2 (HTTP/2)* [Online]. Available: http://tools.ietf.org/html/rfc7540
[4] M. Thomson (ed.), M. Belshe, and R. Peon. (2015, Feb. 11). *Hypertext Transfer Protocol version 2 - draft-ietf-httpbis-http2-16* [Online]. Available: https://tools.ietf.org/html/draft-ietf-httpbis-http2-16
[5] M. Belshe, and R. Peon. (2013, Nov.). *SPDY protocol—Draft 3.1* [Online]. Available: http://www. chromium. org/spdy/spdyprotocol/spdy-protocol-draft3-1
[6] W3Techs. (2015, Jul.). *Usage of HTTP/2 for websites* [Online]. Available: http://w3techs.com/technologies/details/ce-http2/all/all
[7] J. Padhye and H. F. Nielsen, "A comparison of SPDY and HTTP performance," Microsoft Technical Report MSR-TR-2012-102, 2012.
[8] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How speedy is SPDY," in *Proc. 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI'14)*, Seattle, USA, 2014.
[9] Y. Elkhatib, G. Tyson, and M. Welzl. "Can SPDY really make the web faster?" in *IFIP Networking Conference*, Trondheim, Norway, Jun. 2014. doi: 10.1109/IFIPNetworking.2014.6857089.
[10] J. Khalid, S. Agarwal, A. Akella, and J. Padhye. (2016). *Improving the performance of SPDY for mobile devices* [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/sagarwal-hotmobile15-poster.pdf
[11] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan, "Towards a SPDY'ier mobile web?" in *Proc. Ninth ACM Conference on Emerging Networking Experiments and Technologies*, Santa Barbara, USA, Dec. 2013, pp. 303–314. doi: 10.1145/2535372.2535399.
[12] G. Mineki, S. Uemura, and T. Hasegawa, "SPDY accelerator for improving web access speed," in *15th International Conference on Advanced Communication Technology (ICACT)*, Pyeong Chang, Korea (South), Jan. 2013, pp. 540–544.
[13] Google Play. (2016). *UC Browser—Google Play for Android Applications* [Online]. Available: https://play.google.com/store/apps/details?id=com.UCMobile
[14] Google Play. (2016). *Nanfang Weekend - Google Play for Android Applications* [Online]. Available: https://play.google.com/store/apps/details?id=net.coollet.infzmreader
[15] Google Play. (2016). *Xunlei Kankan—Google Play for Android Applications* [Online]. Available: play.google.com/store/applications/developer?id=迅雷看看&hl=zh_TW
[16] Google Play. (2016). *Sina Weibo—Google Play for Android Applications* [Online]. Available: play.google.com/store/applications/details?id=com.sina.weibo
[17] Xinhua. (2014, May). *Data of phone browsers in Q1 2014* [Online]. Available: http://www.bj.xinhuanet.com/hbpd/hbrj/rjy/2014-05/16/c_1110729487.htm
[18] WANem. (2016). *The Wide Area Network emulator* [Online]. Available: http://wanem.sourceforge.net
[19] Google Play. (2016). *Shark - Google Play for Android Applications* [Online]. Available: http://play.google.com/store/applications/details?id=lv.n3o.shark
[20] I. Grigorik, "Making the web faster with HTTP 2.0," *Communications of the ACM*, vol. 56, no.12, pp. 42–49, Dec. 2013. doi: 10.1145/2534706.2534721.
[21] W. Tarreau, A. Jeffries, and A. de Croy. (2012, Mar.). *Proposal for a Network-Friendly HTTP Upgrade* [Online]. Available: https://www.ietf.org/archive/id/draft-tarreau-httpbis-network-friendly-00.txt
[22] Gzip. (2016). *The gizp home page* [Online]. Available: http://www.gzip.org

Manuscript received: 2015-07-12

Biographies

DONG Zhenjiang (dong.zhenjiang@zte.com.cn) received his MS degree in telecommunication from Harbin Institute of Technology, China in 1996. He is vice president of the Cloud Computing & IT Research Institute of ZTE Corporation. His main research areas are cloud computing, big data, new media, and mobile Internet technologies.

SHUANG Kai (shuangk@bupt.edu.cn) received his PhD from State Key Laboratory of Networking & Switching Technology, Beijing University of Posts & Telecommunications (BUPT) in 2006. He is currently an associate professor with the BUPT. His research interests include cloud computing and the mobile Internet.

CAI Yanan (522018144@qq.com) is a master candidate of Beijing University of Posts and Telecommunications, China. Her research direction is the mobile Internet.

WANG Wei (wang.wei8@zte.com.cn) received her MS degree from Nanjing University of Aeronautics and Astronautics, China. She is an engineer and project manager in the field of mobile Internet at the Cloud Computing and IT Research Institute of ZTE Corporation. Her research interests include new mobile Internet services and applications, PaaS, terminal application development, and other technologies. She has authored five academic papers.

LI Congbing (li.congbing@zte.com.cn) received his MS degree from Nanjing University of Science and Technology, China. He is currently a technical researcher in the field of mobile Internet and AI at the Service Research Institute of ZTE Corporation. His research interests include WebRTC, HTML5, open platform, and robot AI. He is responsible for researching mobile Internet IM services and mobile robot navigation.