Journal of
Electrical & Electronic Systems
ISSN: 2332-0796

Alshammari and Al-Mogren, J Electr Electron Syst 2016, 5:3
DOI: 10.4172/2332-0796.1000193

Review Article Open Access

HTTP/2 in Modern Web and Mobile Sensing-based Applications Analysis, Benchmarks and Current Issues

Abdullah S. Alshammari* and Ahmad Al-Mogren

Computer Science College King Saud University Riyadh, Saudi Arabia

Abstract

This research paper is a survey paper that focuses on the current status of HTTP protocol and the suggested solutions that are implemented recently. New protocols and some new modification to the current protocol have been discussed and implemented in the recent years to meet the new requirements of advanced web and mobile sensing-based applications. In this paper, we explore the current status of HTTP 1.1, SPDY, HTTP/2 and HTTP over UDP protocols. In addition to the analysis and benchmarking of HTTP/2 in modern web services and mobile sensing-based applications.

Keywords: HTTP; SPDY; HTTP2; UDP; QUIC; HTTP-Pipelining

Introduction

HTTP has been the Internet backbone for a long time even with some known drawback and bottlenecks. In HTTP protocol (version 1.0), the client had to open a new connection on each request, after each response the connection should be closed. This affects the performance of the web page load due to a known TCP problem " TCP Slow Start issue" which is related to the requirement of round trips for each connection initiating and that leads to a slowness once the connection is opened.

There are other issues such as security option in HTTP is not the first citizen and so on. There are several techniques and suggested protocol presented as experimental suggestions last years. In addition, HTTP 1.1 has been introduced several years ago that helps to improve the protocol.

The demand on HTTP protocol has been increasing in the recent years due to the increase of personal computers, smart phones, tablets and wearable devices. The protocol has been suggested as inter-communication protocol for web services and mobile sensing-based applications that depend on embedded sensors to collect and share data.

In this paper, we show some new techniques that are implemented by several web client and server applications. These techniques include HTTP Pipelining, SPDY/HTTP2 and CUIC protocol. HTTP pipelining has been implemented in several web browsers/web servers, and the result so far was better than HTTP 1.0, however, there are still some issues that not have been solved yet.

We start with the list of drawbacks in HTTP 1.x from the performance perspective in addition to the other criteria such as security. Then we discuss the new feature in HTTP 1.x that helps to improve the performance dramatically by presenting some benchmarks and technical details. HTTP 1.1 still has some issues and that will be presented in section too. Section V we will analyze a new protocol called SPDY, it is still in the experimental phase but it shows good results from a performance perspective, in addition to the new security and bandwidth features that help build better web browsers and web servers. We also describe the new protocol called CUIC (pronounced as Quick) which is a new protocol in transport layer that helps to implement HTTP over UDP, and we will see the possibility of building the HTTP protocol based on this type of protocols, and the idea of building HTTP applications based on UDP protocol in general.

Finally, we analyze the current HTTP 1.x and HTTP/2 protocol from a different perspective such as performance and energy consumption.

HTTP Protocol Background Study

HTTP 1.x protocol

The Hypertext Transfer Protocol is defined in the RFC as "an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through the extension of its request methods, error codes, and headers" [1].

HTTP has several bottlenecks such as the dependency on more than one connection for a parallel connection implementation. This leads to more problems such as extra round trips for each connection initiating (RTT)" [2].

HTTP by nature does not implement any security protocol, it is a plain text protocol, and can be easily inspected by the Man-In-The-Middle attack. SSL/TLS can be used, as an external layer to get what is known as "HTTPS".

HTTP 1.0 has the primitive support of caching mechanism, as in the web servers may cache a response with the help of "expires field" in HTTP header, to specify the expiring date of the response (or the document or image in case it's not a plain text response). The web browser (or the web client in general) can ask for the caching page in a conditional way using what's known as "conditional request". By specifying the conditional request the client may receive the cached response from the server, or the latest version depends on the client requirement and the server settings [3].

HTTP 1.0 did not deal well with the bandwidth, it is expected for

***Corresponding author:** Abdullah S. Alshammari, Computer Science College King Saudi University Riyadh, Saudi Arabia, Tel: +966 11 805 0953; E-mail: info@abdullah.sh

Received May 16, 2016; **Accepted** August 22, 2016; **Published** August 29, 2016

Citation: Alshammari AS, Al-Mogren A (2016) HTTP/2 in Modern Web and Mobile Sensing-based Applications Analysis, Benchmarks and Current Issues. J Electr Electron Syst 5: 193. doi: 10.4172/2332-0796.1000193

a wasted bandwidth issue in the forward direction case. Let say that an HTTP/1.0 server receives a big request which could not be accepted due to large requests, that will lead to an error code. That will not help since the bandwidth had already been consumed. What it's should be implemented is the ability to communicate with a server and to ensure that it's able to handle this type of request size before even sending them. Another issue is related to a client, a client may need only part of a resource. For example, the client may need to display just the start of a big document, or it may want to resume a downloading file after a transfer was stopped in mid-stream [4].

HTTP 1.1 pipeline

HTTP 1.0 requests are generated in sequential order, so the next request is not issued until the response to the first request has been received completely and the connection is closed. In addition, multiple connection issues is a known problem in HTTP 1.0. Depending on network bandwidth and latencies, this can result in a delay before the server starts receiving the next request.

HTTP/1.1 has a new feature that calls persistent connection; which allows for multiple requests and responses during the same connection. In HTTP/1.1, the persistent connections is a default option and HTTP/1.1 clients, servers, and proxies will suppose that a connection will be kept open after the requests and their corresponding response have been processed and received. However, it is still possible to close a connection at any time, in order to save the resources and this could be done through tuning of persistent connections feature by sending the head "connection: close" to inform the recipients of the request that the connection could not be used anymore (Figure 1).

HTTP/1.1 also allows for multiple requests to be sent through a socket without pausing and waiting for the other responses. HTTP clients are granted to receive the responses in the same order in which they were issued. Requests pipelining can lead to a dramatic improvement in the performance of page loading and that performance improved more over high latency connections [5-8] (Figure 2).

SPDY protocol

SPDY is a modified protocol of HTTP and it's primarily by Google for transporting web content [2]. SPDY improves HTTP-based applications performance by reducing the web page load latency in addition to other features such as security improvements.

SPDY adds various features such as multiplexing of multiple requests, which is a parallel stream of request on a single TCP connection. It also adds a framing layer which is optimized for HTTP request-response multiplexing stream, with the keeping of backward compatibility of the current HTTP applications, so they can work over SPDY without affecting the applications that do not support SPDY protocol.

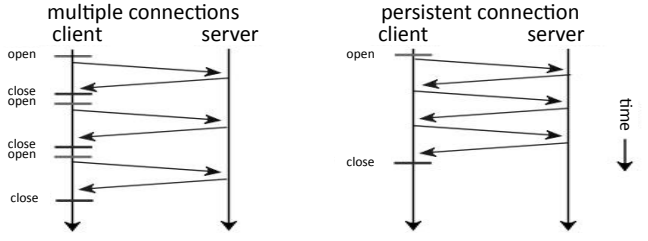

Figure 1: A comparison between HTTP 1.1 and HTTP 1.0 tcp connection: It shows the different between reused connections and non-reused connection which is known as a persistent and non-persistent connection.

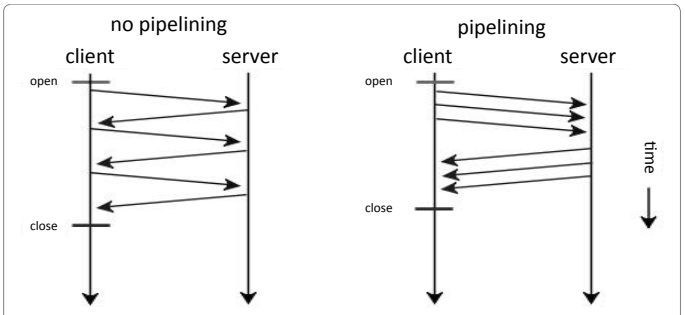

Figure 2: HTTP with and without pipelining: It shows how the HTTP pipelining works comparing that with non-pipelining, It's obvious that the pipelined version should be faster, but that's not the case all time.

Since SPDY improve the performance by decreasing the number of required TCP connection to load a web page, it improves the performance in a dramatic way, especially for the web pages contain more resource such as multiple CSS and JavaScript files. That's done by a mechanism which is called multiplexing. Multiplexing is a process of sending a list of HTTP requests in a single TLS/TCP connection. This has a better latency improvement comparing with the pipelining feature that introduced in HTTP 1.1. When the SPDY protocol in a browser that supports it, each web request will never wait due to the "connection limitation exhausted" error, for example, the limit of an HTTP 1.1 connections is 6 or 8 connections to the same host name.

However, there are still some issues with SPDY and one of the main issues is the RTT still high at some points because there is a known issue in TCP called "TCP head-of-line blocking", and that's why QUIC protocol appeared which it depends on UDP instead of TCP. We discuss this protocol later in this paper.

HTTP Requests prioritization is another feature. The client can issue a request for specific resources to be delivered before any other requests. This helps to avoid the problem of filling the network with a fewer priority resources while another high priority HTTP request is pending. Another feature is headers compression and that overcomes the current issues where the client sends a significant amount of the same data in the HTTP headers, the compressing algorithm that is used is GZIP compression algorithm. Finally, the last feature is server pushed streams which allow the server to push the content to clients without a need for a request so it acts a publish -subscribe protocol which is useful in some applications such as chatting applications.

All HTTP 1.x features such as cookies, ETags, configurable headers work in the same way in SPDY, it only replaces the way the data is written to the socket [9-14].

QUIC protocol and HTTP over UDP

One of the reasons for introducing QUIC protocol was that in TCP there is a known issue called a delay of a single packet "head-of -line blocking" for an entire set of HTTP streams. QUIC has improved the multiplexing request which means that only one stream would pause.

QUIC provides better features such as high security which is similar to TLS in TCP, Fast connectivity with 0 round trip time, packet pacing to reduce the cost of packets, packet error correction instead of retransmission which lead to a retransmission latency, and it avoids TCP head-of-line blocking issue (Table 1).

HTTP/2 Protocol

HTTP 2.0 is a new protocol which has an RFC called rfc7540 [15].

Protocol	Properties			
	RTT	Congestion control	Security	On Packet Lost
HTTP/TCP	>0	Exists	TLS	Resending
HTTP/UDP	0	Exists using a Pluggable interface	Crypto	Packet Pacing

Table 1: A Comparison between HTTP over UDP vs. TCP: shows a comparison between HTTP Implementation of TCP over UDP and HTTP over TCP.

This protocol based on SPDY protocol with some differences. HTTP 2.0 allows for faster and more secure data transfer in web applications and services. HTTP/2 is based on SPDY which is explained earlier. It keeps the same feature as HTTP/1.x. This includes HTTP methods such as GET and POST and DELETE, status codes such as 200, URL format, and how header fields are defined and used. HTTP/2 introduces several features such as the supporting of single and persistent connection, multiplexing, header compression using HPACK algorithm, prioritization and TLS/SSL Encryption support within the protocol itself.

HTTP/2 single and persistent connection

From a theoretical perspective, HTTP implementation on top of UDP will help improve the performance since there a lot of bottlenecks in TCP will be ignored using UDP such as HOL packet issue (Head of Line), RTT, error correction and congestion control. QUIC (Quick UDP Internet Connections) is an early-stage network protocol that is in development phase right now by Google, it " runs a stream multiplexing protocol over a new flavor of Transport Layer Security (TLS) on top of UDP instead of TCP" [4].

HTTP/1.0 protocol enforces the client to open a connection for each request. Even though it's possible to keep the connection by setting the Connection header field to keep-alive HTTP/2, it's not mentioned clearly in the RFC so it's not considered as a standard technique (Figure 3). Connection is single and persistent which helps to carry more requests at the same connection: This allows many requests to be shared at the same connection. This helps to reduce the network congestion since the number of TCP connections is lower. And it reduces the CPU and memory usage because of the fewer connections that are open at the same time which in turn saves the energy on the mobile device.

Multiplexing

Each connection used by HTTP/2 can support more than one request which is called multiplexing, this is allowing it to be used by opened requests and responses. On another hand, HTTP 1.1 open up to 6 connections to mimic the multiplexing feature.

Table 2 shows a case of page-loading process workflow. It is clear that HTTP/1.1 requires more time and steps to complete the process than HTTP/2 which leads to a better performance.

Header compression and binary encoding

In HTTP/1.x, header data is sent as plain text. This has been changed in HTTP/2. The header data compressed using a special algorithm which is called HPACK. It is a binary encoding for the header and it's improved comparing with GZIP algorithm. It's defined in RFC7541 [5].

HPACK compressing adds CPU overhead and in some cases where the uncompressed request header fits in a single TCP packet, compression doesn't help.

In HPACK compression, a static table contains known header fields and common values, each assigned an index number. A recent study shows compression of greater than 50% on headers sent by the client and nearly 90% on headers sent by the server [6].

HTTP/1.1 Loading of page	HTTP/2 loading of page
Up to 6 connection	Single Connection.
Request the HTML page.	Request the HTML page.
The page is received	The page is received
Issue 6 requests to fetch files in HTML pages	Request all files
Wait for response	Multiplexing
Request next resource (repeat until finish)	
Close all connection	Close just one connection

Table 2: HTTP 2.0 multiplexing vs. HTTP 1.1.

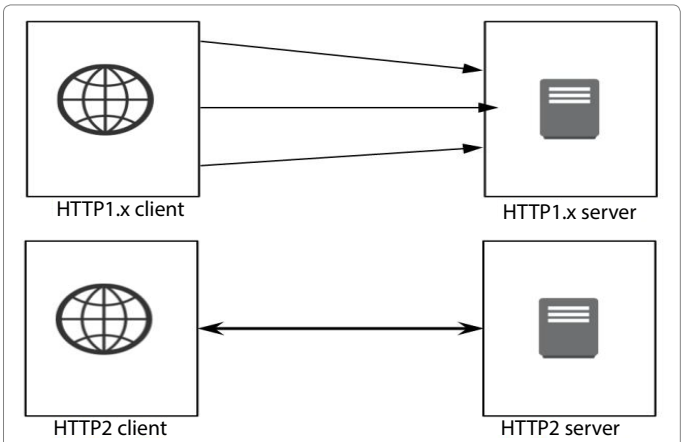

Figure 3: HTTP2.0 multiple connections vs. HTTP 1.x persistent connection.

Prioritization

In HTTP/2 it's possible to prioritize the requests, so a request can be received from the other if its high prioritized. This normally useful in a web page that contains a large number of resources such as JavaScript or CSS files [16].

HTTP Performance on Mobile Devices

In recent years, the popularity of mobile devices (e.g., smartphones, and tablets) has dramatically increased, as millions of users are using these devices in their daily lives. These devices offer users the opportunity to have more computational power and even more capability in the palm of their hand than most users had on their desktop just a few years ago. This leads to the main concern about the current protocol used traditionally in the desktop applications and if they are energy efficient in the mobile applications environment. A recent study [17] shows that HTTP/2 can save energy more than the current HTTP 1.x protocol. This study compares a real world example using Google.com and Twitter.com websites to evaluate the client mobile applications (Figure 4).

To compare HTTP/2 protocol to HTTP 1.x protocol from the energy consumption perspective, we prepare a new experiment using Android OS and Android APIs from the client side. From the server side we use nginx with http/2 support. It is a high concurrency web

server. It can also act as a reverse proxy server for a several different protocols. We build a special website for this purpose, the website specifications are shown in Table 3.

To measure the energy consumption we use Android energy consumption tool that is shipped with any Android device. We run an http 1.x app for 4 hours that sends multiple requests over the time and we see how much percentage is consumed during the running time. The same applied to HTTP/2 app (Figure 5).

Performance Benchmark

HTTP 1.1 pipelinging performance

We conduct a test to find how fast the HTTP pipeline compared with non-pipelined one. This test is based on the open source web engine Firefox v32.0. We applied the test by sending HTTP requests to a web server, these requests have been run two times, the first round was without HTTP pipelining and the second one is with HTTP pipelining (Figure 6). It shows the result, it's obvious that HTTP pipelined helps improving the performance most of the time. However, first run of the test shows that pipelining feature does not help improving the performance. This happens sometimes if the server does not support HTTP pipelining properly or some queuing and reordering happened during the request sending. This is because all pending requests have to wait for the first request to be received and this may lead to a performance issue in some cases.

QUIC (HTTP over UDP) protocol performance

This test is to find the performance of HTTP under different protocols/techniques. The comparison of the performance is between HTTP/2 and QUIC (HTTP over UDP). The simulation is done under three different network conditions which are (latency, lost packets and download speed) and by downloading a 20 Mega Bytes file. The web browser is Chromium.

In Figure 7 the comparison shows the normal HTTP request (HTTP2 over TCP) with QUIC (HTTP2 over UDP) as the packet loss increasingly. When 1% of the packet lost, the speed of downloading drops down from 4Mbps to 1Mbps. That because TCP tries to resend the lost packets again, but UDP does not, however, since QUIC added the Forward error correction concept to UDP, it's the reason of this slowness. In Figure 8 the comparison shows file downloading via normal HTTP request (HTTP2 over TCP) versus QUIC (HTTP2 over UDP) as the latency increase. This diagram shows that UDP degrades the download speed as the latency increase more than TCP does [18].

HTTP/2 on web services performance

Among the key improvements brought by HTTP/2 are multiplexed streams, header compression, server push, and a binary protocol instead of textual one. These and other positive changes allowed to achieve good web pages loading results, including those having lots of additional files attached to them (e.g. styles, scripts, images, fonts). There are a lot of web service protocols that have been implemented during the last decade including Protocol Buffer, Protocol Buffer over HTTP2, Thrift and MQTT [19].

In this performance benchmark study we create a simple Web Service APIs, we use nginx as HTTP/2 web server and chrome developers tool as a benchmarking tool in addition to Apache Benchmark and Apache work which both help benchmark the web service API in real web services environment. We Measure the performance by counting the number of requests per second and

Protocol	Properties			
	HTML	Image	CSS	JS
Test website	1	238	5	1

Table 3: Our Comparison between HTTP over UDP vs. TCP.

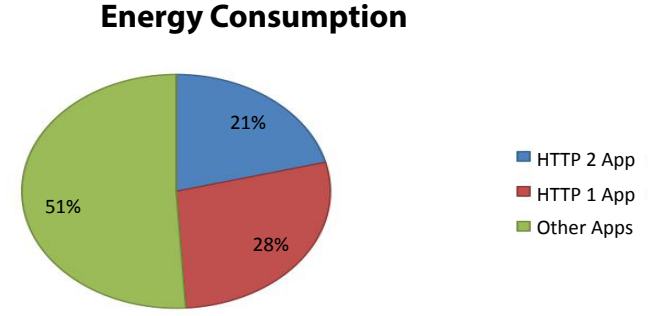

Figure 4: HTTP2.0 vs. HTTP 1.x energy consumption.

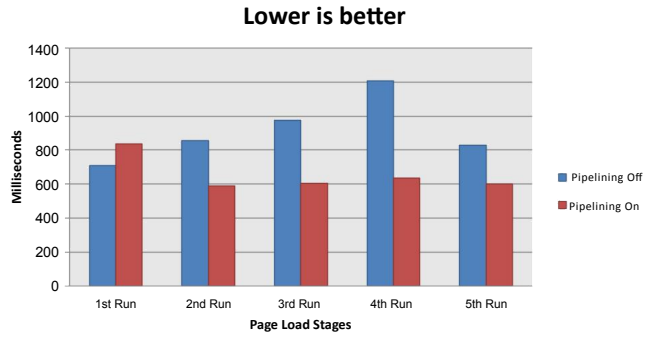

Figure 5: An experiment for the HTTP Pipelining Performance vs. No HTTP Pipelining on Firefox v32.0: The final results shows that HTTP 2 is better than HTTP 1.x in the term of energy consumption.

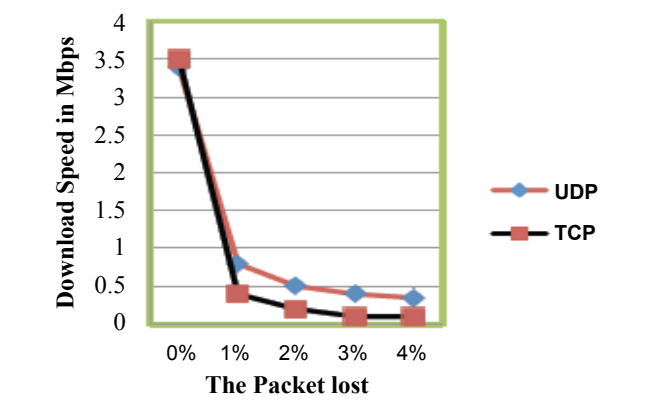

Figure 6: File Download speed decreased as packets loss increasing, HTTP/UDP is a little bit better than HTTP/TCP.

the result in Figure 8 shows that HTTP/2 could help improving the performance by up to 30% comparing to HTTP 1.x protocol. In General, HTTP/2 main improvement is focusing on web applications with so many resources instead of single web service (Figure 8).

Conclusion

We showed the status of HTTP protocol from the beginning of HTTP 1.0 to the current status. HTTP 1.0 had a limitation in many

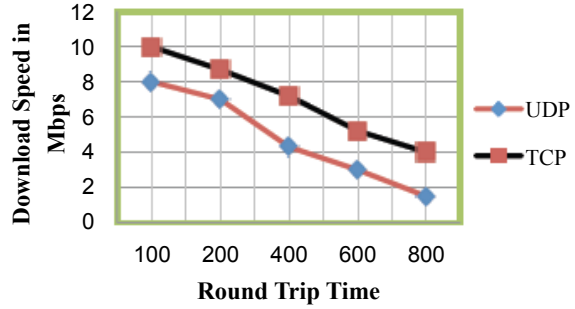

Figure 7: File Download speed decreased as the latency increase, HTTP over UDP is worse than HTTP over TCP.

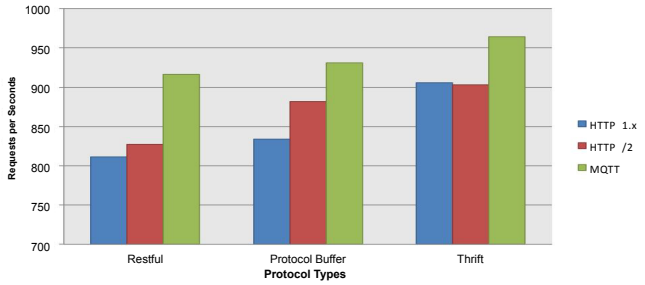

Figure 8: Number of requests per second in HTTP 2.x, HTTP 1.x and MQTT protocols.

areas which has been improved in HTTP 1.1, especially regarding the performance and bandwidth improvements by introducing HTTP pipelining. We show that HTTP pipelining has some limitations in HTTP specification which is called "head of line blocking", since the specification force the web server to deliver the requests in the same order as it arrived. That issue almost solved in SPDY protocol, by introducing HTML multiplexing mechanism.

Implementing HTTP protocol on top of TCP is not the only option, QUIC is another protocol, that solve some issues such as HOL (Head of Line) packet and it should improve the performance from theoretically perspective. However, the tests that we conducted and the resulted benchmarks show clearly that HTTP over UDP has some main issues and sometimes it's worse than TCP especially when the latency is high. The working on HTTP over UDP idea is going on to improve HTTP response time, and there are many ideas that we can work on such as using a different protocol like RTP and SCTP (Stream Control Transmission Protocol) which may increase the performance and overcome the current issues in HTTP performance.

References

1. Yang F, Amer P, Leighton J, Belshe M (2012) A Methodology to Derive SPDY's Initial Dictionary for Zlib Compression.
2. Belshe M, Peon R (2012) SPDY protocol. The Chromium Projects.
3. Thomas B, Jurdak R, Atkinson J (2012) SPDYing up the web. Communications of the ACM 55: 64-73.
4. Regundwar NP, Shukla DA, Lokhande P (2013) A Study paper on SPDY protocol: Let's make the web faster. International Journal of Scientific and Engineering Research 4: 222-226.
5. Carlucci G, Cicco LD, Mascolo S (2015) HTTP over UDP: an Experimental Investigation of QUIC. ACM SAC.
6. Peon R, Ruellan H (2015) HPACK: Header Compression for HTTP/2. Internet Engineering Task Force.
7. Nginx (2015) HTTP/2 for Web Application Developers.
8. Montenegro G, Mazahir O, Padhye J, Trace R (2012) HTTP 2.0 Principles for Flow Control.
9. Shimizu K, Kihara B (2012) Considerations for Protocols with Compression over TLS. Network Working Group.
10. Miller K (2010) Wormhole-An Active HTTP Tunnel. EuroSys Conference, Paris.
11. Peon R (2012) Explicit Proxies for HTTP/2.0. Network Working Group.
12. Barrett M, Blackledge J (2011) A Transparent approach to web site streaming using Man-in-The-middle code injection and HTML5 features. ISAST Transactions on Computers and Intelligent Systems 3: 1-70.
13. Erman J, Gopalakrishnan V, Jana R, Ramakrishnan KK (2013) Towards a SPDY'ier mobile web. Proceedings of the ninth ACM conference on Emerging networking experiments and technologies. New York, NY, USA.
14. Fielding R, Gettys J, Frystyket H, Masinter L, Leach P, et al. (1999) Hypertext transfer protocol--HTTP/1.1. No. RFC 2616. Network Working Group.
15. Stewart R (2007) Stream control transmission protocol. Network Working Group.
16. Grigorik I (2013) Making the web faster with HTTP 2.0. Communications of the ACM 56: 42-49.
17. Chowdhury SA, Sapra V, Hindle A (2015) Is HTTP/2 more energy efficient than HTTP/1.1 for mobile users. PeerJ PrePrints 3.
18. Thomson M (2015) Hypertext Transfer Protocol Version 2 (HTTP/2). No. RFC 7540. Internet Engineering Task Force.
19. Essaili AE, Schroeder D, Staehle D, Shehada M, et al. (2013) Quality-of-experience driven adaptive HTTP media delivery. IEEE International Conference on Communications (ICC).