How quick is QUIC?

Péter Megyesi, Zsolt Krämer, Sándor Molnár
High Speed Networks Lab., Dept. of Telecomm. and Media Informatics,
Budapest University of Technology and Economics, Budapest, Hungary
E-mail: {megyesi, kramer, molnar}@tmit.bme.hu

***Abstract*—HTTP has been the protocol for transferring web traffic over the Internet since the 90s. However, over the past 20 years websites have evolved so much that today this protocol does not provide optimal delivery over the Internet and became a bottleneck in decreasing page load times. Google is pioneering in finding better solutions for downloading web pages and they implemented two new protocols: SPDY in 2009 and QUIC in 2013. Since the wide range deployment of SPDY clients and servers it has been revealed that in some scenarios SPDY can negatively affect the page transfer time mainly due to the fact that the protocol is working over TCP. To tackle these obstacles QUIC uses its own congestion control and based on UDP in the transport layer. Since QUIC is a very recent protocol, this paper could help further understand its operation and performance in a wide range of network scenarios. We present a comprehensive study about the performance of QUIC, SPDY and HTTP particularly about how they affect page load time. We found that none of these protocols is clearly better than the other two and the actual network conditions determine which protocol performs the best.**

I. Introduction

Regarding today's Internet, one of the most widely used protocols is the Hypertext Transfer Protocol (HTTP) which is responsible for delivering news, video, and countless other web applications to billions of devices of all sizes, from desktop computers to smartphones. However, our websites have changed significantly since the publication of HTTP 1.1 in RFC 2616. As web pages and web applications continue to evolve and the Internet traffic increases rapidly, it becomes necessary to search for improvements and new technologies. Moreover, Page Load Time (PLT) has become a crucial aspect of web performance since it is directly correlated with page abandonment. As a result, web service providers are constantly working on finding better solutions for web page transfers.

Google started to develop a new web transfer protocol called SPDY in 2009 [1]. Today, SPDY is implemented not only on the servers of Google and in the Chrome browser, but on popular sites such as Facebook and Twitter, and the protocol is also supported by the newest versions of Firefox and Internet Explorer. The new HTTP/2 which is being developed by the Internet Engineering Task Force (IETF) and is now a proposed standard, is largely based on SPDY [2].

However, it is not just HTTP 1.1 that has its shortcomings and limitations. In the transport layer, TCP (Transmission Control Protocol) which delivered remarkable results in the past, also has some issues to deal with. Google has sound observations on TCP performance in recent networks since about 20-25% of all Internet traffic goes through their servers, and Chrome is the market leader web browser with 40% market share. Based on the these experiences Google started to develop a new protocol called QUIC (Quick UDP Internet Connections) [3] in 2013 which uses UDP (User Datagram Protocol) protocol in the transport layer instead of the traditional TCP.

Since QUIC is a very recent protocol, little is known about its performance in comparison with the two aforementioned protocols. The main goal of this paper is to contribute to the better understanding of the performance of QUIC comparing it with the traditional HTTP 1.1 and SPDY.

This paper is organized as follows. Section II describes the background of SPDY and QUIC and also presents the related work. In Section III we give details about the measurement environment we used to test the performance of the QUIC, SPDY and HTTP protocols. Section IV presents the results of our measurements. Finally, in Section V we summarize our work.

II. Background and Related Work

A. HTTP 1.1 Limitations

HTTP 1.1 is a request-response based protocol, designed in the 1990's when web pages were much simpler than they are nowadays. Developers and users are now demanding near-real-time responsiveness which HTTP 1.1 cannot meet mainly due to the following limitations.

One of the bottlenecks of HTTP performance is the opening of too many TCP connections to achieve concurrency. A large portion of HTTP data flows consist of small (less than 15KB), bursty data transfers over dozens of distinct TCP connections as presented in Fig 1. However, TCP is optimized for long-lived connections and network Round-Trip Time (RTT) is also a limiting factor in the speed of the new connections due to TCP congestion control mechanisms. If HTTP tries to improve performance by opening more TCP connections and the number of the connections is too large, network congestion can occur, resulting in high packet loss and ultimately worse performance. The number of connections becomes even higher if web objects are being delivered from multiple domains. A workaround of this problem was introduced with HTTP pipelining but it effectively failed due to deployment issues [4].

Another limitation is that HTTP based web transfers are strictly initiated by the client. This presents a serious problem because it hurts performance significantly in the case of loading embedded objects. The servers have to wait for an explicit request from the client which can only be sent after the client processed the parent page.

Moreover, as a TCP segment cannot carry more than one HTTP request or response, clients are sending a considerable amount of redundant data in the form of HTTP headers.

978-1-4799-6664-6/16/$31.00 ©2016 IEEE

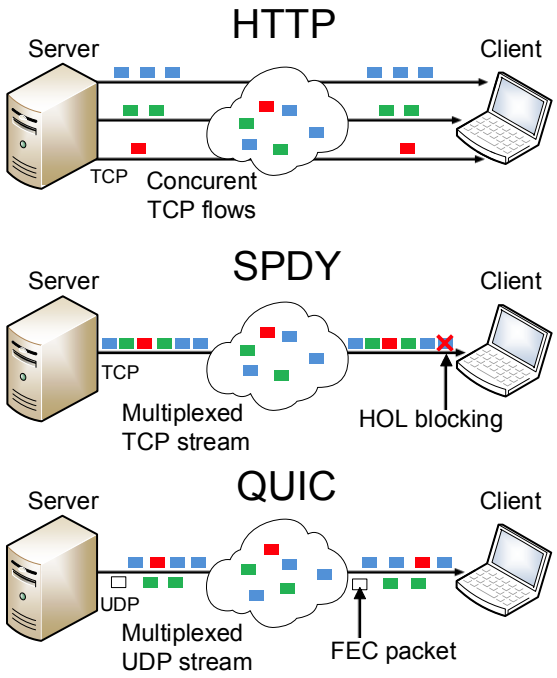


Fig. 1: Streams in HTTP, SPDY and QUIC protocols

This overhead is especially large if there are many small embedded objects on a page so the requests and responses are small. For modems or ADSL (Asymmetric Digital Subscriber Line) connections in which the uplink bandwidth is fairly low the latency caused by this overhead can be significant. The developer community used to try to minimize this effect by concatenating small files with the same type creating larger bundles and in some cases files are inlined in the HTML (HyperText Markup Language) document to avoid the request entirely. These workarounds can backfire though, as concatenating has a negative impact on caching and the common method to concatenate Cascading Style Sheets (CSS) and JavaScript files delays processing [4].

B. SPDY

SPDY[1] protocol is designed to fix the aforementioned issues of HTTP [1]. The protocol operates in the application layer on top of TCP. The framing layer of SPDY is optimized for HTTP-like response-request streams enabling web applications that run on HTTP to run on SPDY with little or no modifications. The key improvements offered by SPDY are described below.

SPDY uses multiplexed requests and opens a single TCP connection to a domain as shown in Fig. 1. There is no limit to the requests that can be handled concurrently within the same SPDY connection (called SPDY session). These requests create streams in the session which are bidirectional flows of data. This multiplexing is a much more fine-tuned solution than HTTP pipelining. It helps with reducing SSL (Secure Sockets Layer) overhead, avoiding network congestion and improves server efficiency. Streams can be created on either the server- or the client side and can concurrently send data interleaved with other streams [1].

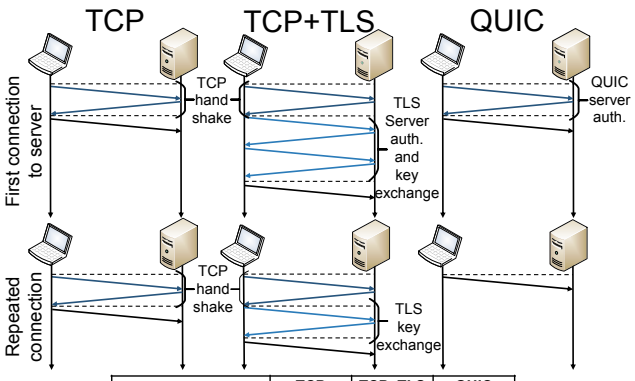
SPDY also introduces request prioritization. The client is allowed to specify a priority level for each object and the server then schedules the transfer of the objects accordingly. This helps avoiding the problem when the network channel is congested with non-critical resources and high-priority requests (e.g., JavaScript code modules) are pending.

Server push mechanism is also included in SPDY thus servers can send data before the explicit request from the client. Without this feature, the client must first download the primary document, and only after it can request the secondary resources. Server push is designed to improve latency when loading embedded objects but it can also reduce the efficiency of caching in a case where the objects are already cached on the clients side thus the optimization of this mechanism is still in progress.

Furthermore, clients today send a significant amount of data in the form of redundant HTTP headers if a web page requires many subrequests. SPDY presents a solution to this issue by introducing header compression. The result is fewer packets and bytes transmitted, and this could improve the serialization latency to send requests.

SPDY's performance still to this day is not perfectly understood since conflicting results can be found in the literature. Google [5] and Microsoft [6] reported significant performance increase (up to 60% speedup in PLT) for SPDY compared to HTTP, whereas results by Akamai [7] and Cable Labs [8] showed only modest performance gain or even slight performance loss in some scenarios. [9] and [10] also reported small gain in high RTT scenarios (e.g., satellite connections) but another study [11] showed that this gain is lost over 3G network due to the structure of cellular networks.

[12] and [13] are two very recent studies about SPDY with comprehensive sweep of the network parameter space over a controlled and isolated testbed. Both papers showed similar results: i) SPDY has better performance than HTTP in case of high object number or high RTT mainly due to its multiplexing and header compression features, ii) SPDY's gain is larger when the bandwidth is low, in high bandwidth



	TCP	TCP+TLS	QUIC
First connection	1 RTT	3 RTT	1 RTT
Repeated connection	1 RTT	2 RTT	0 RTT

Fig. 2: Connection Round Trip Times in TCP, TLS and QUIC protocols

[1]SPDY is pronounced speedy and is not an acronym.

scenarios the results are comparable to HTTP, iii) HTTP performs batter in high packet loss environments where the performance of SPDY is greatly decreases due to head-of-line blocking (HOL blocking). Our study strengthen these results related to the performance of SPDY vs. HTTP. However, we also compare QUIC to the these protocols, presenting performance comparisons between QUIC vs. HTTP and QUIC vs. SPDY.

C. TCP Limitations

SPDY successfully addressed many shortcomings of HTTP, but some issues that are hindering performance improvement on today's Internet stem from the use of TCP in the transport layer. One of the most important features in TCP is the congestion control mechanism used to avoid congestion collapse by adjusting sending rates. Numerous new TCP versions were suggested (e.g., [14], [15]) and research groups also proposed new alternative transport layer protocols to make the Internet faster. However, these processes are very slow to evolve and especially slow to deploy. It not only needs to be deployed to servers and clients but also to middle boxes throughout the Internet thus this means that improvements in the protocol can easily take ten or even more years to spread.

Bandwidth is becoming less and less of a bottleneck regarding web performance as broadband Internet connections' throughput continue to grow and residential fiber solutions are gaining popularity. However, latency is often forgotten. Network RTT is the limiting factor in throughput of new TCP connections and is mainly governed by the speed of light, thus reducing the number of round-trips is the only way we can significantly lower these latencies. As presented in Fig. 2 TCP needs one round-trip to open the connection before the actual web request can be made with HTTP or SPDY. This number even increases when we use TLS encryption; at least one round-trip is added for TLS key exchange and in case of the first contact between the client and the server an authentication phase is also added which adds one more extra round-trip.

Another important issue with using TCP for transport is the occurrence of HOL blocking. TCP provides reliable data transfer and ordered delivery, thus this means that if one packet is lost, all data transfer needs to wait for the retransmission. In [12] and [13] authors of both papers showed that in network environments with high loss, HTTP outperforms SPDY because with SPDY there is only one TCP connection to a domain and HOL blocking causes all streams to wait, while HTTP opens many TCP connections and therefore it is much less affected by this problem.

D. QUIC

One of the most important goals of QUIC [16] is to reduce latency of web traffic by experimenting with the usage of UDP for downloading web applications instead of the traditional TCP solution. QUIC also implements a new encryption mechanism which replaces SSL/TLS, supports the reducing of round trips between client and server during connection establishment, while its safety remains comparable to TLS [16].

Since QUIC is working over UDP, the protocol does not force in-order delivery of packets thus QUIC avoids HOL blocking. Moreover, a client can achieve 0-RTT connection cost when connecting to a server if there was a connection established between the two before (see Fig. 2). This is achieved by including a connection ID in every packet which replaces the traditional IP fourtuple (source and destination address and port pairs). The extra round-trip in TLS is not an actual requirement of security and privacy reasons but only comes from the implementation of the handshake procedure. The encryption introduced in QUIC aims to change this and its design resembles DTLS (Datagram Transport Layer Security) according to the QUIC Design Document and Specification Rationale [16].

As shown in Fig. 1 QUIC also uses Forward Error Correction (FEC) codes in order to reduce retransmissions latency after packet loss. This means that QUIC is willing to sacrifice bandwidth for decreased latency by doing proactive speculative retransmission of critical packets, such as the connection establishment UDP packet. According to Google, 5% extra bandwidth used for sending FEC packets results in 8% less retransmissions [17].

The default congestion control algorithm in QUIC is TCP-Cubic, but TCP-Reno can also be used as an alternative. The protocol also implements a variety of TCP loss recovery mechanisms [18]. Furthermore, QUIC includes a feature called packet pacing which is under constant optimization. In order to make packet pacing effective, QUIC monitors inter-packet spacing and uses this to estimate available bandwidth and control packet pacing. Early measurements presented that packet pacing can reduce congestion related packet loss [17] but also that it can hinder performance significantly in a low-loss network environment [19].

With mobile web traffic at an inflexion point, making mobile connections faster is one of the highest priorities of today's research. As TCP relies on IP fourtuples to identify connections, when a mobile client changes network interface (IP address) the TCP connection will be automatically broken. Since QUIC uses a connection ID instead of the IP fourtuple, if the client changes networks interface it can still continue communicating with the server afterward. For example, when a mobile client uses a campus Wi-Fi connection to download a YouTube video and then walks out to the street and switches to cellular network connection, with QUIC the transition becomes smooth and the downloading proceeds. If the client were using TCP then a new connection has to be built up. Also, if for some reason, the UDP connection gets torn down, QUIC has the 0-RTT reconnection to fall back on.

Given that QUIC is a very recent protocol there are only a few studies examining its performance. Google presented some early research results in [16] and [17] and some scenarios are also presented in [19], but these studied completely lack of methodology descriptions. [20] concluded that QUIC outperforms SPDY in case of a lossy channel and that the FEC module, if enabled, worsens the performance of QUIC. [21] found that QUIC is faster than its predecessors over low bandwidth and high-delay links by 10-60%. The latter two

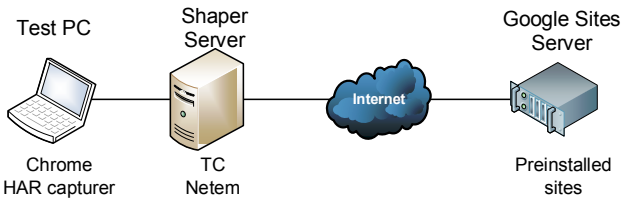


Fig. 3: The assembled measurement setup

studies used isolated testbeds and a prototype QUIC server made public [22] by Google. This makes the measurements completely repeatable but it can also endanger the accuracy of the results thus we used a different approach described in Sec. 3. Due to our knowledge, we are the first to present the performance of QUIC in wide range of live network scenarios.

III. Measurement Environment

In this section we provide insights to the methodology we used to compare the performance of QUIC, SPDY and HTTP. Fig. 3 sketches our assembled test environment. We used a regular laptop[2] with Chrome browser to download the context of websites and Chrome HAR capturer [23] to automate this process and generate log files in HTTP ARchive (HAR) format. HAR files contain every necessary information to extract page load times from the measurements. Also, browser caching was turned off during the process.

On the server side we installed four different simple pages on Google Sites. We chose this approach since the deployment of QUIC capable web servers is in a very early stage. QUIC works on most of the Google servers (including web services such as Gmail, YouTube and Google Translate) but other then these servers only a basic server module is available to test the protocol implementation [22]. Previous work with SPDY ([12], [13]) and QUIC ([20], [21]) used isolated testbeds in a controlled environment using Apache web server with SPDY module. However, since our measurements were conducted in our campus site where the Google AS is very close, we found very little variances of bandwidth and RTT values. Thus we believe that this approach has less influence on the results as if we had used the basic QUIC web server against Apache for HTTP and SPDY.

Since both QUIC and SPDY are multiplexing protocols, the gain of their usage would apply in case of websites with large number of objects. Regarding these, the four web pages we installed to Google Sites are containing either small sized (400B-8KB) or large sized (128KB) and either small number (5) or large number (50) of objects. The objects themselves are pictures: the small pictures are national flags whereas the large pictures are high resolution photos[3].

Moreover, we installed a shaper server between the client and the Google Sites server in order to emulate different network conditions. We used the Netem function of the TC (Traffic Control) package to manipulate the bandwidth, the packet loss and the delay of the connection. For bandwidth

[2]Intel Core i5 CPU with 4 GB RAM using Ubutu 14.04 (64 bit). Chrome version was 37.0.2062.94, Linux kernel version was 3.13.0-37

[3]Google Sites automatically resize large photos to 128KB. The original sized photos only available after clicking on the resized picture.

TABLE I: The parameter space of the measurements

Category	Parameter	Values
Network	Bandwidth	2 Mbps, 10 Mbps, 50 Mbps
	RTT	18 ms, 218 ms
	Packet loss	0%, 2%
Web page	Object number	5, 50
	Object size	400 B - 8 KB, 128 KB

we defined the values for small, medium and large access speed as 2 Mbps, 10 Mbps and 50 Mbps, respectively. In case of loss, we investigated two cases: low loss when we do not add any extra loss to the network and high loss when we set TC to randomly drop 2% of the packets in both upstream and downstream directions. Also, for delay in one case we do not add any extra delay to the network (thus the average RTT remained 18 ms) whereas for emulating high RTT scenarios we used TC to add an extra 100 ms for both up- and downstream directions, making the average RTT to 218 ms.

Finally, in Table I we summarized the five-dimension parameter space. These values define 48 distinct network scenarios and we completed the page downloads for all the three protocols at least a hundred times.

IV. Measurement Results

In this section we present the comparison of QUIC, SPDY and HTTP regarding how fast they can download the previously installed pages from the Google Sites server. Due to paper size limitation we do not present all the 48 different scenarios, but we emphasize some selected results which can capture the main advantages and disadvantages of the three protocols.

Fig. 4 presents the Cumulative Distribution Function (CDF) of the Page Load Times (PLT) in six different cases. In Fig. 4a and Fig. 4b the performance of the three protocols are comparable. Although, there are minor differences between the average values, the deviation of the curves are larger than this difference thus we can not state that one protocol is clearly better than the other two. Results were very similar using the pages with high number of small objects and low number of large objects. Thus we identified that the three protocols perform similarly under good network conditions (high bandwith, low RTT and low packet loss) downloading from websites with both small and medium size pages.

In case of large websites the difference between the protocols' performance shows larger deviation. Fig. 4c and Fig. 4d plot the values for the measurements with low packet loss and low RTT downloading the page with high number of large objects using 10 Mbps and 50 Mbps as a bottleneck, respectively. When the bandwidth is set to 10 Mbps (Fig. 4c) the page load times are again comparable, but in the 50 Mbps scenario (Fig. 4d) QUIC performs significantly worse than SPDY and HTTP. In this case the average PLT with QUIC is more than three times larger than for the other two protocols. The main reason for this behavior is the packet pacing mechanism in QUIC: the goodput of the multiplexed QUIC stream cannot reach the maximum capacity of such

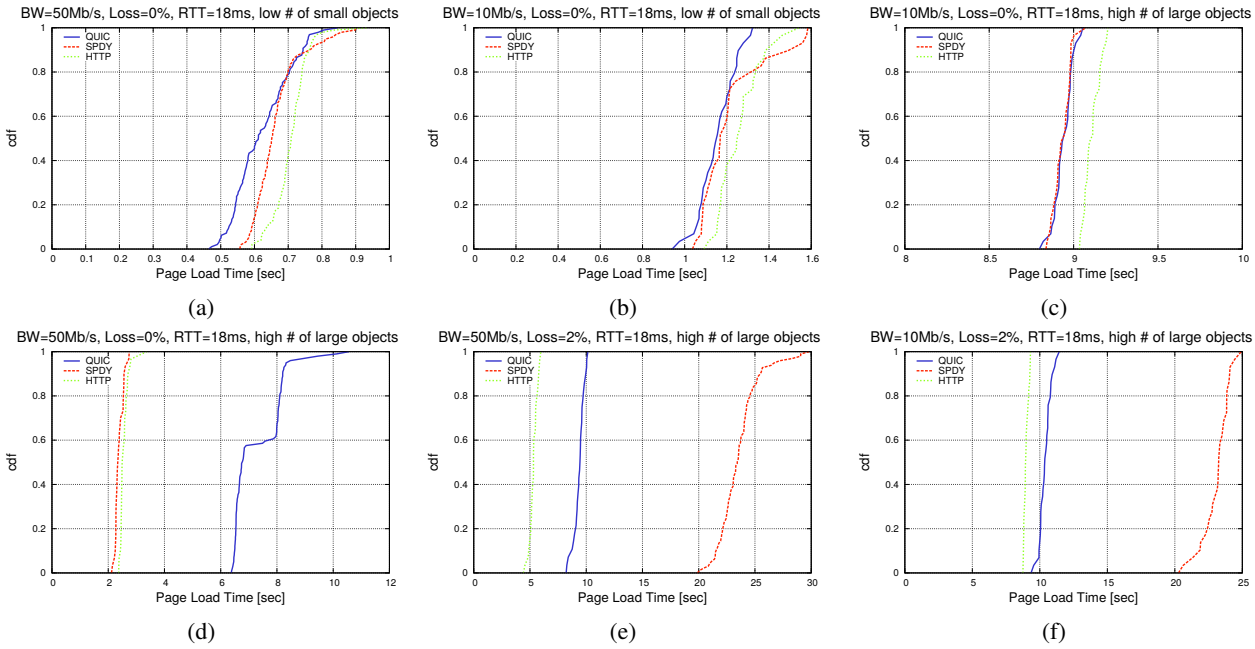


Fig. 4: Distribution of Page Load Times in various network conditions. One can investigate that none of the three protocols is better than the other two, the network conditions determine which one is the fastest.

high speed link. Since this behavior didn't occur nor in the 10 Mbps case neither for the smaller page types, we identified that QUIC only performs very poorly when facing with very high speed links and large amount of download data.

Fig. 4e and Fig. 4f and depicts the results for similar scenarios that are presented in Fig. 4d and Fig. 4c, respectively but with the addition of extra packet loss to the network. In this case SPDY performed very badly: the average PLT increased multiple times for SPDY after adding 2% of packet loss to the network. This phenomenon that HOL blocking can cause significant performance loss in SPDY's performance was already showed in previous publications [12], [13]. However, such amount of performance drop was not presented in those papers since authors didn't investigate such high speed access links. We also emphasize that this scenario corroborate to the idea of QUIC's proactive error correction mechanism since compared to the lossless case the PLT for QUIC only increased by 20% whereas for HTTP the average PLT roughly doubled.

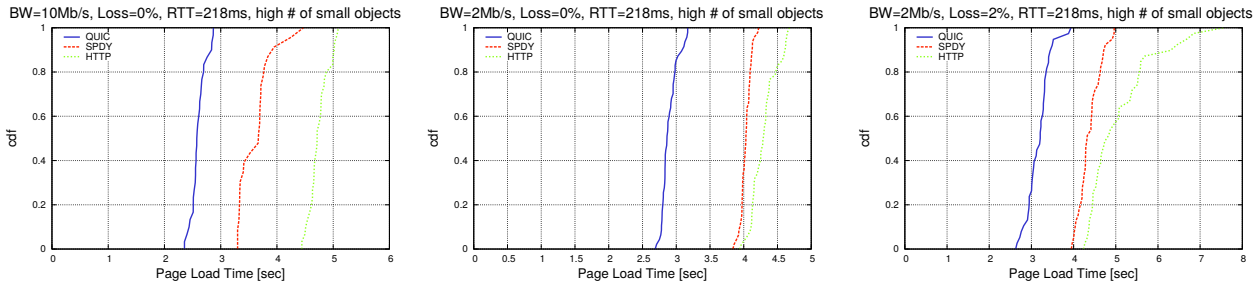
Fig. 5 presents the measurement results for three cases where QUIC clearly outperforms the other two protocols. The common parameters in these cases were high RTT and downloaded high number of small objects. In Fig. 5a and Fig. 5b the bandwidth was 10 Mbps and 2 Mbps, respectively and there was no additional loss on the channel, whereas in Fig. 5c we added 2% of loss with a bandwidth of 2 Mbps. Typically, one would expect that the gain from the multiplexing nature of QUIC and SPDY comes forward in case of high number of small objects and our results confirm this expectation. In this case the 0-RTT connection time of QUIC is also proven to work well: QUIC is by far the fastest protocol being roughly 25-30% faster than SPDY and 35-40% faster than HTTP. We also want to point out that the difference between SPDY and HTTP is about 7-12%, which also strengthens the finding in SPDY's literature about being only slightly faster than HTTP.

Furthermore, we summarize our measurement results in Fig. 6 where we present a decision tree for all the scenarios in the parameter space. We consider a protocol better than another if at least in the 70% of the measurement cases performed at least 10% faster than the other protocol's average PLT. If two of the protocols are fulfilling this condition against the third one but not against each other we marked both of them. In case of this condition doesn't stand between any of the three protocols we marked them as equal. Finally, based on the decision tree and the entire measurement database we concluded the following:

- QUIC performs poorly under very high bandwidth when large amount of data needs to be downloaded
- On the other hand, QUIC performs very good compared to the other two protocols under high RTT values especially when the bandwidth is low
- The Forward Error Correction of QUIC works well under high packet loss scenarios since those cases the Page Load Times of QUIC only increase acceptably whereas for the TCP based HTTP and SPDY the performance drops significantly
- SPDY performs very poorly in high loss environments due to HOL blocking
- Small object size favors QUIC and SPDY against HTTP due to multiplexing
- HTTP is the fastest protocol in case of high speed, high packet loss and high number of large objects.

V. Conclusion

QUIC (Quick UDP Internet Connections) is a very recent protocol developed by Google in 2013 for efficient transfer of web pages. QUIC aims to improve performance compared to SPDY and HTTP by multiplexing web objects in one stream over UDP protocol instead of traditional TCP. In this study, we



(a) (b) (c)

Fig. 5: Distribution of Page Load Times in such network conditions when QUIC clearly outperforms the other two protocols.

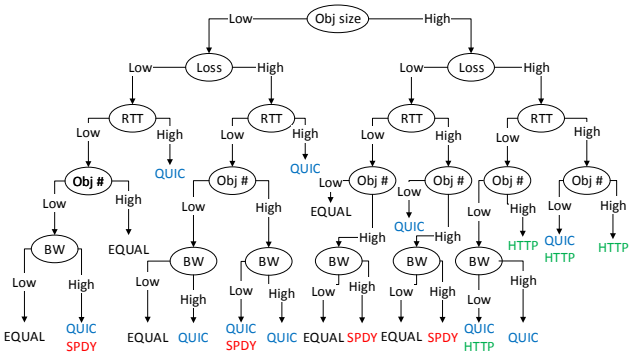


Fig. 6: Decision tree over the parameter space

presented a comparative analysis of QUIC, SPDY and HTTP, and identified the bottlenecks of their performance. We built a simple test environment using a laptop to download different web pages from Google Sites and used a shaper server to emulate different network conditions.

In more than 40% of the scenarios the experimental QUIC protocol improved page load time significantly, but our results also showed that HTTP can perform better than the two multiplexed protocols when downloading large objects. The tests also confirmed previous publications' results ([12], [13]) which said that a lossy network environment has a large negative impact on the performance of SPDY.

High round-trip time proved to be the condition, where QUIC could help the most. Due to mobile Internet connections (especially 3G) having large round-trip delay, we recommend that when the protocol exits the experimental status, it shall be implemented on servers of web pages with high mobile traffic and also be enabled by default in Google Chrome for Android.

High bandwidth is the only bottleneck that we could identify in QUIC's operation. This is mainly caused by the packet pacing mechanism implemented in the protocol: the goodput of a QUIC stream cannot reach the capacity of high speed links. Some network operators may also rate-limit UDP traffic based on security concerns and this could hurt QUIC's performance. The effective differentiation of web traffic over QUIC and potentially dangerous UDP traffic is a crucial task of the near future for both protocol developers and network operators.

Our future plan is to continue the performance evaluation study for more case studies and also following the upcoming new versions of QUIC.

Acknowledgments.: This work was supported by Ericsson. The authors would like to thank Géza Szabó and Sándor Rácz, Ericsson Research Hungary for their comments on this paper.

REFERENCES

[1] "SPDY Protocol - Draft 3," Retrieved: Feb., 2016. [Online]. Available: http://www.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3

[2] "Hypertext Transfer Protocol version 2 - RFC 7540." [Online]. Available: https://tools.ietf.org/html/rfc7540

[3] R. Hamilton, J. Iyengar, I. Swett, and A. Wilk, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2," [Online]. Available: https://tools.ietf.org/html/draft-tsvwg-quic-protocol-02

[4] I. Grigorik, "Making the Web Faster with HTTP 2.0," *Communications of the ACM*, vol. 56, no. 12, pp. 42–49, Dec. 2013.

[5] "SPDY: An Experimental Protocol for a Faster Web," Retrieved: Feb., 2016. [Online]. Available: http://www.chromium.org/spdy/spdy-whitepaper

[6] J. Padhye and H. F. Nielsen, "A Comparison of SPDY and HTTP Performance," Microsoft Technical Report MSR-TR-2012-102, 2012.

[7] G. Podjarny, "Not as SPDY as You Thought," 2012. [Online]. Available: http://www.guypo.com/technical/not-as-spdy-as-you-thought/

[8] G. White and D. Rice, "Analysis of SPDY and TCP initcwnd." [Online]. Available: http://tools.ietf.org/html/draft-white-httpbis-spdy-analysis-00

[9] A. Cardaci, L. Caviglione, A. Gotta, and N. Tonellotto, "Performance Evaluation of SPDY over High Latency Satellite Channels," in *Personal Satellite Services*, 2013, pp. 123–134.

[10] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "Demystifying Page Load Performance with Wprof." in *Proc. NSDI'13*, 2013, pp. 473–485.

[11] J. Erman, V. Gopalakrishnan, R. Jana, and K. Ramakrishnan, "Towards a SPDY'ier Mobile Web?" in *in Proc. ACM CoNEXT*, 2013, pp. 303–314.

[12] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How Speedy is SPDY," in *Proc. NSDI'14*, 2014, pp. 387–399.

[13] Y. Elkhatib, G. Tyson, and M. Welzl, "Can SPDY Really Make the Web Faster?" in *IFIP Networking Conference*, 2014, pp. 1–9.

[14] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-host Congestion Control for TCP," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 3, pp. 304–342, 2010.

[15] S. Molnár, B. Sonkoly, and T. A. Trinh, "A Comprehensive TCP Fairness Analysis in High Speed Networks," *Computer Communications*, vol. 32, no. 13, pp. 1460–1484, 2009.

[16] J. Roskind, "QUIC Design Document and Specification Rationale,"

[17] J. Roskind, "QUIC - Multiplexed Stream Transport over UDP." [Online]. Available: http://www.ietf.org/proceedings/88/slides/slides-88-tsvarea-10.pdf

[18] I. Swett and J. Iyengar, "QUIC Loss Recovery and Congestion Control," [Online]. Available: https://tools.ietf.org/html/draft-tsvwg-quic-loss-recovery-01

[19] A. Gizis, "Taking Google's QUIC for a Test Drive," Retrieved: Feb., 2016. [Online]. Available: http://www.connectify.me/taking-google-quic-for-a-test-drive/

[20] G. Carlucci, L. De Cicco, and S. Mascolo, "HTTP over UDP: an Experimental Investigation of QUIC," in *ACM SAC'15*, 2015.

[21] S. R. Das, "Evaluation of QUIC on Web Page Performance," 2014.

[22] "QUIC Test Server," Retrieved: Feb., 2016. [Online]. Available: http://www.chromium.org/quic/playing-with-quic

[23] "Chrome HAR Capturer," Retrieved: Feb., 2016. [Online]. Available: https://github.com/cyrus-and/chrome-har-capturer