Is There a Case for Parallel Connections with Modern Web Protocols?

Jawad Manzoor[1], Ramin Sadre[1], Idilio Drago[2], and Llorenç Cerdà-Alabern[3]

[1]Université Catholique de Louvain, [2]Politecnico di Torino, [3]Universitat Politècnica de Catalunya

***Abstract*—Modern web protocols like HTTP/2 and QUIC aim to make the web faster by addressing well-known problems of HTTP/1.1 running on top of TCP. Both HTTP/2 and QUIC are specified to run on a single connection, in contrast to the usage of multiple TCP connections in HTTP/1.1. Reducing the number of open connections brings a positive impact on the network infrastructure, besides improving fairness among applications. However, the usage of a single connection may result in poor application performance in common adverse scenarios, such as under high packet losses. In this paper we first investigate these scenarios, confirming that the use of a single connection sometimes impairs application performance. We then propose a practical solution (here called H2-Parallel) that implements multiple TCP connection mechanism for HTTP/2 in Chromium browser. We compare H2-Parallel with HTTP/1.1 over TCP, QUIC over UDP, as well as HTTP/2 over Multipath TCP, which creates parallel connections at the transport layer opaque to the application layer. Experiments with popular live websites as well as controlled emulations show that H2-Parallel is simple and effective. By opening only two connections to load a page with H2-Parallel, the page load time can be reduced substantially in adverse network conditions.**

***Index Terms*—HTTP/2, QUIC, MPTCP, Performance, Measurements**

I. Introduction

The web has become an essential part of our daily lives. We see a continuous trend of migrating traditional applications to the cloud, e.g., Microsoft Office 365 and Google Apps. As a result, modern web content has become extremely complex. This complexity requires efficient web delivery protocols to maintain users' experience regardless of the technology they use to connect to the Internet and despite variations in the quality of users' Internet connectivity.

HTTP, which is the de facto standard protocol of the web was developed in early 1990s as a simple request/response protocol to deliver content over the Internet. Its first versions, HTTP/1.0 and HTTP/1.1, have inherent inefficiencies when dealing with modern web content. For example, they suffer from head-of-line (HOL) blocking, where responses must arrive sequentially, following the order of requests. As web pages were getting more and more complex over the years, these inefficiencies started to hurt Page Load Time (PLT).[1] Despite these limitations, HTTP/1.1 over TCP has maintained a dominant position for around 20 years due to well-known challenges in replacing popular Internet protocols.

[1]Page Load Time is the time from when a user fires a web page request (e.g., by clicking on a link) until the page is fully loaded by the browser.

Browser vendors have reacted to HTTP/1.1 inefficiencies throughout the years by deploying ad-hoc optimizations to speed up PLT. One such optimization is the opening of several persistent TCP connections towards each web server when retrieving pages. Browsers can issue requests in parallel in the multiple connections, reducing the effect of HOL blocking. As a side effect, they compete for resources with other applications in the network more aggressively. For example, while the transfer rate of a single TCP connection is limited by the small congestion window (cwnd) during TCP slow start, multiple connections sum up their cwnd, resulting in faster startup rates. The fierce competition among browser manufacturers has pushed browsers to open a large number of parallel connections in an attempt to speed up page rendering [1].

Only recently, with Google's development of SPDY and QUIC, new protocols to replace HTTP/1.1 have gained momentum. The successful deployment of SPDY over TLS has opened the way for the HTTP evolution, triggering the standardization of HTTP/2 [2]. HTTP/2 borrows many of SPDY's principles and solves several shortcomings of HTTP/1.1. In particular, HTTP/2 *multiplexes* requests in a single TCP connection, eliminating the HOL blocking bottlenecks. This feature has prompted the IETF to recommend clients to open a single TCP connection per host-port pair for HTTP/2 transactions [2].

QUIC (Quick UDP Internet Connections) is another promising protocol developed by Google that provides multiplexing, congestion control and security functionality similar to HTTP/2, TCP and TLS, respectively, on top of UDP. It implements several optimizations including 0-RTT connection establishment, where clients can start repeated sessions with a known server without a three-way handshake, improved congestion control and better RTT estimation and loss recovery mechanism than TCP [3].

Recent studies have tracked the adoption of HTTP/2 and QUIC, showing not only a manifold increase in their usage, but also real performance gains [4], [5], [6]. However, both protocols use a single connection by design, which may result in poor application performance under adverse network conditions, in particular if different protocols compete for resources. For example, HTTP/2 is known to be particularly vulnerable in WiFi networks with high random packet losses. Equally, whereas QUIC uses a different congestion control strategy that reduces the effects of random packet losses, the implications

ISBN 978-3-903176-08-9 © 2018 IFIP

of QUIC's use of a single connection – e.g., during congestion in load-balanced links – are not fully understood yet.

In this paper we investigate the performance of browsing using modern web protocols in some adverse network scenarios. We use both active measurements with live websites and emulations in a testbed. We first confirm that HTTP/2 (and to a lesser extent QUIC) suffers more than HTTP/1.1 with multiple TCP connections in the tested scenarios. The use of a single connection partly explains the results. We then test whether adopting multiple TCP connections with HTTP/2 helps in mitigating the problems. We call this practical solution H2-Parallel, and implement it by modifying the source code of the Chromium browser. We compare H2-Parallel with HTTP/2 using a single TCP connection, HTTP/1.1 (both cleartext and encrypted) using multiple TCP connections, QUIC over a single UDP connection, as well as HTTP/2 over Multipath TCP (hereafter called H2-MP). The latter creates parallel connections at the transport layer opaque to the application layer. Note that HTTP/2 and QUIC always employ encryption by default.

Our experiments with popular live websites as well as controlled emulations show that H2-Parallel has some interesting advantages. It reduces PLT when compared to HTTP/2 over a single connection. In a scenario with around 2% of packet loss, H2-Parallel with only two parallel connections reduces the average PLT of HTTP/2 by 55%, and practically makes the performance of HTTP/2 similar to what is obtained by HTTP/1.1 with several parallel connections. Although QUIC is not affected by packet losses as severely as HTTP/2 over TCP thanks to its new congestion control strategy, we show that QUIC can also benefit from the use of parallel connections in some scenarios. Finally, H2-Parallel and H2-MP present similar performance with different practical trade-offs.

We make the following contributions:

- We identify scenarios that challenge HTTP/2 and QUIC performance, namely (i) packet losses in wireless networks and (ii) congestion in ISP networks with load balancers, a scenario not addressed in prior work yet;
- We implement *H2-Parallel*, a Chromium-based user agent that fans out HTTP/2 requests to a destination over multiple TCP connections;
- We compare *H2-Parallel* against the major web protocols. Our results differ from previous works by (i) including all relevant web protocols, instead of only a subset of them; (ii) considering real websites and browsers instead of simplistic downloads or TCP transfers, thus giving a view on how users perceive performance while browsing; (iii) testing the latest protocol versions (e.g., QUIC 39).
- We evaluate whether the protocols are fair to one another when competing for bandwidth and find that *H2-Parallel* and QUIC behave similarly for long transfers.

To simplify the terminology, in the remainder of this paper, we will refer to the non-encrypted (i.e., cleartext) version of HTTP/1.x as **H1C**, to HTTP/1.x over TLS as **H1**, and to HTTP/2 over TLS as **H2**.

The rest of the paper is organized as follows. Section II discusses the related work. In Section III we discuss scenarios in which H2 and/or QUIC may exhibit poor performance. We present our measurement methodology in Section IV, and discuss results in Section V. Section VI concludes the paper.

II. Related work

Zimmermann et al. [5] investigate H2 adoption. They show that around 12.5% of Alexa top-million domains provide full H2 support, with a 66% increase in H2 enabled domains between Sep 2016 and Jan 2017. They also study H2 performance, but in contrast to our work, they focus on the impact of H2 server push functionality, showing that some websites profit from the feature to speed up PLT.

Other works characterize the performance of websites according to the used HTTP versions. Zarifis et al. [7] explore the PLT differences between H1 and H2 using data collected from real users of the Akamai CDN. They find that in around 60% of the time H2 has lower PLT than H1. Varvello et al. [4] build a measurement platform to actively monitor H2 adoption by probing Alexa top-million websites. They show that around 80% of the websites adopting H2 improve PLT.

Saxcé et al. [8] compare the performance of H1 and H2. They clone the Alexa top-20 websites and find that, apart from network conditions, PLT depends on the website structure and content. Erman et al. [9] focus on mobile browsing. They measure PLT for the top-20 Alexa websites using SPDY and H1 proxies in a 3G network and find that SPDY performs poorly due to the large number of retransmissions and TCP backoff. Elkhatib et al. [10] reach similar conclusions by comparing the performance of SPDY with H1 in simulated networks. All these works however do not explore possible solutions for the scenarios where H2 performance degrades.

Wang et al. [11] perform experiments with synthetic pages and cloned pages (Alexa top-200). They propose a solution for SPDY inefficiencies by tuning TCP: increasing initial window, increasing receive window and reducing backoff rate in case of packet loss. This solution is not practical since making changes to TCP is hard and can take years to be widely deployed.

Recently, there has been a lot of interest in QUIC. Carlucci et al. [12] investigate QUIC (v. 21) on emulated network environments using synthetic pages. They report that QUIC performs worse than H1, but better than SPDY, with large web pages and 2% random packet loss rate. Without packet loss, QUIC performs better than H1 and SPDY for small and medium web pages, but worse for large pages due to the usage of only six parallel streams.

Megyesi et al. [13] test QUIC (v. 20) while emulating different values for bandwidth, delay and packet loss. They host four synthetic pages having different size and number of images. They show that, with packet loss, SPDY performs the worst, followed by QUIC and H1. In case of high bandwidth and large page size, QUIC's PLT is three times larger than H1 and SPDY. Kakhki et al. [14] show the root-cause for the problem, which prevented slow start threshold update (fixed in newer QUIC versions). The authors show that QUIC (v. 34)

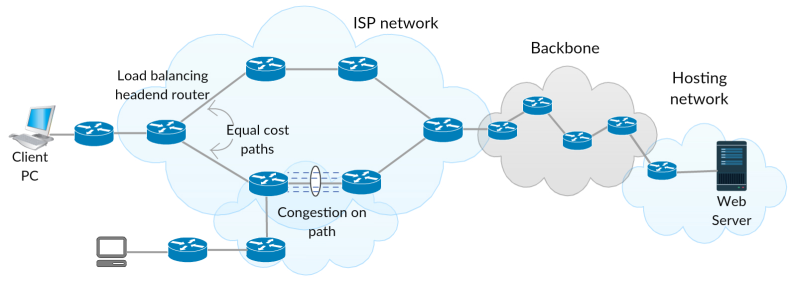


Fig. 1: Congestion on one of the load-balanced paths in ISP network

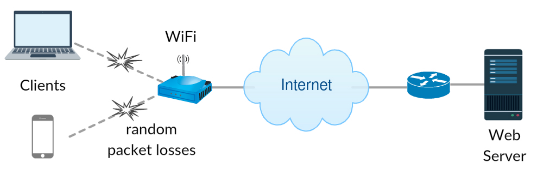


Fig. 2: Random packet losses in WiFi network

always outperforms H2, except with a very large number of small objects. QUIC (v. 34) is however unfair with other protocols, taking more than 50% of the bottleneck bandwidth when competing with 2 or even 4 TCP connections, a result that we will revisit with the newest version of QUIC.

In our prior work [15] we showed that some browsers arbitrarily create up to 6 connections towards a destination when using H2, but this was due to an issue only recently discovered by the developers [16] (fixed in Chromium v. 61), and not a performance improvement feature. In this paper we implement such a solution for H2 (i.e., *H2-Parallel*) and show its effectiveness.

Other studies have investigated the use of newer transport protocols such as MPTCP to alleviate the performance degradation of H2 and SPDY in presence of packet losses. Han et al. [17] provide the first measurement study of mobile web performance over MPTCP using SPDY and H1. They download 25 websites from Alexa top-100 list and measure PLT by combining LTE and WiFi with MPTCP. They show that MPTCP helps in mitigating performance penalties of SPDY under packet loss. In [18] the same authors provide a cost-benefit analysis of MPTCP in terms of improved user experience and energy consumption on mobile devices. The assumption in these studies is that clients are multi-homed – e.g., WiFi and LTE can be used simultaneously. Our work has a different scope: we check whether MPTCP has an impact on H2 performance even on single-homed devices, e.g., laptops or PCs with WiFi only. We will show that MPTCP can create multiple TCP connections and help in reducing the impact of packet losses and congestion in the network.

III. Considered Scenarios

We consider real-world scenarios where H2 and QUIC may perform poorly. These scenarios are described in the following and investigated in the next sections.

A. Random packet losses in WiFi networks

Random packet losses in WiFi networks are common under real-world conditions due to various factors like noise and the distance from clients to the access point. The impact of packet loss on the performance of H1, SPDY and H2 has been thoroughly studied in prior work [10], [11], [8], [17]. There is a consensus that H2 performs poorly compared to H1 when there is high packet loss rate. We investigate the extent to which other protocols suffer from similar problems, and propose a practical solution applicable to regular H2 over TCP.

B. Load balancers and congestion in ISP networks

A large percentage of the Internet traffic between source-destination pairs traverses multiple paths due to deployment of load-balancing routers [19], [20] in ISP networks. These routers split packets across multiple paths using techniques like Equal-Cost Multipath Routing (ECMP). Since traditional Internet measurement tools like traceroute may fail to identify these paths, alternative tools such as Paris traceroute have been developed to quantify multipath routing in the Internet. Augustin et al. [21] performed a large scale measurement using over 68 thousand destinations and showed that around 70% of paths between source and destination networks traverse a load balancer.

Three different load-balancing schemes exist: packet-based, destination-based and flow-based. Packet-based algorithms distribute all incoming packets evenly on all network paths, e.g., in a round robin fashion. Since different paths can have different delay, this approach may result in massive packet reordering and hence out-of-order delivery of the packets [22]. Therefore it is rarely used in practice. The destination-based scheme routes all traffic destined to the same host over the same path. This scheme can lead to uneven traffic distribution. The flow-based scheme is more popular. It defines a flow using different fields in the packet header, such as source and destination IP addresses, port numbers and protocol. All packets belonging to the flow according to the chosen definition are sent over the same path.

However, optimal load balancing is hard. Figure 1 shows a load-balancing headend router with two equal-cost paths in an ISP network where one of the paths is shared by traffic from other nodes. Despite evenly distributing traffic using ECMP on the headend router, the shared link may become overloaded and cause congestion. A number of large scale measurements [23], [24], [25], [26] show that congestion predominantly occurs in ISP networks and the described scenario happens quite often. In such a scenario H2 and QUIC may perform poorly as the browser will open a single connection, and that connection may be routed over the congested path. Hence they cannot exploit the underlying path diversity of the network. H1 on the other hand establishes multiple connections which may be distributed across the available paths by the load-balancing routers. Therefore, the performance is not significantly affected while downloading pages with H1 in such a scenario. This is an important scenario, yet it has not been investigated in prior studies. To quantify the extent of severity of this scenario by performing experiments in the Internet is an interesting topic for future work, but is out of scope for this paper.

C. Fairness among competing connections

In the recent years the traffic share of H2 and QUIC has rapidly increased and today, connections belonging to H1, H2 and QUIC co-exist in web traffic. These connections essentially compete for the bottleneck bandwidth. Maintaining fairness is very important for the network as unfairly taking bandwidth share from other protocols may lead to substantial performance degradation for some applications. While it is clear that H1 is unfair to H2 because of its aggressive use of connections, it is more interesting to see how QUIC competes with H1 and H2. In a recent study [14] on QUIC (v. 34) it was shown that QUIC is unfair to 2 and even 4 competing TCP connections. Since QUIC is evolving rapidly with major changes and improvements in each new version, we want to observe whether this behavior has changed in the most recent version (v. 39). Moreover, we want to verify how QUIC compares to our H2-Parallel implementation.

IV. Methodology

We now explain the design of H2-Parallel and H2-MP, our testbed, and measurement of PLT and cwnd of TCP and QUIC.

A. H2-Parallel

H2-Parallel is our Chromium-based user agent that fans out H2 requests over parallel TCP connections to mitigate the negative impact of a single connection on H2 PLT. Our objective is to verify that allowing the user agent to open parallel TCP connections for H2, similar to what most user agents do for H1, would improve the PLT.

Chromium browser maintains a single H2 session per domain, in accordance with the H2 specification. H2 sessions are tracked by a key consisting of the destination host–port pair. Each H2 session goes on a separate socket. In order to allow *two* TCP connections per domain, we have modified Chromium such that it stores two keys for each host-port pair. When issuing a new request, the state of the H2 session is controlled. First we check if we already have two keys for destination host-port pair of the current request. If not, we create a session with the new key and initialize the TCP connection. For each subsequent request, we call a function that returns one of the two available connections and use it for the request.

To keep the modifications as straightforward as possible, we have implemented a basic scheduler that assigns requests in a round robin fashion to one of the two connections. Note that requests assigned to the same connection are still multiplexed by H2. For the server, the two connections look like two regular H2 sessions from the same source IP. Despite this approach being simple, it distributes the requests fairly equally over the two connections.

TABLE I: Statistics of cloned web pages. The columns *HTML*, *CSS*, etc. show the number of objects of that respective type.

Website	HTML	CSS	JS	Image	Other	Total	Size (kB)
Baidu	1	1	1	6	1	10	50
Google	2	1	3	5	1	12	56
Live	2	2	2	2	0	8	262
Twitter	6	1	4	2	3	16	421
Wikipedia	1	1	2	20	1	25	441
Reddit	4	2	5	26	2	39	470
Yahoo	16	13	5	48	4	86	839
VK	4	1	14	3	1	23	920
Taobao	2	2	7	38	4	53	1 320
Instagram	3	1	7	25	1	37	1 409
QQ	15	6	19	115	6	161	1 728
Sohu	13	11	33	167	4	228	2 056
YouTube	8	3	5	113	20	149	2 911
Facebook	1	1	8	123	1	134	3 560
Amazon	5	2	14	41	2	64	3 723

TABLE II: Statistics of the pages in live websites

Website	Objects	Size (kB)	Domains	Connections		
				H2	H2-Parallel	H1
Google	17	286	1	1	2	4
Bing	32	421	1	1	2	2
Wikipedia	36	882	2	2	4	4
Mozilla	37	931	2	2	4	8
Poloniex	19	1 028	2	2	4	7
Paypal	64	1 415	2	2	4	12
Instagram	35	1 785	3	3	6	14
Blogger	61	2 061	2	2	4	11
Twitter	18	2 429	2	2	4	5
Facebook	86	4 266	2	2	4	12

B. H2-MP

H2-MP uses MPTCP to create parallel subflows to the servers to load a web page. We use H2-MP to compare the performance of creating parallel connections at transport layer versus application layer(as implemented by H2-Parallel). MPTCP is an enhancement of TCP that allows bandwidth aggregation and improved reliability by utilizing multiple paths simultaneously. It provides the same socket interface as TCP and spreads the data across several subflows without requiring applications or upper-layer protocols to be aware of the multiple paths. An MPTCP connection is initiated with the usual TCP 3-way handshake over one path. The handshake however includes a `MP_CAPABLE` message in the options field of the `SYN`, `SYN/ACK` and `ACK` packets. Further (sub-)flows can be added to the MPTCP session by sending `MP_JOIN` in the option field of additional 3-way handshakes regardless of the path used to open the flow.

We use stable release v0.91 of MPTCP and use the *ndiff-ports* path manager with the number of subflows set to 2, which creates two subflows between the same pair of IP-addresses by modifying the source port. We set the *default* scheduler which starts by sending data on the subflow with the lowest RTT. When its cwnd is full, it sends data on the subflow with the next lowest RTT. This is the recommended scheduler as it is known to provide the best performance. Hence two TCP connections are established between client and server without any modification of the browsing applications and having a complete view of the state of the connections at the transport layer. However, it requires MPTCP-compatible network stacks in the client and in the server.

C. Mininet testbed setup

We use Mininet [27] version 2.3 to emulate the three scenarios described in Section III. Mininet emulates a large network comprising multiple hosts, links and switches running real kernel and application code. We run Ubuntu Linux kernel 4.1.38 with the stable release v0.91 of MPTCP on the client and server and use Cubic congestion controller on both sides. We use Chromium browser version 60 on the client which supports H1, H2 and QUIC(v. 39). The server node hosts H2O web server[2] which provides an open-source implementation of H2, and quic-go web server which is an implementation of the QUIC protocol in Go[3]. We use Linux's Traffic Control *(tc)* and Network Emulation tools to configure network path characteristics such as bandwidth, delay and packet loss.

1) Emulating ECMP and congestion: For the scenario of ECMP with congestion, we emulate a typical home network shown in Figure 1 where a client's home router is connected to the headend router of an ISP with a 10 Mbps link [28]. The link from the ISP to the web server is configured with 1 Gbps bandwidth capacity. The headend router at the ISP performs load balancing using two paths. We emulate the congested bottleneck link in the lower path by generating traffic on it with 90% of its bandwidth capacity using iPerf with 8 connections.

We use flow-based routing in the load-balancing router for the reasons explained in Section III-B. Flow-based ECMP routing is not available in the latest MPTCP-capable Linux kernel that we use in our experiments. Therefore, we build a custom Linux kernel and implement a flow-based routing algorithm where the next hop is selected by hashing the flow 5-tuple, i.e., source address (SA), destination address (DA), source port (SP), destination port (DP), and protocol type (PT) of a connection. Our hash function H is defined as

$$H = SA \oplus DA \oplus SP \oplus DP \oplus PT$$

where $\oplus$ is the bitwise XOR function. We calculate H mod 2 to select either the first path or the second path.

This design avoids any informed decision at the router on how the multiple flows of a single H1, H2-Parallel or MPTCP session are routed through the paths. For instance, the MPTCP or H2-Parallel flows may all take the congested or the non-congested paths during emulations. Obviously, each protocol will react to path choices differently. For instance, MPTCP is able to detect congestion and move traffic to non-congested paths, if at least one subflow is routed to the non-congested path. H2-Parallel instead will blindly schedule requests on the multiple connections.

2) Emulating random losses in WiFi: We emulate the network shown in Figure 2. The WiFi link has 7 Mbps bandwidth and 50 ms delay representing realistic network conditions based on large-scale measurement study [29] and also used in prior studies [17], [18]. We perform tests without and with packet loss in the WiFi link. For the latter, we inject random packet losses using *netem*. We use 2% packet loss rate as suggested in prior work [11], [12], [13]. We also perform experiments using other loss rates but the results are not shown due to space limitation.

D. Measuring page load time

We have selected 15 websites from Alexa's top 100 list and downloaded their landing pages or other publicly available pages onto the H2O server. The selected websites are a mix of social networking, online shopping, news and search. The main characteristics of the cloned pages are summarized in Table I. Chromium browser is used on the client to load the pages from the server. We configure dnsmasq [4] on the client to ensure that all hostnames resolve to the IP address of the server and do not leave the testbed. We have also selected 10 popular H2 enabled websites for *live* experiments listed in Table II. The key requirement for this selection is that *all* of the content must be delivered by the server using H2.

To automate the page loading we create a script that uses *Chrome-HAR-capturer*[5] to connect to the browser via its remote debugging API and load each page multiple times with cold cache. When the experiment ends, an HTTP Archive (HAR) file is created, containing detailed performance data. Our script parses the HAR file, extracts PLT for each of each run and calculates the arithmetic mean of all runs. We load the web pages using H1C, H1, H2, H2-Parallel, H2-MP and QUIC.

E. Measuring cwnd changes

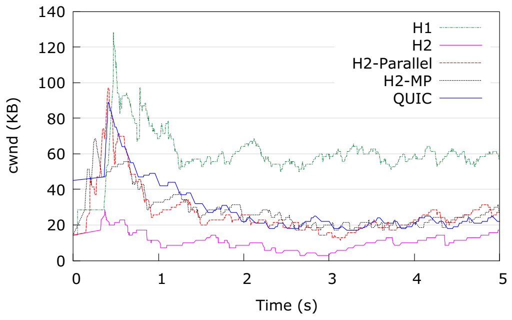
We monitor the changes in the cwnd size for TCP using the *tcpprobe*[6] module. In case of H1 we calculate the sum of the cwnd sizes of all individual connections. In case of QUIC we instrument the source code of quic-go web server to collect logs that allow tracking of the cwnd size on each ACK.

V. Measurement Results

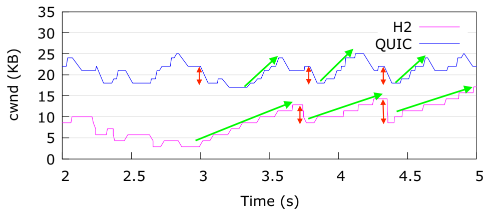
A. Impact of packet loss on single connection

We start our experiments by determining the impact of packet losses on the performance of various protocols by monitoring changes in cwnd size while loading a web page.

[2]https://h2o.examp1e.net/
[3]https://github.com/lucas-clemente/quic-go
[4]http://www.thekelleys.org.uk/dnsmasq/doc.html
[5]https://github.com/cyrus-and/chrome-har-capturer
[6]https://wiki.linuxfoundation.org/networking/tcpprobe



(a) Comparison of cwnd of all protocols



(b) 3-second zoom comparing cwnd of H2 and QUIC

Fig. 3: Timeline showing the impact of packet losses on congestion window size during a page load

We perform an experiment where we host a static web page comprising several JPG images on our web server and load it on the client using Chromium browser via H1, H2, H2-Parallel, H2-MP and QUIC. We configure the bottleneck link with 7 Mbps bandwidth, 50ms RTT, 45 kB buffer size and inject 2% random packet losses into the network path using *tc* and *netem*. We log the changes in cwnd sizes.

Figure 3a shows a 5-second zoom of the cwnd size comparison of the protocols. We can see that H1 has a much higher cumulative cwnd size than others. This is because browsers usually maintain up to 6 parallel connections to each server for H1 transfers and only some connections may be affected by random losses at a time. So the sum of cwnd size of all individual connections remains high. On the other hand, since browsers establish only one connection to the server when using H2, the same connection keeps experiencing the losses resulting in continuous reduction of cwnd. This limits the cwnd to a very small size which in turn results in very low throughput and long page load time. We can also see that QUIC has almost twice the size of cwnd as compared to H2 although both use a single connection and face the same rate of packet losses. There are several reasons for this behavior. First, note that the initial window size in QUIC is around 45 kB (32 segments) while for H2 (and others based on TCP) the size is around 15 kB (10 segments). Second, QUIC has an advantage over H2 because it uses its congestion controller to emulate the behavior of two TCP connections over UDP (in QUIC version 39). In other words, in the event of packet loss the cwnd is reduced at half the rate of H2, depicted by red double-headed arrows in Figure 3b. Finally, QUIC recovers more quickly from packet losses than H2, which can be observed by a steep upslope as shown by green arrows in Figure 3b.

In case congestion in network with load-balancers, the packet losses are dynamic, however, the results that we observe are almost the same and are not shown here. In such a scenario, the single connection of H2 and that of QUIC suffers losses when it is on the congested path, while H1, H2-Parallel and H2-MP are able to use the non-congested path simultaneously for part of their traffic.

B. *Impact of packet loss in WiFi networks*

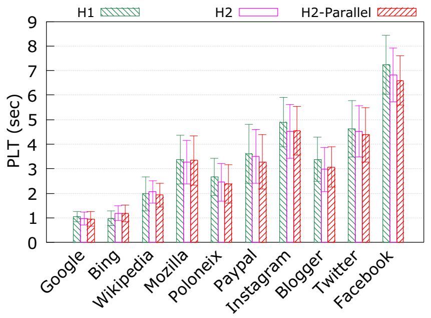
1) Packet loss in live websites: When visiting a live website, several aspects influence the PLT perceived by the user, e.g., delays of DNS queries or of server-side operations to prepare the content. Furthermore, a live website might consist of objects coming from different domains (e.g., due to domain sharding) that are not delivered from the same host.

We study the performance of H1, H2 and H2-Parallel with live websites. We do not perform experiments with MPTCP since none of the top websites support it at the server side. We also skip QUIC as it is only available for Google services and we cannot compare it with other websites. The only parameter that we will vary in our experiments with live websites is the random packet loss rate. To this end, we have selected 10 H2 enabled websites. The page characteristics are shown in Table II, where we give the number of objects per tested page, the total size in kBytes and the number of domains delivering the objects. The latter determines the number of TCP connections opened by the browser (also shown in the table). For each domain Chromium opens one TCP connection with H2, two with our H2-Parallel implementation, and up to six connections with H1. Note that the pages loaded from live websites are not exactly identical to those we cloned onto our testbed.

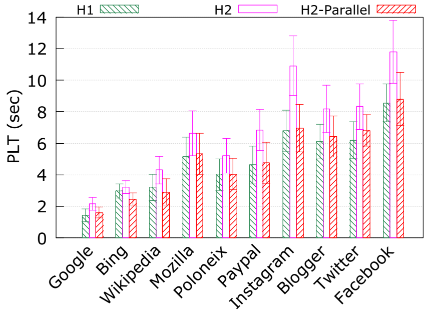
The RTT to the web servers is in the 15–165 ms range. We load each page 15 times with an empty cache. Figure 4 shows the average PLT (and its standard deviation) when using H1, H2 and H2-Parallel without packet loss and with 2% packet loss.

Focusing on Figure 4a, we notice how the performance of H2 is mostly better than H1. Whereas differences are not extremely large, these results are significant if we consider the number of TCP connections opened by the browser for each protocol (see Table II). Our implementation of H2-Parallel achieves similar performance as H2 when network conditions are good, although a small overhead for opening and managing the extra TCP connections are visible in some cases.

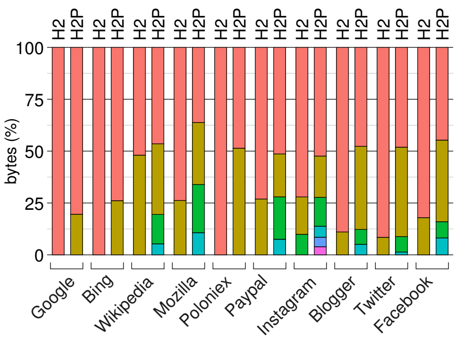
Obviously, the PLT increases significantly when packet loss is introduced – see Figure 4b (note the different scale of the y-axes). However, the increase of PLT with H1 is less pronounced than with H2, thanks to the use of multiple TCP connections by the former. H2 suffers severely under the packet loss. We notice how the PLT for Facebook reaches almost 12 s on average for H2, whereas it is around 8.5 s for H1 with 2% packet loss. The figure also shows that H2-Parallel achieves similar performance to H1 thanks to its second TCP connection. We are able to achieve 53% reduction in PLT on average for all websites by using H2-Parallel.



(a) Average PLT and std dev without packet loss

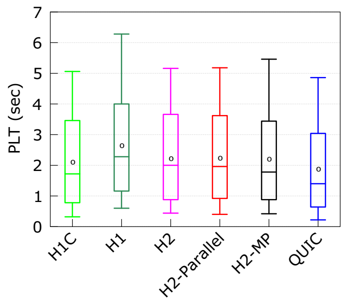


(b) Average PLT and std dev with 2% packet loss

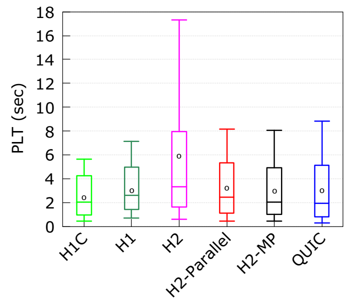


(c) Percentage of data transferred on each connection with H2 and H2-Parallel (H2P)

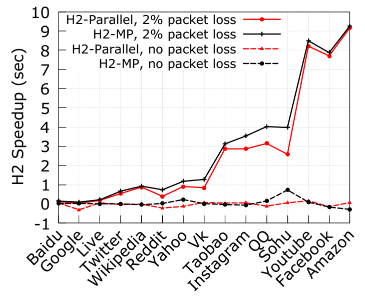
Fig. 4: Performance comparison of live websites with and without packet loss



(a) WiFi with 0% packet loss rate

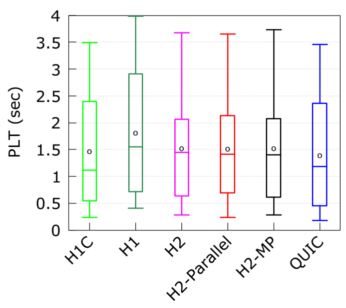


(b) WiFi with 2% packet loss rate

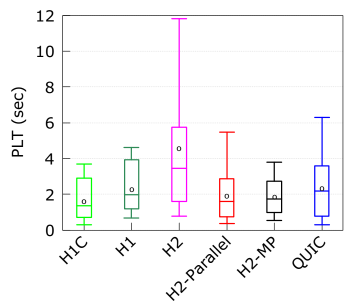


(c) H2 speedup against a single connection

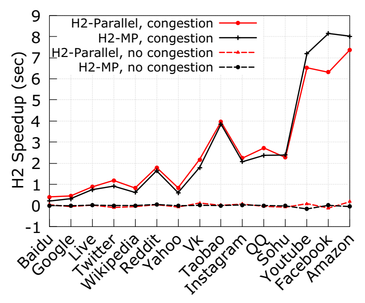
Fig. 5: Emulation of WiFi network. Note differences in y-axes.



(a) No congestion on any path



(b) Congestion on one path

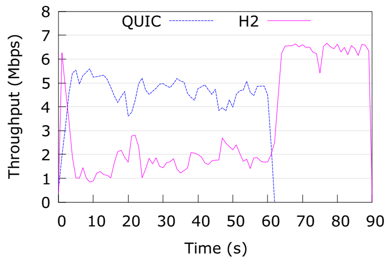


(c) H2 speedup against a single connection

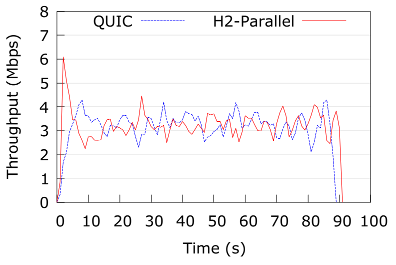
Fig. 6: Emulation of ECMP routing in the network. Note differences in y-axes.

Figure 4c shows the number of connections established with the server to load a website (depicted with different colors) and the percentage of bytes transferred with each connection (depicted by the size of the bar of each color) using H2 and H2-Parallel. We can see that with H2, while in some cases multiple connections are established due to multiple domains on the server side, most of the data (83% on average for all tested websites) is still transferred using only one connection. In case of H2-Parallel, a single connection carries 52% of traffic on average, thus distributing the load more evenly across multiple connections and reducing the probability of a single connection experiencing packet losses repeatedly.

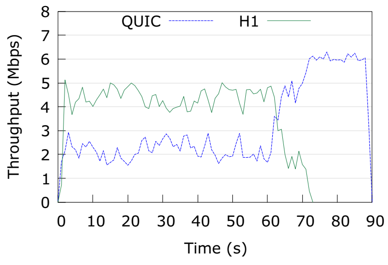
2) Packet losses in emulated environment: We perform emulations in a controlled mininet environment for reproducibility of results. In this experiment we measure the effect of random packet losses on the performance of the different web protocols using cloned web pages. Since the web server is under our control we can test QUIC and MPTCP and compare them with other protocols for the same pages, which was not possible with live websites. We load each page 30 times with empty cache using automated scripts. Note that the performance of H1 and H2 in scenarios with packet losses in WiFi networks has been studied in [10], [8], [9], [11], [17]. We confirm results from the previous works and evaluate to what extent H2-Parallel and H2-MP improve performance.



(a) QUIC competing with H2



(b) QUIC competing with H2-Parallel



(c) QUIC competing with H1

Fig. 7: Comparison of fairness of bandwidth share among competing connections of different protocols

In the following, box-whisker plots show the arithmetic mean (small circle), the median (middle horizontal line), the first and third quartiles (upper and lower box edges) and the minimum and maximum (whiskers) of the PLT over all 15 web pages. In our experiments, minimum and maximum will typically be several seconds apart. This is due to the large difference in characteristics of the selected web pages, i.e., small web pages like Google have very small PLT (represented by the lower end of the whisker) while large web pages like Facebook or Amazon have very large PLT (represented by the upper end of the whisker).

Figures 5a and 5b show plots of the PLT measured over all web pages without and with 2% packet loss, respectively. As expected, H2 performs better than H1 when there is no packet loss, and H2-Parallel and H2-MP do not show significant differences under good network condition. However, *with* packet loss, H2's PLT increases greatly while the other protocols see only a moderate increase by around 20%. Again, H2 is affected the most due to the use of a single TCP connection by the browser. Another disadvantage using H2 is that when there is a packet loss event, all streams get stalled until packet recovery due to the in-order delivery guarantee of TCP. Using QUIC, only the stream related to that packet gets blocked while others keep functioning normally. QUIC also maintains larger cwnd size as compared to H2 as shown in Figure 3a. Due to these reasons its performance is not affected as severely as H2.

Both H2-Parallel and H2-MP are able to reduce the performance penalty of packet losses and achieve a performance similar to H1 by increasing the cumulative cwnd size.

Figure 5c shows the average speedup (in seconds) that H2-MP and H2-Parallel achieve relative to regular H2 for each tested website (sorted by their size, with the smallest on the left). It can be seen that the speedup relative to regular H2 with packet loss is particularly pronounced for large web pages.

C. *ECMP and network congestion in emulated environment*

We now emulate the ECMP scenario with Mininet using H1C, H1, H2, H2-Parallel, H2-MP and QUIC. For each considered protocol, the client loads each web page 30 times with an empty cache. Remember that we do *not* actively control how the multiple flows are load-balanced in the available paths to emulate realistic scenarios. That is, in some experiment rounds, the multiple MPTCP or H2-Parallel flows may both take the congested or the non-congested path by chance. We can see in Figure 6a that, without congestion, H2 performs slightly better than H1 thanks to its various optimizations and new features. Not a surprise, H1C is faster than H1 because of the TLS overhead in the latter. Using two connections (H2-Parallel and H2-MP) in good network conditions brings no noticeable advantage while QUIC performs slightly better than others in the mean and median case.

However, the situation changes drastically in the presence of congestion. H2's PLT shoots up, with some pages taking as much as 12 s on average and up to 20 s in the worst case (not shown) to be fully loaded. H1C, H1, H2-Parallel and H2-MP are only slightly affected thanks to the parallel connections, which may be routed in the two available paths. In fact H2-MP performs the best as it can route the traffic away from the congested path on the fly and move it to the good path, which is not possible with any other protocol. QUIC is not affected as severely as H2 because its congestion controller reduces the cwnd size less aggressively when dealing with packet losses due to congestion. However, it still performs worse than H2-Parallel and H2-MP because in many cases it cannot take advantage of the non-congested path due to the use of a single connection. QUIC is 34% slower than H2-Parallel and H2-MP on average for medium and large websites but the situation is different in case of small websites. QUIC loads small websites quite fast due to 0-RTT connection establishment, and even with a single connection it performs slightly better than H2-Parallel and H2-MP. In fact for small websites creating parallel connections doesn't provide much benefit because most of the data is already transferred on the first connection before the second connection gets its turn.

Finally, in Figure 6c we can see that there is no speedup using H2-Parallel and moderate speedup using MPTCP when both network paths are congestion-free. When congestion is created on one path, both H2-Parallel and MPTCP achieve impressive speedups particularly for large pages.

Among the tested protocols, QUIC looks the most promising. Although it does suffer from performance degradation in the above scenario, we believe that using two connections with QUIC (similar to H2-Parallel) instead of emulating two connections using the congestion controller could improve QUIC performance in this scenario.

D. *Fairness comparison*

So far we have measured PLT of various protocols while running in isolation. Now we investigate their behavior while competing with one another. For this experiment we use two clients connected to two servers using the same 7 Mbps bottleneck link. We host a synthetic web page with large JPG images on the web servers and both clients load the web page at the same time, but using a different protocol. H2 and QUIC use a single connection, H2-Parallel uses two connections while H1 uses four connections to load the page. We measure the throughput of each protocol using tcpdump. Figure 7 shows the bandwidth share of competing connections per protocol pair.

In Figure 7a we can see that QUIC gets twice as much bandwidth share as compared to H2 as it emulates two connections using its congestion controller. It has been shown in a recent study [14] that QUIC version 34 is unfair to TCP even when competing against 2 or even 4 TCP connections. However, we do not observe such behavior in our experiments with QUIC version 39. Figure 7b clearly shows that H2-Parallel using 2 TCP connections and QUIC version 39 get an equal share of bandwidth, as expected from QUIC's congestion controller. Figure 7c shows that H1 using 4 TCP connections is more aggressive than QUIC and thus has an unfair advantage when competing with both H2 and QUIC.

VI. Conclusion

We presented a performance evaluation of modern web protocols in adverse real-world scenarios. We confirmed that H2 exhibits suboptimal performance in such scenarios and suffers from unfairness when competing with other protocols due to its use of a single TCP connection. Results showed that QUIC is not as severely affected, because it implements a congestion controller that emulates the behavior of two TCP connections over UDP.

We implemented and evaluated a solution to improve H2 performance, called H2-Parallel, which lets browsers open multiple TCP connections for H2 as they usually do for H1. We compared H2-Parallel with QUIC, H1, H2 and H2-MP, which relies on MPTCP to open parallel subflows. H2-Parallel has interesting advantages: it presents performance similar to QUIC, profits from parallel Internet paths similar to H2-MP, and requires changes only in the client browser thus easing deployment. Our experiments show that using only two connections with H2-Parallel provides significantly better performance than regular H2 in the tested scenarios, hence it avoids overloading the network with large number of connections like H1.

References

[1] B. Thomas, R. Jurdak, and I. Atkinson, "SPDYing Up the Web," *Commun. ACM*, vol. 55, no. 12, pp. 64–73, 2012.

[2] M. Belsche, R. Peon, and M. Thomson, "RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2)," 2015. [Online]. Available: https://tools.ietf.org/html/rfc7540

[3] A. Langley *et al.*, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the SIGCOMM*, 2017, pp. 183–196.

[4] M. Varvello, K. Schomp, D. Naylor, J. Blackburn, A. Finamore, and K. Papagiannaki, "Is the Web HTTP/2 Yet?" in *Proceedings of the PAM Conference*, 2016, pp. 218–232.

[5] T. Zimmermann, J. Rüth, B. Wolters, and O. Hohlfeld, "How HTTP/2 Pushes the Web: An Empirical Study of HTTP/2 Server Push," in *Proceedings of the IFIP Networking Conference*, 2017.

[6] J. Manzoor, I. Drago, and R. Sadre, "How HTTP/2 is Changing Web Traffic and How to Detect It," in *Proceedings of the TMA Conference*, 2017.

[7] K. Zarifis, M. Holland, M. Jain, E. Katz-Bassett, and R. Govindan, "Modeling HTTP/2 Speed from HTTP/1 Traces," in *Proceedings of the PAM Conference*, 2016, pp. 233–247.

[8] H. de Saxcé, I. Oprescu, and Y. Chen, "Is HTTP/2 Really Faster than HTTP/1.1?" in *Proceedings of the IEEE Conference on Computer Communications Workshops*, 2015, pp. 293–299.

[9] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan, "Towards a SPDY'ier Mobile Web?" *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, pp. 2010–2023, 2015.

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

[11] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How Speedy is SPDY?" in *Proceedings of the NSDI*, 2014, pp. 387–399.

[12] G. Carlucci, L. De Cicco, and S. Mascolo, "Http over udp: an experimental investigation of quic," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, pp. 609–614.

[13] P. Megyesi, Z. Krämer, and S. Molnár, "How quick is quic?" in *Proceedings of the ICC*, 2016, pp. 1–6.

[14] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, "Taking a long look at quic," 2017.

[15] J. Manzoor, I. Drago, and R. Sadre, "The Curious Case of Parallel Connections in HTTP/2," in *Proceedings of the CNSM*, 2016, pp. 174–180.

[16] (2017) Chrome opening up to 6 connections with H2. [Online]. Available: https://bugs.chromium.org/p/chromium/issues/detail?id=718576

[17] B. Han, F. Qian, S. Hao, and L. Ji, "An Anatomy of Mobile Web Performance over Multipath TCP," in *Proceedings of the ACM CoNEXT*, 2015, pp. 5:1–5:7.

[18] B. Han, F. Qian, and L. Ji, "When Should We Surf the Mobile Web Using Both Wifi and Cellular?" in *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges (ATC)*, 2016, pp. 7–12.

[19] Cisco. (2017) BGP Best Path Selection Algorithm. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html

[20] Juniper. (2017) Understanding BGP Multipath. [Online]. Available: https://www.juniper.net/documentation/en_US/junos/topics/concept/bgp-multipath-understanding.html

[21] B. Augustin, T. Friedman, and R. Teixeira, "Measuring Multipath Routing in the Internet," *IEEE/ACM Transactions on Networking*, vol. 19, no. 3, pp. 830–840, 2011.

[22] J. Bellardo and S. Savage, "Measuring packet reordering," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, 2002, pp. 97–105.

[23] Z. Cataltepe and P. Moghe, "Characterizing Nature and Location of Congestion on the Public Internet," in *Proceedings of the ISCC Symposium*, 2003, pp. 741–746.

[24] A. Akella, S. Seshan, and A. Shaikh, "An Empirical Evaluation of Wide-Area Internet Bottlenecks," in *Proceedings of the IMC*, 2003, pp. 101–114.

[25] A. Tachibana, A. Shigehiro, T. Hasegawa, M. Tsuru, and O. Yuji, "Locating Congested Segments over the Internet Based on Multiple End-To-End Path Measurements," *IEICE Transactions on Communications*, vol. 89, no. 4, pp. 1099–1109, 2006.

[26] J. Zhang, K. Xi, L. Zhang, and H. J. Chao, "Optimizing Network Performance using Weighted Multipath Routing," in *Proceedings of the ICCCN Conference*, 2012, pp. 1–7.

[27] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *Proceedings of the Hotnets Workshop*, 2010, pp. 19:1–19:6.

[28] S. Sundaresan, W. De Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè, "Broadband Internet Performance: A View from The Gateway," in *Proceedings of the SIGCOMM*, 2011, pp. 134–145.

[29] J. Sommers and P. Barford, "Cell vs. WiFi: On the Performance of Metro area Mobile Connections," in *Proceedings of the IMC*, 2012, pp. 301–314.