network

Article

Evaluation of Modern Internet Transport Protocols over GEO Satellite Links

Aljuhara Alshagri [1,*] and Abdulmohsen Mutairi [2]

[1] Institute of Next Generation Connectivity and Wireless Sensors, King Abdulaziz City for Science and Technology, Riyadh 11442, Saudi Arabia
[2] Department of Computer Engineering, King Saud University, Riyadh 14511, Saudi Arabia; mutairi@ksu.edu.sa
* Correspondence: aalshagri@kacst.edu.sa

Abstract: New versions of HTTP protocols have been developed to overcome many of the limitations of the original HTTP/1.1 protocol and its underlying transport mechanism over TCP. In this paper, we investigated the performance of modern Internet protocols such as HTTP/2 over TCP and HTTP/3 over QUIC in high-latency satellite links. The goal was to uncover the interaction of the new features of HTTP such as parallel streams and optimized security handshake with modern congestion control algorithms such as CUBIC and BBR over high-latency links. An experimental satellite network emulation testbed was developed for the evaluation. The study analyzed several user-level web performance metrics such as average page load time, First Contentful Paint and Largest Contentful Paint. The results indicate an overhead problem with HTTP/3 that becomes more significant when using a loss-based congestion control algorithm such as CUBIC which is widely used on the Internet. Also, the results highlight the significance of the web page structure and how objects are distributed in it. Among the various Internet protocols evaluated, the results show that HTTP/3 over QUIC will perform better by an average of 35% than HTTP/2 over TCP in satellites links specifically with a more aggressive congestion algorithm such as BBR. This can be attributed to the non-blocking stream multiplexing feature of QUIC and the reduced TLS handshake of HTTP/3.

Keywords: satellite networks; HTTP/1.1; HTTP/2; HTTP/3; QUIC; TCP congestion control; BBR; cubic

Citation: Alshagri, A.; Mutairi, A. Evaluation of Modern Internet Transport Protocols over GEO Satellite Links . *Network* **2023**, *3*, 451–468. https://doi.org/10.3390/network3030019

Academic Editor: Chin-Tser Huang

Received: 14 June 2023
Revised: 4 September 2023
Accepted: 12 September 2023
Published: 18 September 2023

1. Introduction

Satellite networks have become an important component of the Internet due to their wide coverage, high bandwidth and broadcasting capabilities. However, satellite communication links suffer from several limitations. The first primary limitation is the large propagation delay which is especially notable in geostationary (GEO) satellites that are located around 36,000 km above earth. The second limitation is the high bit error rate (BER) which is caused by the significant path loss in the satellite propagation channel. In modern satellite systems employing Adaptive Modulation and Coding (AMC), the link bit rate, and hence the frame size, varies with the channel conditions resulting in excessive segmentation and reassembly at higher layers. Another problem is the bandwidth asymmetry that exists between the hub gateway and satellite terminals which is caused by the nature of the shared return channel [1]. These problems at the link level greatly affect the performance of many Internet services running over satellite networks.

Over the years, there have been continuous efforts to overcome these problems at different layers of the Internet protocol stack which is illustrated in Figure 1.

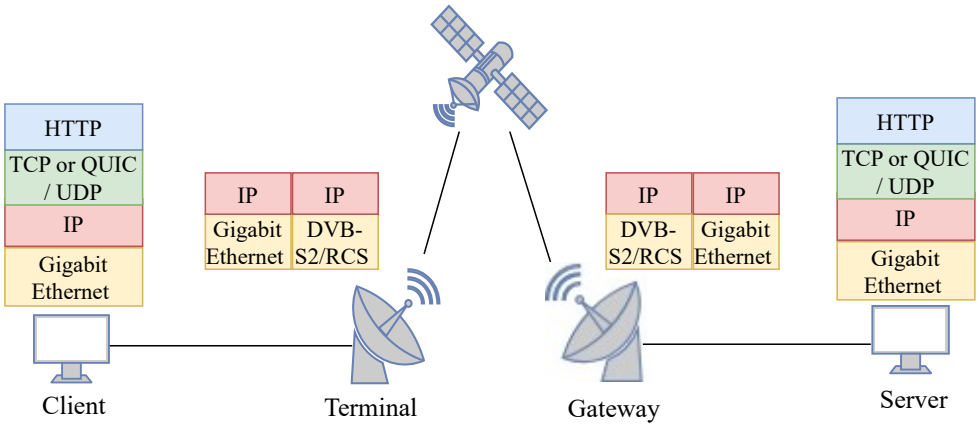

Figure 1. Typical satellite link connection.

At the transport layer, various TCP congestion control algorithms were designed to address the high latency problem of satellite links [2]. Also, performance enhancement proxies (PEPs) were widely used to improve TCP connection throughput using various, mostly proprietary solutions. At the application layer, newer forms of Internet protocols such as HTTP/2 over TCP and HTTP/3 over Quick UDP Internet Connection (QUIC) have been designed to address many of the shortcomings of legacy HTTP and TCP protocols [3].

Since satellite networks are usually deployed for long-term service often spanning many years, it is important to provide a cost-effective means for evaluating new and evolving Internet protocols over these networks. In this research work, we investigated the behavior of the modern HTTP variants such as HTTP/2 over TCP and HTTP/3 over QUIC running in modern satellite networks using a realistic emulation testbed. The testbed was built using state-of-the-art web browsers and web servers to mimic the true behavior of Internet protocols. The testbed can be used for optimizing the performance of these protocols as well as devising new algorithms to enhance their performance. Also, it will provide guidelines for different configurations of the protocol stacks and congestion control algorithms that could be used to improve Internet performance over future satellite networks.

In this paper, we present the findings of a detailed investigation of HTTP/3 over QUIC and HTTP/2 over TCP in GEO satellite links with modern congestion control algorithms and various web page structures. Specifically, the paper provides the following contributions:

- A testbed for GEO satellite links, with full DVB-S2/RCS emulation, running state of the art web servers and clients with complex web page structures that are otherwise extremely difficult to achieve via simulation;
- A performance comparison of HTTP/2 and HTTP/3 over GEO satellite links when using modern congestion algorithms such as Cubic and BBR;
- A web-level performance comparison of HTTP/2 and HTTP/3 when using complex web page structures with varying objects sizes and distribution. Here we employ First Contentful Paint (FCP) and Largest Contentful Paint (LCP) in addition to the page load time (Time) as our web-level user-perceived performance metrics.

The rest of the paper is organized as follows. A literature review of related work on evaluating Internet protocols in satellite networks is presented in Section 2. Next, Section 3 explains the details of the modern Internet protocols such as QUIC and HTTP/2. The evaluation testbed with its hardware and software components is described in Section 4. The evaluation scenarios and the results are presented in Sections 5 and 6, respectively. Finally, we summarize our main findings and conclusions at the end of the paper.

2. Related Work

The performance of TCP and QUIC protocols over satellite networks has been studied in numerous recent papers. The authors of [4] studied the QUIC protocol in a virtual

testbed and recommended using Performance Enhancement Proxy (PEP) to enhance its performance in satellite networks. However, their results are sensitive to the CPU limitation of a single machine hosting a virtual testbed. On the other hand, the authors of [5] compared PEP for both QUIC and TCP in terrestrial and satellite networks, and found that the performance of both TCP and QUIC is enhanced by using PEP.

Furthermore, the authors of [6] compared the performance of QUIC with TCP-PEP in satellite networks by using an experimental testbed. Using the throughput as a performance measurement, they found that the enhanced TCP-PEP outperforms QUIC in a lossy satellite link. They claim that QUIC needs a better loss recovery mechanism in the higher RTT and lossy networks. However, they did not specify which congestion control algorithm was used in their QUIC configuration.

In [7], the authors examined the performance of TCP-PEP and different QUIC implementations (QUIC Chromimum, QUIC Apache, QUICKLY and NGTCP) over an experimental satellite testbed. The congestion control algorithms were Reno and Cubic. They found that QUICKLY outperforms the other QUIC implementations over satellite networks but the performance of all QUIC implementations is degraded with packet losses.

The performance of QUIC with a recent congestion control algorithm called Bottleneck Bandwidth and Round-trip time (BBR) has been evaluated in [8], using an emulation testbed of satellite networks. Their preliminary evaluation indicated better results for QUIC with BBR compared to QUIC with Cubic. However, these papers did not investigate the perceived user-experience indicators which we are going to study in this paper.

Also, the effects of acknowledgment overheads in QUIC over satellite networks have been evaluated using an emulated satellite testbed in [9]. It compared TCP with two different implementations of QUIC: QUICKLY-IETF-27 and Chromium- IETF -26. It found that the return link traffic on QUIC is larger than TCP due to the overhead of acknowledgments in QUIC. However, the congestion controls used in the experiments are different since Chromium used BBR, while TCP and QUICKLY used New Reno. This had a significant effect on performance.

The authors of [10] evaluated, by simulation, the performance of HTTP/1.1 over TCP wave and standard TCP in terms of page load time with different web page sizes. They found that TCP wave performs better than standard TCP because it uses a burst-transmission paradigm instead of a window-based paradigm. Also, the performance of HTTP/2 over QUIC and TCP with Explicit Congestion Notification (ECN) was studied in [11] using an experimental testbed for an integrated space–terrestrial network. They evaluated the performance based on the page load time and the number of blocked messages. In their experiments, they found that HTTP/2 over QUIC with Cubic congestion control performed better than HTTP/2 over TCP with Cubic even with ECN enabled.

The authors of [12] examined the performance of multiple satellite operators (Avanti, Astra and Tooway) in an experimental testbed with multiple HTTP protocols. They studied HTTP/1.1 and HTTP/2 over TCP, HTTP/1.1 and HTTP/2 over UDP, and HTTP/3 over QUIC with Cubic as the congestion control. They found that HTTP/1.1 outperforms other HTTP protocols. They also concluded that Performance Enhancement Proxies (PEPs) are necessary in satellite networks to enhance performance. However, the authors used Chromium to test the HTTP protocols which creates six fixed connections by default for HTTP/1.1 over TCP. As a result, HTTP/1.1 over TCP outperformed the other HTTP protocols in that paper.

The authors of [13] addressed the challenges of using QUIC over satellite networks. The experiments were performed on an emulated satellite platform, where they tested HTTP/2 over TCP and HTTP/3 over QUIC with two web page sizes (11 KB and 5 MB). They found that QUIC with the default congestion control (Cubic) degrades the performance when web page size is increased because of the lack of PEP proxy supports in QUIC. However, they did not test TCP without PEP proxy to be compared with QUIC to ensure fairness in comparison.

Finally, the authors of [14] evaluated the performance of TCP-PEP and Google QUIC (GQUIC) with Cubic congestion control over satellite networks. They found that TCP-PEP outperforms GQUIC in terms of web page load time due to the splitting feature on TCP PEP proxies. However, they did not mention their network architecture and evaluation methodology. Also, the feature of multiplexing on HTTP is absent here since their web page profiles consisted of one object only (one stream).

On different, but related, developments, micro satellites with Small Optical Transponder (SoTA) have been developed and tested to provide low-orbit optical communication link to a ground station [15]. Optical LEO satellite links are considered a potential solution to overcome the limited spectrum and data rates of RF satellite links. Preliminary experimental results have been reported in [16] while a comprehensive simulation platform for LEO optical communication was developed in [17] which also reports the effect of many system configurations and parameters on link performance. However, as the technology is being developed most of the focus is currently on perfecting and evaluating the optical link itself and, hence, higher layer protocols such as HTTP are not yet considered.

All of the papers discussed above did not evaluate the performance of HTTP traffic with different web page structures over QUIC and TCP transport protocol with different congestion control algorithms. In this paper, we will evaluate the performance of HTTP/3 over QUIC, and HTTP/1.1 and HTTP/2 over TCP with recent congestion algorithms and various web page structures. Also, we will provide important user-level performance metrics which are not considered in the above papers. Moreover, we will analyze the behavior of HTTP traffic based on objects' size distribution within the web page.

3. Modern Internet Transport Protocols

Over the years, the Transmission Control Protocol (TCP) has been the primary transport protocol in the Internet. Applications relied on TCP to provide end-to-end reliable delivery of data while networks depended on it to manage bandwidth sharing and to control congestion. On top of TCP, the Hyper Text Transfer Protocol (HTTP) has become the de facto application-layer protocol to carry most Internet services. To overcome the limitations of TCP, various transport protocols have been proposed in the past. Some of them are enhancements of TCP, others are new layer-4 protocols and some others are encapsulated in UDP to minimize the impact on current Internet infrastructures. Of the latter, the most notable example is the Quick UDP Internet Connection (QUIC) which is backed by major Internet companies such as Google and others [18]. In this section, we will describe QUIC, and the new HTTP protocols such as HTTP/2 and HTTP/3 along with their recent congestion control algorithms.

3.1. Quick UDP Internet Connection

Quick UDP Internet Connection (QUIC) is a new UDP-based transport protocol that was designed to overcome several practical limitations of TCP. The current standardized protocol evolved from two versions: Google QUIC (GQUIC) and Internet Engineering Task Force QUIC (IETF-QUIC) [18]. The two versions have different implementations but they follow the same principles. Essentially, it is a byte-stream protocol that creates a reliable secure connection between the client and the server. Similar to TCP, it employs congestion control algorithms for detecting and recovering from packet loss to ensure error-free delivery. In the following, we summarize the main differences between TCP and QUIC to highlight the features of the new protocol.

First, TCP requires the full three-way handshake for connection setup before any data transfer and another handshake to establish the TLS parameters for the connection whenever TLS-level security is desired. In contrast, QUIC allows faster connection setup with a mandatory TLS session to create a secure connection with minimum handshake overhead. It can establish a secure HTTP session within 1-RTT and even 0-RTT as illustrated in Figure 2. Although QUIC could use 0-RTT to establish a new connection, it will be vulnerable to security issues such as IP spoofing, amplification attacks and denial-of-service

attacks. Therefore, it normally uses 1-RTT to reduce the handshake overhead compared to TCP without causing significant security risks [18].

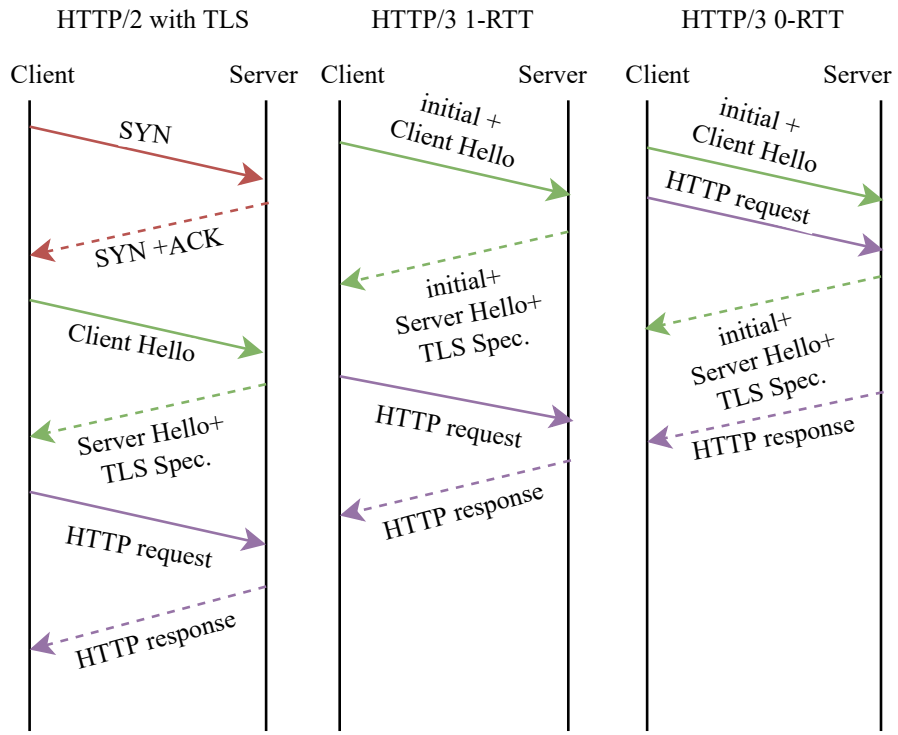

Figure 2. Example HTTP requests/responses in HTTP/2 with TLS, HTTP/3 with 1-RTT and HTTP/3 with 0-RTT.

Second, TCP suffers from the head of line blocking (HoLB) problem because it multiplexes multiple application-layer messages into a single layer-4 byte stream, in which a segment loss affecting one message will hold the other messages until the loss is recovered. QUIC avoids this problem by multiplexing the messages into one QUIC connection but each message is processed independently from the others. Thus, a segment loss in one message will not affect the others; only the affected message will be delayed until it recovers from the loss. For example, suppose application-layer messages A, B and C have their sequence of multiplexed segments as $A_1A_2B_1C_1C_2B_2A_3C_3$. If the client uses TCP, then a loss in segment B_1 will delay the processing of all subsequent segments at the receiver until it recovers from the loss. However, QUIC will delay only the segments belonging to message B while the remaining segments in messages A and C will be processed as they arrive in order at the receiver [18].

Third, TCP uses the 4-tuple (client IP address, client port, server IP address and server port) to uniquely identify the connection between the client and the server. So, when the client moves from one network to another, the TCP connection will be terminated because the connection ID is associated with the client IP address. A new handshake would be needed to restart the connection with the server which will restart any ongoing application-layer data transfer. QUIC solves this issue by using a unique connection ID (CID) in both endpoints to eliminate the dependency on IP addresses and ports when establishing the connection. This allows client handoff to another network, e.g., from WiFI to 4G, with a new IP address without resetting the QUIC connection because it is detached from the client IP address [18].

QUIC adds a new abstraction layer between applications and transport in order to be easier to evolve and deploy in existing Internet platforms. However, this abstraction comes with the cost of increasing the overhead of the protocol stack and encryption. Because the

implementation of QUIC is part of the applications, all TLS related encryption is done at the user space. This deprives QUIC of the faster kernel-space implementation and hardware acceleration that is available to TCP. Also, QUIC encrypts each segment individually which is slower than TCP+TLS which encrypts a group of segments (chunk) together [18,19].

3.2. Evolution of HTTP Protocols

The Hyper Text Transfer Protocol (HTTP) is an application-layer protocol that was originally designed in the 1990s to transfer web pages between Internet hosts. The first version of the protocol (HTTP/1.0) worked over TCP but it required a new transport-level connection for each HTTP request. It soon evolved into HTTP/1.1 which allowed reusing the same TCP connection to fetch multiple web objects such as HTML pages, embedded images, scripts, etc. Also, it allowed the client to send multiple requests back-to-back (pipelining) without waiting for the server's response for each request. This reduced the latency of downloading web pages especially under high traffic load. Since then, HTTP/1.1 had become the de facto web standard for more than 20 years with only minor enhancements [3].

The complex composition of today's web content has uncovered many of HTTP/1.1 shortcomings. First, the protocol uses text-based headers and messages which creates a lot of chatter between endpoints. This adds a significant overhead in the processing of these protocol messages especially in high-demand servers. Second, persistent TCP connections consume the server resources even when the clients are idle. Also, pipelining of requests often runs into the head of line blocking (HoLB) problem which occurs when a high priority object is delayed by a low priority one or when packets arrive out of order [20]. The lack of a multiplexing layer has caused web clients to open multiple parallel connections, usually up to eight, to the same server in order to speed up the download of the different objects that compose the web page. However, this decreases network efficiency because TCP does not share congestion control across multiple connections. On the server side, web providers often use optimization workarounds to improve HTTP/1.1 speed. For example, so-called *domain sharding* is used to distribute web contents across many server sites to reduce server load. Also, the concept of *image sprites*, which is a collection of multiple images aggregated in one file, is utilized to reduce embedded images' load times [20,21].

HTTP/2 was developed and finalized in 2015 to overcome the problems of HTTP/1.1 while maintaining the original HTTP semantics, messages and status codes [21]. At its core, HTTP/2 uses a more efficient binary encoding of the header fields instead of the original plain text headers of HTTP/1.1. In HTTP/2 terminology, a *stream* is a flow of bytes, with a unique identifier and a priority, which may carry one or more messages. A *message* is a logical HTTP message, such as a request, or a response, which consists of one or more binary *frames*. For example, one frame could be carrying the HTTP header and another frame could be carrying the message payload, etc. All communication is performed over a single TCP connection that can carry any number of bidirectional streams. Frames from different streams can be interleaved and then reassembled at the receiver using the stream identifier embedded in the header of each frame as illustrated in Figure 3. This ability to interleave frames from different streams allows HTTP/2 to multiplex several requests and responses in parallel, and hence avoids the HoLB issue in HTTP/1.1. However, it does not eliminate the HoLB problem at the transport layer, because all streams are eventually multiplexed in one TCP byte stream [21].

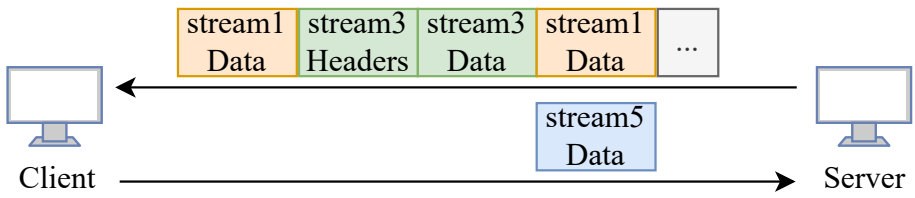

Figure 3. An example of stream multiplexing in HTTP/2.

There are other features in HTTP/2 such as stream prioritization and server push mechanisms which allow the server to open additional streams in order to send objects that may be requested later without being explicitly requested by the client. Finally, there are some TCP enhancements that were proposed in order to improve the performance of HTTP/2. For example, TCP Fast Open introduced a cookies-based mechanism in order to authenticate the client with the server from the first connection. Also, a larger initial window was proposed to reduce web object download times with a moderate cost in terms of increasing the congestion and loss rate [3].

HTTP/3 is a new major version of the HTTP protocol recently standardized in RFC 9114 [22]. Essentially, it is the same as HTTP/2 but runs over QUIC instead of TCP. With stream multiplexing and per-stream flow control, QUIC ensures reliability at the stream level and congestion control across the entire connection. Also, since QUIC multiplexes streams at the transport layer, it eliminates the head of line blocking problem at the transport layer which provides better HTTP performance. Moreover, running HTTP over QUIC reduces the time consumed in connection setup and supports early data transmission compared to HTTP/2 over TCP. Finally, it is worth noting that HTTP/2 over QUIC was recently renamed HTTP/3 by the Internet Engineering Task Force (IETF) in 2018 [19]. Figure 4 illustrates the difference in protocol stacks for HTTP/2 running over TCP versus HTTP/3 running over QUIC.

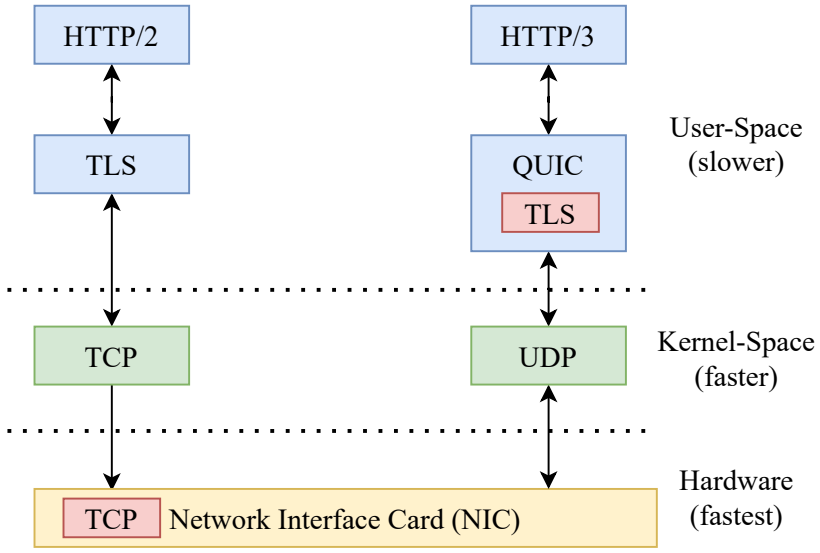

Figure 4. The protocol stack layering for HTTP/2 and HTTP/3.

3.3. Congestion Control Algorithms

There have been many congestion control algorithms developed for TCP over the years. Currently, the most widely used algorithm is TCP Cubic which is the default TCP in Linux, Mac OS and latest Windows operating systems [23]. Cubic is a loss-based algorithm that was proposed to overcome the low utilization problem of the standard New Reno TCP over high bandwidth-delay-product (BDP) networks. Another recent congestion control algorithm called the Bottleneck Bandwidth and Round-trip time (BBR) was developed and used by Google in 2016 [24]. Unlike loss-based algorithms, BBR aims to avoid congestion before it occurs by estimating the BDP and trying to manage the bottleneck's queue. The two algorithms are briefly explained in the following subsections.

3.3.1. Cubic

TCP Cubic congestion control grows its transmission window (*cwnd*) according to a cubic profile instead of the linear profile in TCP New Reno [23,25]. Its core idea is to fill the pipe very quickly to increase utilization in high BDP links but to prefer stability over utilization when operating near the congestion point. This means it gradually increments

cwnd as it moves toward the critical operating point that caused the last packet loss (concave region of the cubic function) and then increases cwnd more aggressively as it moves away from the critical operating point (convex region). This serves to maintain stability while improving network utilization.

The algorithm operates in three phases. The first phase is the slow start where the transmission rate is increased exponentially as in standard New Reno or as in other slow start algorithms such as HyStart until the cwnd reaches the slow start threshold (*ssthresh*). Then, it will move into the congestion avoidance phase which will increment cwnd every acknowledgment received according to a cubic function of the time elapsed since the last congestion event. When a packet loss is detected, the algorithm moves to the multiplicative decrease phase which decreases cwnd and ssthresh by the multiplicative decrease factor [23,25].

It is worth noting that the cubic function is disabled when operating in a TCP-friendly environment such as small BDP networks, where the standard congestion control performs better. Thus, the Cubic algorithm ensures that the throughput in these networks achieves at least the same throughput as in the standard New Reno congestion control [23,25].

3.3.2. BBR

The Bottleneck Bandwidth and Round-trip time (BBR) congestion control is based on the fact that the slowest link or bottleneck of the network path bounds transport performance. It determines the connection's maximum data-delivery rate and it is where persistent queues build up. In principle, BBR tracks the path's BDP by estimating the bottleneck bandwidth and the round trip propagation delay. Then, it tries to operate at the optimum BDP point without causing an excessive queue in the bottleneck, hence reducing the delay. This is in sharp contrast to loss-based algorithms such as Cubic which try to fill the bottleneck queue to deliver full link bandwidth at the cost of high delay and frequent packet loss [24,26].

BBR probes the maximum bottleneck bandwidth (BtlBW) and the minimum round-trip time (minRTT) and uses them as efficient unbiased estimators of the bottleneck bandwidth and RTT of the network path. The minRTT is estimated from RTT samples obtained from ACKs while BtlBW is estimated by tracking the delivery rate at the receiver which can be inferred from ACKs and the corresponding amount of data delivered. It then paces the packet-arrival rate at the bottleneck queue to its departure rate. This keeps the bottleneck queue near its minimum, maximizing delivered bandwidth while minimizing delay and loss. The primary control parameter in BBR is the pacing gain, which determines the sending rate as a function of BtlBW [24,26].

4. Evaluation Testbed

For a realistic evaluation of modern Internet protocols, we built a testbed running on dedicated computers and hosting the full stack software implementation of HTTP/2, HTTP/3 with Cubic and BBR congestion control algorithms. The testbed was realized using the OpenSAND Emulation platform which is an open-source project developed by Thales Alenia Space to provide demonstration, validation and performance evaluation of satellite network applications [27]. The emulator implements most of the new satellite transmission standards such as DVB-RCS2 and DVB-S2 including the protocol stacks as well as the DVB physical layer blocks for each uplink and downlink. It also supports the Adaptive Modulation and Coding (AMC) loop which is used by DVB-RCS2 transmitters to match the transmission rate with the channel quality. In addition, the emulator provides multiple attenuation profiles that depends on the orbit characteristics of the satellite [27].

The testbed consists of five computers deployed as the network depicted in Figure 5. Three machines host the three primary components of OpenSAND: terminal, satellite and gateway. We added a machine at the terminal side to run the client software and control the experiments, and another machine at the gateway side to host the web server. The specifications of all software and hardware are summarized in Table 1.

Table 1. Software and hardware specifications of the testbed computers.

Machine	CPU Speed	Memory	Processor	Operating System	Software
Client	1.9 GHz	16 GB	Intel core i7-8650U	Windows 10	FireFox Nightly V.90.0a1
Terminal	1.9 GHz	16 GB	Intel core i7-8650U	Linux kernel 5.4	OpenSAND V.5 Terminal
Satellite	2.3 GHz	8 GB	Intel core i3-7020U	Linux kernel 5.4	OpenSAND V.5 Satellite
Gateway	2.67 GHz	8 GB	Intel core i7 M620	Linux kernel 5.4	OpenSAND V.5 Gateway
Server	2.7 GHz	16 GB	Intel core i7-7500U	Linux kernel 5.4	OpenLightSpeed Web Server 1.7.12

There are two direct point-to-point links and a switched subnet in this setup. The first direct link is from client to terminal and the other is from server to gateway. The switched subnet consists of a bridge that connects the terminal, the gateway and the satellite via IP tunnels when the emulation is enabled. All traffic arriving at the terminal will be received and processed by the satellite emulator node. Then, the traffic will be transferred and processed by the corresponding blocks of the gateway. Therefore, we can produce a realistic satellite channel by adjusting the emulator link characteristics as needed in each experiment. The satellite link parameters used in the testbed are listed in Table 2.

Table 2. Satellite link parameters used in testbed.

Configuration Parameters	Value
Maximum link data rate	10 Mb/s
Delay for UL/DL	125 ms
Average round trip time	530 ms
Channel BER	Varies from 10^{-7} to 10^{-4}
Channel attenuation	20 dB (Clear-Sky in OpenSAND)

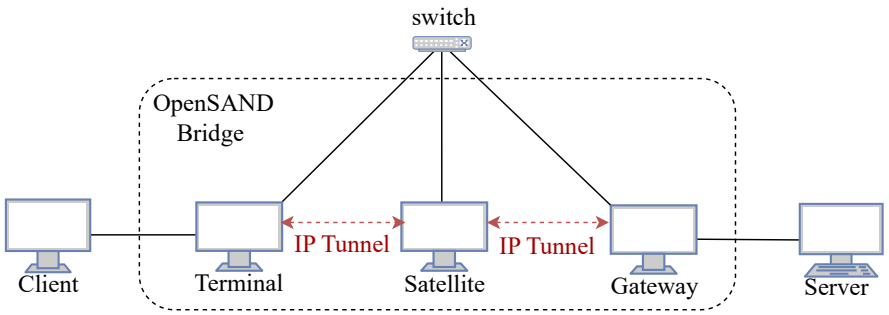

Figure 5. The satellite link structure used in testbed .

For the server machine, we deployed the Open LiteSpeed v1.7 which is a free web server provided by LiteSpeed technologies [28]. We chose this web server because it implements different HTTP protocols (HTTP/1.1, HTTP/2 and HTTP/3) and supports TCP and QUIC with various congestion control algorithms such as Cubic and BBR. On the client machine, we used Firefox Nightly v90.0 which was, at the time of evaluation, the most recent development version of Firefox because it supports HTTP/3 as well as HTTP/2 [29]. Also, it has been chosen among other browsers such as Opera, Chromium, etc. since it allows controlling the number of parallel TCP connections to the server. This is necessary in our testbed to ensure fair comparisons to QUIC in the experiments.

5. Experiments' Setup

The testbed experiments were carefully designed to examine the interaction of the new HTTP protocols with congestion control over a typical satellite link. As explained in Section 3, one of the important features of HTTP/2 and HTTP/3 is the use of parallel streams over one transport connection. This can only be evaluated using web pages of complex structures and can only be measured using detailed application-level performance

metrics. Therefore, we developed three HTML-based templates consisting of multiple embedded objects with different distributions of sizes and organization within the main HTML file. For the embedded objects, we produced eight JPEG images with varying sizes adapted in each HTML page to produce the desired total size of the web page. The distribution of objects in each page structure is listed in Table 3 and illustrated in Figure 6.

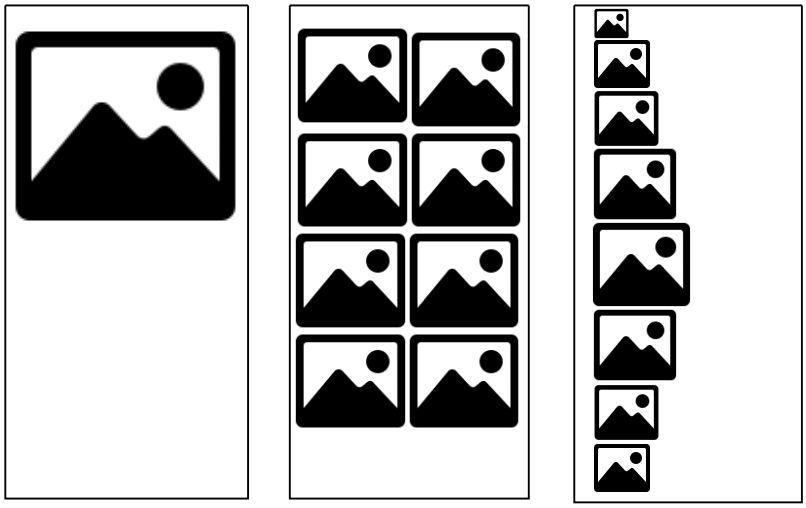

Figure 6. The distribution of objects in each page structure used in experiments.

Table 3. The size and organization of objects in each web page structure.

Total Page Size	Single-Object Web Page	Uniform Multi-Objects Web Page	Triangular Multi-Object Webs Page
100 K bytes	One object with size 100 K bytes	12.5 k bytes for all objects	3 k, 8 k, 14 k, 20 k, 22 k, 16 k, 11 k and 6 k bytes
500 K bytes	One object with size 500 K bytes	62.5 k bytes for all objects	20 k, 40 k, 70 k, 100 k, 110 k, 80 k, 50 k and 30 k bytes
1 M bytes	One object with size 1 M bytes	125 k bytes for all objects	5 k, 50 k, 135 k, 230 k, 270 k, 200 k , 100 k and 10 k bytes
2 M bytes	One object with size 2 M bytes	250 k bytes for all objects	10 k, 100 k, 270 k, 450 k, 540 k, 400 k, 200 k and 20 k bytes

The first template is a *single-object* web page which consists of only one embedded object whose size is matched to the required size of the web page. The second template is a *uniform multi-object* web page which contains eight objects of identical size such that their aggregate size is equal to the total web page size. The third one is a *triangular multi-object* web page where the eight objects are organized inside the page according to a special order of their size. The first four objects are ordered in increasing size up to the middle of the web page then the four remaining object are ordered in decreasing size down to the end of the web page. The reason for this arrangement is to investigate the effect of parallel streams in HTTP/2 and HTTP/3 on the visual rendering of complex web pages. With parallel fetching of the embedded objects in such complex web pages, the user's perceived browsing performance of the web page will depend on the position of the largest objects in the page. If the largest object is embedded at the end of the web page, then the page would seem to load faster than if it was embedded at the beginning of the page because the smaller objects will load faster with parallel streams. The behavior will depend on the distribution of the embedded objects within the page; hence, we designed the above templates as a simplified representation of more realistic complex web pages.

The testbed allows many evaluation scenarios each considering an HTTP variant running with a transport congestion algorithm. Each scenario can be run for different web

page structures with varying sizes and organization. In this paper, we report the results of three main experiments that were designed to examine specific aspects of the protocol stack's configuration. The first experiment studies the behavior of parallel streaming in HTTP/2 and HTTP/3 protocols in a satellite link. The second experiment examines the effect of Cubic and BBR congestion control algorithms on HTTP while the third one focuses on the effect of the web page structure on HTTP/2 and HTTP/3 in a satellite link. In all cases, we set all links' speeds to 10 Mbps and the satellite link RTT to 500 ms. Furthermore, the same scenarios were conducted over a direct LAN link between the client and server in order to provide some baseline for our evaluation. All experiments are summarized in Table 4. It should be noted here that one must take careful consideration of the default operating behavior of the web client and server, and their host computers to ensure isolated, controlled experimentation of HTTP protocols. Issues of web browser caching, server disk delays, network ARP delays and DNS lookup delays need to be isolated first before recording any statistics.

Table 4. Experiment scenarios.

Scenario	HTTP Version	Web Page Structure	Transport	Congestion Control
Effect of parallel streams	All	All	TCP, QUIC	BBR
Effect of congestion control	All	Uniform	TCP, QUIC	Cubic, BBR
Effect of web page structure	HTTP/2, HTTP/3	Uniform, triangular	TCP, QUIC	BBR

6. Results and Discussion

In this section, we will present and discuss the results of the three experiments described in the previous section. Every evaluation scenario was run three times resetting all testbed components after each iteration to ensure an identical operating environment. The results were then collected and averaged using client-side scripts that were developed specifically for the testbed. In the following, we first define the performance metrics that were collected in the evaluation and then discuss the results of each experiment.

6.1. Performance Metrics

In HTTP evaluation, the page load time (PLT) has been used as the primary indicator of layer-7 protocol performance. It summarizes many of the factors that are contributed by the network, transport and application protocols. However, due to the complex interaction and intrinsic features of the modern protocols such as HTTP/2, HTTP/3 and QUIC, the PLT does not always provide a sensible performance measure. Therefore, we need other performance metrics to study the interaction between clients and web servers as seen by the end user.

To assess the relative user experience, we used two additional user-level performance measures: the First Contentful Paint (FCP) and the Largest Contentful Paint (LCP) [29]. The FCP indicates the first response from the server and reveals what the user perceives first from the web page. The LCP expresses the perceived load speed as the arrival time of the most significant object (largest object). All the key performance metrics used in the evaluation are defined in Table 5.

Table 5. Performance metrics used in the evaluation.

Performance Metric	Description
Page Load Time (PLT)	The time when the entire page is loaded including text, images, scripts, etc. [29].
First Contentful Paint (FCP)	The time when the first object is loaded in the browser [29].
Largest Contentful Paint (LCP)	The time when the largest object is loaded in the browser [29].

6.2. The Effect of Parallel Streams in HTTP

The number of parallel streams used to fetch the objects in a web page can affect the load time significantly. Thus, we tested the download of multi-object web pages and compared it to single-object web pages of the same total size. The uniform structure was used for the multi-object web page and the total size of all web pages was set according to Table 3. The test used BBR congestion control for both the direct LAN link and the satellite link.

The average page load times (PLTs) of uniform multi-object web pages and of single-object web pages are shown in Figures 7 and 8, respectively. In multi-object web pages, depicted in Figure 7, we can see that HTTP/1.1 has the longest PLT compared to HTTP/2 and HTTP/3 because it operates with one stream only. This is because of the head of line (HoL) blocking at the application layer where one object blocks the transmission of the other objects until it is fully received. Thus, the multiple objects are transmitted sequentially in HTTP/1.1 over a single TCP connection. To overcome the HoL issue, HTTP/2 uses multiple streams, one stream per object, where all objects are transmitted in parallel to prevent HoL blocking at application layer. This significantly reduces the PLT compared to HTTP/1.1 but it still suffers from HoL blocking at the transport layer because all HTTP/2 streams are multiplexed into a single TCP connection. On the other hand, HTTP/3, which relies on QUIC non-blocking multiplexing at the transport layer, produces much smaller PLTs, about 55% less than HTTP/1.1 and 30% less than HTTP/2.

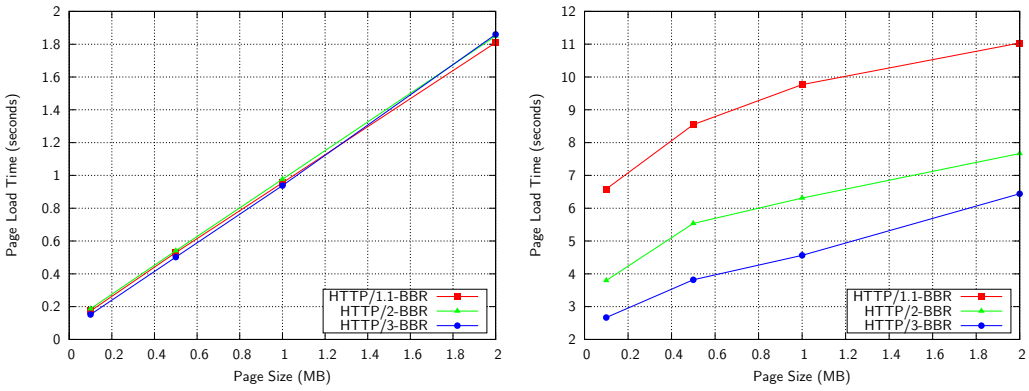

Figure 7. Average PLT of uniform multi-object web page with BBR congestion control. (**a**) Direct LAN (RTT= 1.08 ms). (**b**) Satellite link (RTT= 530 ms).

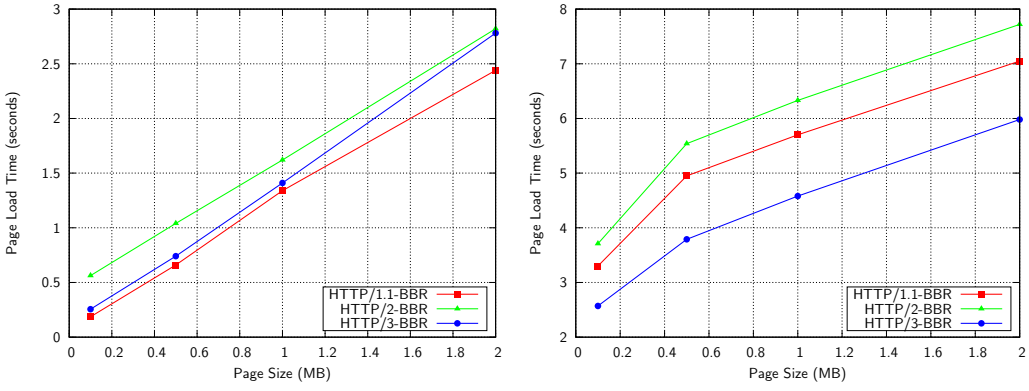

Figure 8. Average PLT of single-object web page with BBR congestion control. (**a**) Direct LAN link (RTT= 1.08 ms). (**b**) Satellite link (RTT= 530 ms).

The situation is almost reversed when the page consists of a single object only where HTTP/2 performed worse than HTTP/1.1 as shown in Figure 8. This is because stream multiplexing is not applicable in this scenario and we are left with the bigger protocol overhead of HTTP/2, e.g., from TLS security and bigger header size, which increase the page load time. However, HTTP/3 still outperforms HTTP/1.1 and HTTP/2 in this scenario because of the nature of QUIC which reduces the initial connection setup time.

The protocol overhead in HTTP/2 and HTTP/3 becomes the dominant factor in the performance over small RTT links such as the direct link scenarios shown in Figures 7a and 8a. With multi-object pages, the gain obtained from stream multiplexing helps offset the protocol overhead produced in HTTP/2 and HTTP/3 compared to HTTP/1.1. However, with single-object web pages, HTTP/1.1 performs better over small RTT links. We can conclude that the stream multiplexing provides the highest gain when operating over high RTT links such as satellite links.

6.3. The Effect of the Congestion Control Algorithm

The congestion control algorithm used in the transport protocol plays a major role in the performance of HTTP protocols. For this, we evaluated the performance of HTTP with the Cubic and BBR algorithms with multi-object web pages in the satellite link. The web page structure was fixed to the uniform multi-object where the total page size was set according to Table 3.

The average PLT of multi-object web pages downloaded with HTTP over the direct LAN link and the satellite link is shown in Figure 9. The solid lines are for Cubic while the dashed lines represent BBR. The results indicate that BBR outperforms Cubic regardless of the type of HTTP protocol. This is because BBR allows up to $2 \times BDP$ packets in-flight during its start-up phase in addition to BDP packets queued in the buffer. As a result, the number of packets in-flight particularly in satellite link will be large enough, due the large RTT, to produce better channel utilization than Cubic. Although Cubic manages to utilize the channel in the direct LAN link, it fails to do so in the larger RTT satellite link because it only updates its transmission rate every RTT, implying slower growth in such high latency links.

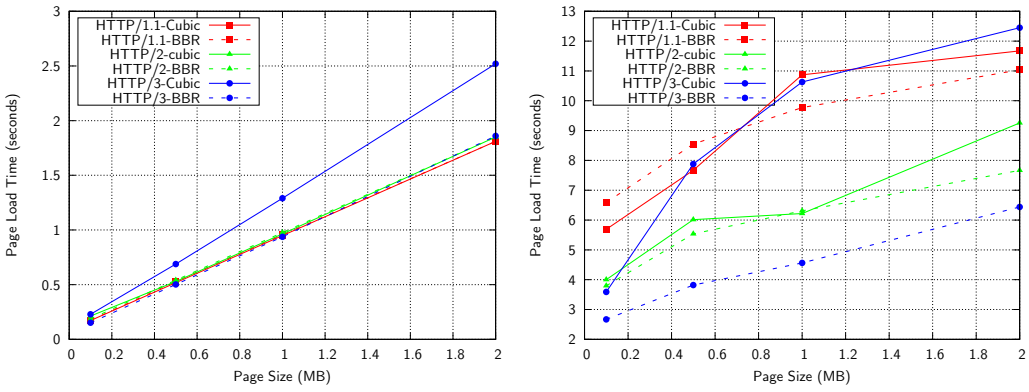

Figure 9. Average PLT of uniform multi-object web page with Cubic vs. BBR. (**a**) Direct LAN link (RTT= 1.08 ms). (**b**) Satellite link (RTT= 530 ms).

Moreover, the evaluation results revealed that Cubic with HTTP/3 also suffers in the direct LAN link. This is because HTTP/3 is able to transfer many more packets compared to HTTP/1.1 and HTTP/2 hence producing more protocol overhead in the link. Table 6 shows the overhead percentage generated from transmitting different web pages in the three HTTP protocols. This is not a problem with BBR since it quickly utilizes the channel as discussed earlier, thus overcoming the effect of the HTTP/3 overhead.

Table 6. The overhead of HTTP protocols at different page sizes.

Total Page Size	100 K	500 K	1 M	2 M
HTTP/1.1	11.76%	6.6%	5.92%	5.57%
HTTP/2	11.27%	6.71%	6.07%	5.71%
HTTP/3	18.23%	9.76%	8.42%	7.79%

6.4. The Effect of the Web Page Structure

The perceived user-level performance of downloading multi-object web pages is affected by the number of embedded objects, their sizes and how they are arranged in the web page. We assess this effect using the First Contentful Paint (FCP) and the Largest Contentful Paint (LCP), defined in Table 5, in addition to the average PLT. The two web page structures considered here are the uniform multi-object page, with equal objects' sizes, and the triangular multi-object page, where the objects are arranged in increasing-then-decreasing order as described in the previous section. The test used HTTP/2 and HTTP/3 over TCP, and QUIC while fixing the congestion control algorithm as BBR because it can fully utilize the channel.

Figure 10 shows the average First Contentful Paint (FCP) of downloading multi-object web pages over HTTP/2 and HTTP/3 in the satellite link. Here, we observe that HTTP/2 needs more time to get the first object from the server compared to HTTP/3. The reason is that HTTP/2 uses TCP which requires a three-way handshake for connection setup and another handshake for TLS security. HTTP/3, on the other hand, uses QUIC which requires only one RTT (1-RTT handshake) for the combined connection setup with TLS security. In addition, all HTTP protocols need at least two RTTs to complete a page request: one RTT to request the main web page (index page) and another RTT to request the first object within the page after the browser parses the main page as illustrated in Figure 11. The triangular multi-object web page structure yields smaller FCP than the uniform in both HTTP/2 and HTTP/3. This is because the first object in the triangular structure is the smallest among the eight objects embedded in the web page. So, it will finish earlier than the first object in the uniform web page structure.

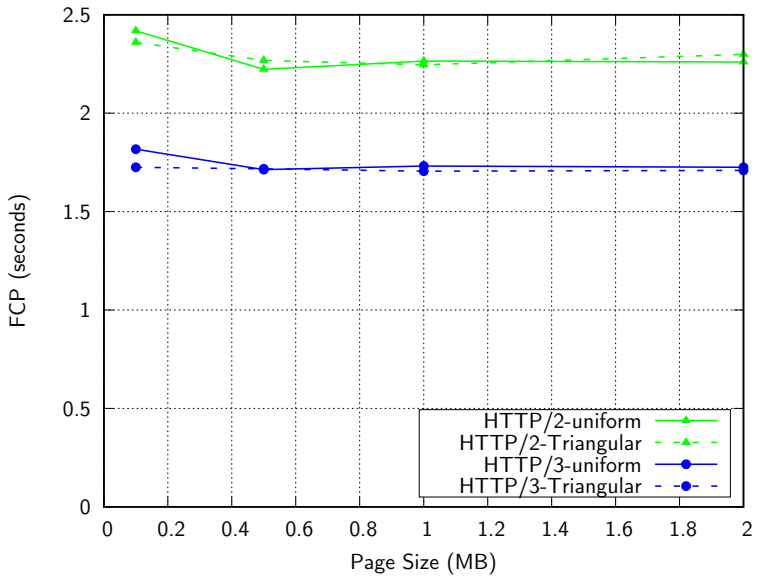

Figure 10. First Contentful Paint (FCP) of multi-object web page in satellite link.

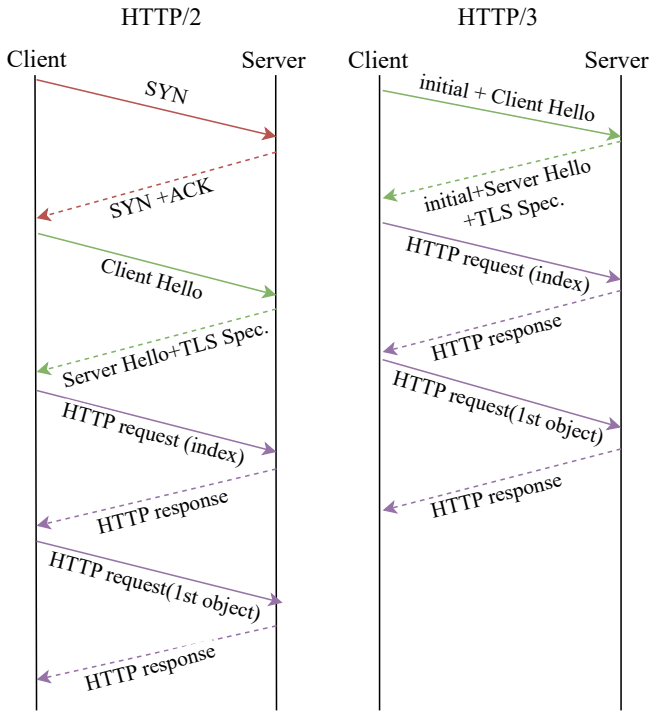

Figure 11. Connection setup, security and request/response timeline in HTTP/2 and HTTP/3.

Next, we look at the second user-experience metric which is the time from the web request until the most significant (largest) object within the web page is loaded successfully. The average Largest Contentful Paint (LCP) of multi-object web pages in the satellite link is shown in Figure 12. Here, HTTP/3 also outperforms HTTP/2 for the same reasons described earlier. However, the results clearly show the effect of the object size distribution and order within the web page itself. The triangular web page has smaller LCP than the uniform web page in both HTTP/2 and HTTP/3. Because the largest object in the triangular structure is located in the middle of the web page, it loads faster when parallel streams are used as in HTTP/2 and HTTP/3. This has a clear improvement on the perceived visual rendering of the web page as seen by the user.

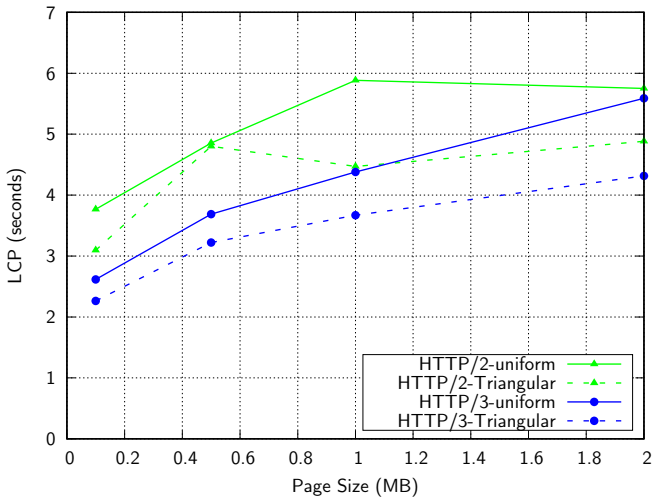

Figure 12. Largest Contentful Paint (LCP) of multi-object web page in a satellite link.

Finally, the average page load time (PLT) of downloading uniform and triangular web pages is shown in Figure 13. Clearly, the object distribution within the web page does not affect the PLT because it is determined by the total time needed to fetch all objects in the page successfully regardless of the size or location of each object.

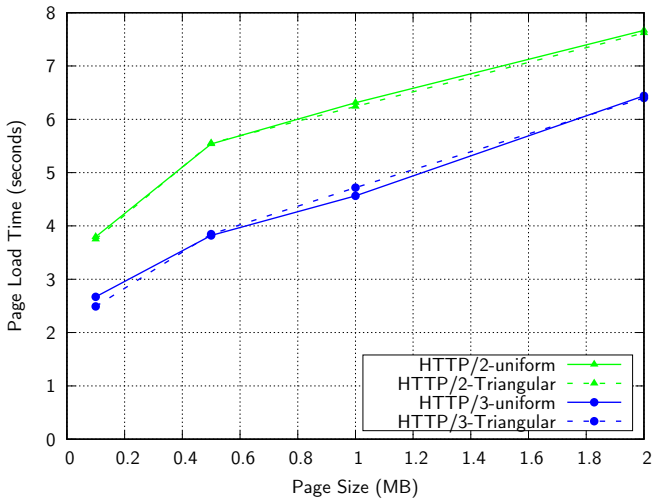

Figure 13. Average PLT of multi-object web page in a satellite link.

6.5. Summary of Main Findings

In general, there is a significant advantage of HTTP/3 compared to either HTTP/2 or HTTP/1.1 in many network environments especially in large RTT satellite links. The results of our investigation can be summarized in the following.

- Most of the performance gain of HTTP/2 and HTTP/3 compared to HTTP/1.1 comes from stream multiplexing of complex web pages. However, this comes at the cost of much larger protocol overhead due to mandatory TLS connection handshake and encryption. We can conclude that the stream multiplexing provides the highest gain when downloading multi-object pages over large RTT links such as satellite links.
- The protocol overhead can dominate the performance of both HTTP/2 and HTTP/3 especially in low RTT links. HTTP/1.1 performs similar to HTTP/2 and HTTP/3 over very low RTT even when it fetches the web page sequentially, because it does not suffer from the overhead of connection handshake and encryption as in HTTP/2 and HTTP/3.
- HTTP/3 further benefits from reducing the initial TLS connection handshake and the non-blocking feature of QUIC. The gains here are large enough to offset the protocol overhead when operating over large RTT links.
- The effect of stream multiplexing is clearly visible when using complex web pages of multiple objects with varying sizes. From our investigation, we found that the perceived visual rendering of web pages will be better in HTTP/2 and HTTP/3 depending on the position of the most significant object within the HTML web page. This is especially notable in HTTP/3 since it does not suffer from HoL blocking at the transport layer.
- BBR congestion control outperforms Cubic regardless of the type of HTTP protocol because it is able to quickly fill the pipe achieving higher utilization of the satellite link. Cubic fails to utilize the satellite link because it only updates its transmission rate every RTT, implying slower growth in these large RTT links.

7. Conclusions

The performance of modern HTTP variants was evaluated over satellite links using a realistic DVB-S2/RCS emulation testbed. We utilized the full-stack software implementation of HTTP/2 and HTTP/3 to study their performance with BBR and Cubic congestion

control algorithms. We considered both simple and complex web pages to uncover the interaction of HTTP with the underlying transport mechanism. For this, we studied multiple user-level performance metrics such as page load time and other page rendering measures such as the FCP and LCP which quantify the perceived visual rendering time of page elements. The evaluation revealed the issue of protocol overhead affecting HTTP/2 and HTTP/3 and how it interacts with the congestion control algorithm in large RTT links. In particular for HTTP/3, the problem becomes more significant when using a loss-based congestion control algorithm such as CUBIC which is widely used on the Internet. The results also suggest that HTTP/3 over QUIC will perform better than HTTP/2 and HTTP/1.1 in satellite links specifically with a more aggressive congestion algorithm such as BBR. Finally, the study revealed a negative interaction between the Cubic slow-update cycle and the protocol overhead in HTTP/3 in both low-latency and high-latency links. This issue is worth further investigation since Cubic is currently the most widely used congestion algorithm on the Internet today.

Author Contributions: Conceptualization, A.A. and A.M.; methodology, A.A.; software, A.A.; validation, A.A. and A.M.; formal analysis, A.A.; investigation, A.A.; resources, A.A.; writing—original draft preparation, A.A.; writing—review and editing, A.M.; visualization, A.A.; supervision, A.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

PLT	Page Load Time
LCP	Largest Contentful Paint
FCP	First Contentful Paint
QUIC	Quick UDP Internet Connections
DVB	Digital Video Broadcasting
BER	Bit Error Rate

References

1. Ippolito, L.J. *Link System Performance*; John Wiley & Sons, Ltd.: Hoboken, NJ, USA, 2017; pp. 75–86.
2. Abdelsalam, A.; Roseti, C.; Zampognaro, F. Steady-state performance evaluation of Linux TCPs versus TCP wave over leaky satellite links. *China Commun.* **2017**, *14*, 17–30. [CrossRef]
3. Secchi, R.; Mohideen, A.C.; Fairhurst, G. Performance analysis of next generation web access via satellite. *Int. J. Satell. Commun. Netw.* **2018**, *36*, 29–43. [CrossRef]
4. Quadrini, M. Performance evaluation of a QUIC-based proxy architecture over a hybrid satellite-terrestrial backhaul network. In Proceedings of the 2019 International Symposium on Advanced Electrical and Communication Technologies, ISAECT 2019, Rome, Italy, 27–29 November 2019.
5. Luglio, M.; Profile, S.; Bujari, A.; Bujari, A.; Palazzi, C.E.; Quadrini, M.; Roseti, C.; Zampognaro, F. A Virtual PEP for Web Optimization over a Satellite-Terrestrial Backhaul. *IEEE Commun. Mag.* **2020**, *58*, 42–48.
6. Border, J.; Shah, B.; Su, C.J.; Torres, R. Evaluating QUIC's Performance Against Performance Enhancing Proxy over Satellite Link | IEEE Conference Publication | IEEE Xplore. In Proceedings of the 2020 IFIP Networking Conference (Networking), Virtual, 22–25 June 2020.
7. Deutschmann, J.; Hielscher, K.S.; Mogildea, C.; German, R. QUIC over Satellite: Introduction and Performance Measurements. In Proceedings of the 25th Ka and Broadband Communications Conference, Sorrento, Italy, 30 September–2 October 2019.
8. Wang, Y.; Zhao, K.; Li, W.; Fraire, J.; Sun, Z.; Fang, Y. Performance Evaluation of QUIC with BBR in Satellite Internet. In Proceedings of the 2018 6th IEEE International Conference on Wireless for Space an Extreme Environments, WiSEE 2018, Huntsville, AL, USA, 11–13 December 2018; pp. 195–199.
9. Custura, A.; Jones, T.; Fairhurst, G. Impact of Acknowledgements using IETF QUIC on Satellite Performance. In Proceedings of the 2020 10th Advanced Satellite Multimedia Systems Conference and the 16th Signal Processing for Space Communications Workshop, ASMS/SPSC 2020, Graz, Austria, 20–21 October 2020.

10. Abdelsalam, A.; Luglio, M.; Roseti, C.; Zampognaro, F. Evaluation of TCP wave performance applied to real HTTP traffic. In Proceedings of the 2017 International Symposium on Networks, Computers and Communications, ISNCC 2017, Marrakech, Morocco, 16–18 May 2017.
11. Yang, S.; Li, H.; Wu, Q. Performance Analysis of QUIC Protocol in Integrated Satellites and Terrestrial Networks. In Proceedings of the 2018 14th International Wireless Communications and Mobile Computing Conference, IWCMC 2018, Limassol, Cyprus, 25–29 June 2018; pp. 1425–1430.
12. Deutschmann, J.; Hielscher, K.S.; German, R. Satellite internet performance measurements. In Proceedings of the 2019 International Conference on Networked Systems, NetSys 2019, Garching b, München, Germany, 18–21 March 2019.
13. Kuhn, N.; Michel, F.; Thomas, L.; Dubois, E.; Lochin, E. QUIC: Opportunities and threats in SATCOM. In Proceedings of the 2020 10th Advanced Satellite Multimedia Systems Conference and the 16th Signal Processing for Space Communications Workshop, ASMS/SPSC 2020, Graz, Austria, 20–21 October 2020.
14. Thomas, L.; Dubois, E.; Kuhn, N.; Lochin, E. Google QUIC performance over a public SATCOM access. *Int. J. Satell. Commun. Netw.* **2019**, *37*, 601–611. [CrossRef]
15. Koyama, Y.; Toyoshima, M.; Takayama, Y.; Takenaka, H.; Shiratama, K.; Mase, I.; Kawamoto, O. SOTA: Small Optical Transponder for micro-satellite. In Proceedings of the 2011 International Conference on Space Optical Systems and Applications (ICSOS), Santa Monica, CA, 11–13 May 2011; pp. 97–101. [CrossRef]
16. Kolev, D.; Takenaka, H.; Munemasa, Y.; Akioka, M.; Iwakiri, N.; Koyama, Y.; Kunimori, H.; Toyoshima, M.; Artaud, G.; Issler, J.L.; et al. Overview of international experiment campaign with small optical transponder (SOTA). In Proceedings of the 2015 IEEE International Conference on Space Optical Systems and Applications (ICSOS), New Orleans, LA, USA, 26–28 October 2015; pp. 1–6. [CrossRef]
17. Fuchs, C.; Perlot, N.; Riedi, J.; Perdigues, J. Performance Estimation of Optical LEO Downlinks. *IEEE J. Sel. Areas Commun.* **2018**, *36*, 1074–1085. [CrossRef]
18. Iyengar, J.; Thomson, M. QUIC: A UDP-Based Multiplexed and Secure Transport. In *RFC 9000*; Internet Engineering Task Force (IETF): Fremont, CA, USA, 2021. [CrossRef]
19. Cui, Y.; Li, T.; Liu, C.; Wang, X.; Kuhlewind, M. Innovating Transport with QUIC: Design Approaches and Research Challenges. *IEEE Internet Comput.* **2017**, *21*, 72–76. [CrossRef]
20. Contributors, M. Connection management in HTTP/1.x. 2019. Available online: https://developer.mozilla.org/en-US/docs/Web/HTTP/Connection_management_in_HTTP_1.x (accessed on 1 January 2022).
21. Grigorik, I. *High Performance Browser Networking*; O'Reilly Media: Sebastopol, CA, USA, 2013.
22. Bishop, M. HTTP/3. RFC 9114. 2022. Available online: https://doi.org/10.17487/RFC9114 (accessed on 1 January 2022).
23. Ha, S.; Rhee, I. CUBIC : A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Oper. Syst. Rev.* **2008**, *42*, 64–74. [CrossRef]
24. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR: Congestion-Based Congestion Control. *Commun. ACM* **2017**, *60*, 58–66. [CrossRef]
25. Rhee, I.; Xu, L.; Ha, S.; Zimmermann, A.; Eggert, L.; Scheffenegger, R. CUBIC for Fast Long-Distance Networks. RFC 8312. 2018. Available online: https://doi.org/10.17487/RFC8312 (accessed on 1 January 2022).
26. Cardwell, N.; Cheng, Y.; Yeganeh, S.H.; Swett, I.; Jacobson, V. BBR Congestion Control. 2022, *work in progress*.
27. Dubois, E.; Kuhn, N.; Dupe', J.B.; Gelard, P.; Arnal, F.; Baudoin, C.; Delrieu, A.; Pradas, D. OpenSAND : An Open Source SATCOM Emulator. In Proceedings of the 23rd Ka and Broadband Communication Conference, Trieste, Italy, 23 October 2017; p. 11.
28. LiteSpeed Technologies Inc. OpenLiteSpeed. 2021. Available online: https://openlitespeed.org/ (accessed on 1 January 2022) .
29. Firefox Nightly 69.0a1, See All New Features, Updates and Fixes. 2021. Available online: https://www.mozilla.org/en-US/firefox/69.0a1/releasenotes/ (accessed on 1 January 2022).