How the Web Was Won

Keeping the Computer Networking Curriculum Current with HTTP/2

Colin Allison
School of Computer Science
University of St Andrews
St Andrews, UK
ca@st-andrews.ac.uk

Hussein Bakri
School of Computer Science
University of St Andrews
St Andrews, UK
hb@st-andrews.ac.uk

***Abstract*— The Internet and the Web continue to grow in their pervasiveness and as new functionality and behavior emerge it is a challenge to keep the computer networking curriculum up to date. There are many excellent networking textbooks available but they cannot always keep pace with the rate of change. Recent developments in HTTP are a good example of this situation. Since around 2012 many of the web transactions between popular browsers and major web sites have been using a protocol called SPDY, which operates significantly differently from HTTP version 1.1 – the version covered in networking textbooks. SPDY has been largely adopted into the final standard of HTTP version 2. This paper seeks to fill the gap between current textbooks and the versions of HTTP now in use. It gives an overview of HTTP evolution from a technical perspective before suggesting materials and approaches that can be used as learning resources for the topic and how conceptual understanding can be reinforced through hands-on activities which use browsers' native network monitoring capabilities and other readily available tools.**

Keywords—Computer Networking Education, HTTP Evolution

I. INTRODUCTION

As a consequence of the Internet boom around the year 2000 computer networking education has been well served in terms of textbooks. A challenge for authors of the more comprehensive networking textbooks however is keeping them up to date.

Table I: Frequency of networking textbook publication update

	Edition and Year of Publication						
	1	2	3	4	5	6	*average update interval*
T&W	1980	1988	1995	2002	2010		7 years
P &D	1996	1999	2003	2007	2011		4 years
K &R	2000	2003	2005	2007	2009	2012	2 years

If we take three major texts: Tanenbaum and Wetherall[1] (T&W) [1], Peterson and Davie (P&D) [2], Kurose and Ross (K&R) [3]; we can get a feel for the rate of change by looking at the frequency of new editions (see Table I).

[1] Andrew Tanenbaum was the sole author of editions 1 – 4; he was joined by David Wetherall as co-author for the 5th edition.

The more recently authored books show a greater frequency of revision and as revising a major textbook is a time consuming process it can be assumed that these updates are considered too important to delay for longer periods. Yet, at the same time, we do not have to look far to find an example of a widely used Internet protocol that is not covered by even the most recent textbooks – SPDY – the basis for HTTP/2.

Why is the HTTP family of protocols (HTTP 1.0, HTTP 1.1, HTTPS, SPDY, HTTP/2) an important part of the networking curriculum? Firstly, these are the application level protocols that carry the largest proportion of Internet traffic including social media, e-commerce, and streaming video. As such it is incumbent on networking education to explain the principles, operation, benefits and drawbacks of such a widely used set of protocols.

Secondly, in contrast to the traditional bottom-up networking pedagogy whereby the physical layer is covered first, then the link layer, and so on, a top-down approach starting with the application level has been introduced by books such as K&R, and widely adopted. Peterson and Davie's book is structured as bottom-up but for the 5th edition they issued an alternative pathway document on how to use their content in a top-down manner. HTTP is naturally one of the most relevant application level protocols to use in a top-down approach. Students can quickly feel a sense of achievement in designing and deploying their own web server which in turn promotes engagement with other aspects of the discipline.

Finally, through the critical study of this family of protocols students can gain insight into Internet protocol design, evolution and standardization. For example, there is educational value in covering HTTP/2 as it shows that key features of HTTP 1.1 such as pipelining, described as performance enhancement in textbooks, never actually worked in practice and were not adopted or deployed. While it is testimony to the value of layered model abstraction that few web users are aware when they are obtaining content via SPDY rather than HTTP 1.1 it is not an acceptable situation for students in computer networking classes, especially as they are still being taught about the operation of earlier forms of HTTP in major texts.

This paper proceeds by reviewing the HTTP story so far then makes suggestions for readily available resources which can

support the inclusion of HTTP/2 in an evolutionary context, within the networking curriculum.

II. THE HTTP STORY SO FAR….

The original Web was based around the hypertext transfer protocol (HTTP) and the hypertext mark-up language (HTML). There were numerous precursors in the form of distributed hypertext systems, but in true Internet tradition the simplicity and openness of the original HTTP and HTML standards allowed them to be readily implemented in forms that could be made to interoperate across the network. HTTP 0.9 was published in 1991 [4]; it was a subset of what was called "Basic HTTP" in 1992 [5] – much of which became known as HTTP 1.0 [6]. In 1993 a major boost came in the form of the Mosaic web browser [7] which was easy to use and brought multimedia web pages to life. It was not uncommon to hear the term "the Mosaic protocol" being incorrectly used to refer to the web at that time. As the use of the web snowballed, HTTP 1.0 (fully specified in 1996), attracted attention from network researchers and they discovered considerable space for improvement [2-5].

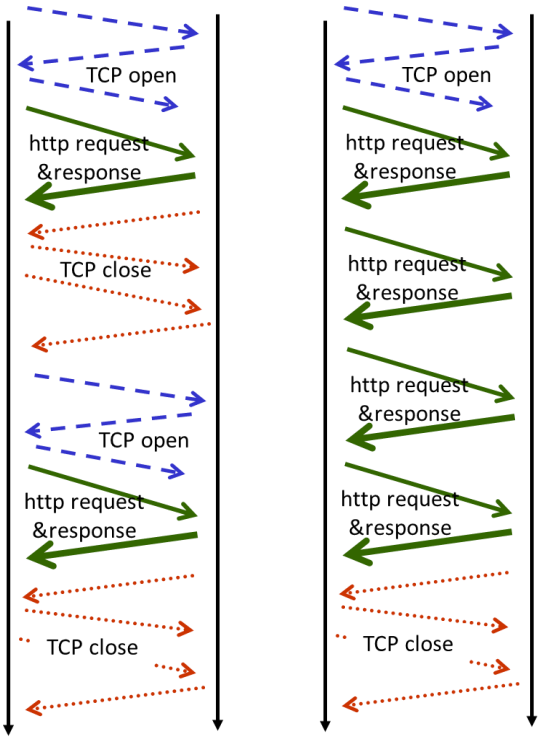

Figure 1. (a) http 1.0 (b) http 1.1

The most significant problem identified was the interplay between TCP and HTTP. Most interactions between a web client and a web server at that time were between a client and the same server. However, each HTTP response and request required its own TCP connection. As TCP uses three segments to set up a connection and up to four segments to close it down, the transport protocol overhead typically used more network resource in terms of Round Trip Time and bandwidth than the application level protocol. Figure 1a shows HTTP 1.0 obtaining two web objects. This profligate use of TCP was seen as wasteful, and of course, congestion avoidance was also a big issue. In addition HTTP 1.0 is a "stop and wait" protocol so if a web page consisting of some text and a few images was to be built and rendered then multiple TCP connections were needed and enterprising browser designers decided to open these in parallel, reducing the overall Page Load Time (PLT), delivering a better user experience, but effectively subverting the aims of TCP's congestion management mechanisms in the sense that one application was getting more than its "fair share" of available network resource. However, depending on the context consisting of the actual clients and servers, their platforms, the web page(s) being requested, and the network path, significant parts of the overall delay in PLT could often be traced to the browser, the server or the TCP protocol rather than the network throughput or HTTP protocol [8] [9].

A. HTTP 1.1

HTTP 1.1 [10] sought to improve over the previous version in the areas of: Caching, Bandwidth optimization, Connection management, Message transmission, Internet address conservation, Error notification, Security, Integrity & Authentication, and Content negotiation [11].

Briefly; IP address conservation was improved through the use of virtual hostnames for servers, specified in the new header "host" field; caching was better supported by the introduction of unique ETags for objects; in practice HTTPS (HTTP over SSL or TLS) was adopted rather than the proposed HTTP 1.1 mechanisms for security; message transmission encoding could be treated distinctly from content encoding.

The concern over the inefficient use of TCP was addressed by improved connection management in the form of persistent connections. This is supported by the "Keep-Alive" header field and is in widespread use. This means that a single TCP connection between a client and a web server can be kept open to support multiple HTTP request/response interactions (see Fig. 1b).

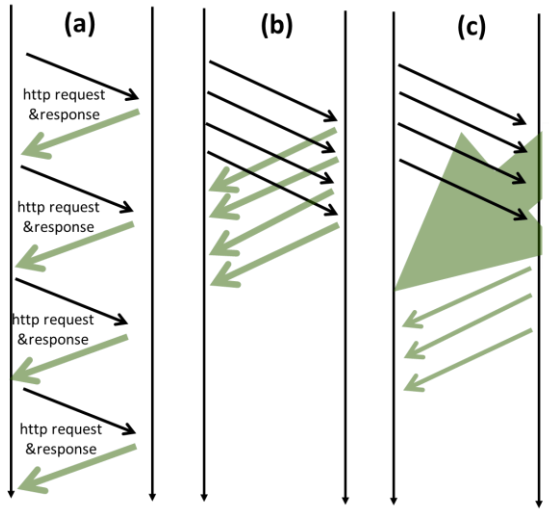

Figure 2. a) Stop & Wait; b) Pipelining; c)Head of Line Blocking.

It was hoped that the introduction of persistent TCP connections would reduce the number of parallel HTTP/TCP connections opened by a browser. In practice this did not happen.

Bandwidth optimization was addressed mainly by the introduction of pipelining, whereby a client did not need to wait for a response before sending a further request (see Fig 2a, b). In practice, pipelining was not adopted. It was partly thwarted by intermediary boxes such as proxies, but also by the "head of line blocking" situation (see Fig 2c), whereby servicing a single long-running request could hold up all subsequent ones, even though they could be answered relatively efficiently. In short, bandwidth optimization did not succeed.

Two points should be noted about HTTP 1.1. Firstly, it was designed to be backwardly compatible with HTTP 1.0, so older clients could still interact with newer servers and newer clients work with older servers. This was achieved by simply making none of the new features mandatory. Hence pipelining could be allowed to fail through lack of popular adoption. However, another new feature, the "Upgrade" header, was intended to allow for both client and server to switch entirely to an alternate protocol for content transfer. This added some future proofing to HTTP 1.1 and is now used for switching to HTTPS or SPDY.

B. HTTPS

As the global, public Internet became increasingly used for commercial and mission critical purposes it became necessary to provide security.

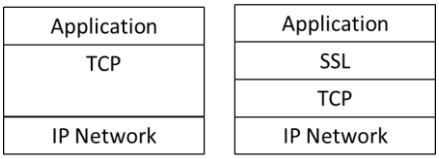

Figure 3: The Secure Sockets Layer

The Secure Sockets Layer (SSL) or Transport Layer Security (TLS) does this for TCP at the transport level. SSL provides confidentiality, integrity & end-point authentication. Any networked application written using the TCP socket programming abstraction can readily improve its security by using SSL. This has led to many common networking applications being deprecated or firewalled and replaced by their SSL-based secure versions. The remote shell command **`rsh`** became **`ssh`**, **`cp`** became **`scp`**, **`ftp`** became **`sftp`** and so on. While some HTTP sites moved to HTTPS, the relative proportion of HTTPS/HTTP traffic remains small, even though there was a post-Snowden surge in May 2014:

> *"before the Snowden revelations encrypted traffic accounted for 2.29 percent of all peak hour traffic in North America, according to Sandvine's report. Now, it spans 3.8 percent. But that's a small jump compared to other parts of the world. In Europe, encrypted traffic went from 1.47 percent to 6.10 percent, and in Latin America, it increased from 1.8 percent to 10.37 percent."* [12]

While HTTPS provides a relatively high degree of security for web traffic compared with HTTP the flawed operation of the Public Key Infrastructure commercial market has partially undermined its reliability [13].

C. HTTP/2 and SPDY

HTTP/2 [14] [15] is a major enhancement to HTTP 1.1, principally motivated by the need to improve the Page Load Time (PLT) of modern, large, complex web pages. Average page sizes and their complexity in terms of the number of objects have grown from approximately 10 Kbytes in 1995 to 1600 Kbytes in 2014, and from two objects in 1995 to over one hundred objects in 2014 [16]. Not only has the number of objects grown, but they are more varied in type and come from an increasing number of different domains.

> *"Today's Web bears little resemblance to the Web of a decade ago. A Web page today encapsulates tens to hundreds of resources pulled from multiple domains. Users access the Web from diverse device form factors, while browsers have improved dramatically…..A constant throughout this evolution is the underlying application layer protocol—HTTP— designed at a time of far less page complexity.….HTTP (1.1) is not optimal, with pages taking longer to load. Studies over the past five years suggest even 100 milliseconds additional delay can have a quantifiably negative effect on Web use, spurring interest in improving Web performance"* [17].

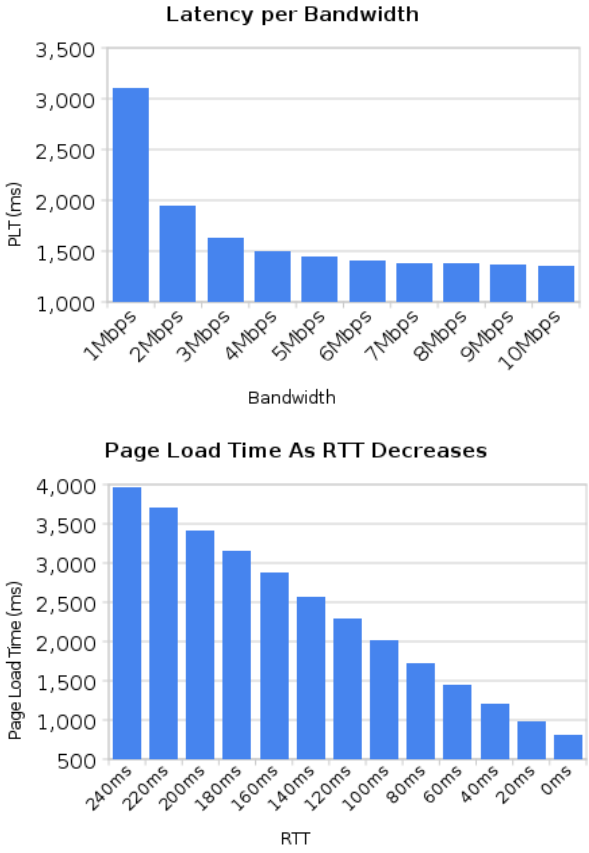

Figure 4: Impacts of Bandwidth vs RTT on Page Load Time (from [18]).

While the network and protocol components are only part of the overall delay in Page Load Time, the latency engendered by HTTP 1.1 was seen a worthwhile target, and hence Google launched the SPDY R&D project in 2009 to provide an alternative protocol.

> "*SPDY adds a framing layer for multiplexing multiple, concurrent streams across a single TCP connection (or any reliable transport stream). The framing layer is optimized for HTTP-like request-response streams, such that applications which run over HTTP today can work over SPDY with little or no change on behalf of the web application writer…..SPDY attempts to preserve the existing semantics of HTTP. All features such as cookies, ETags, Vary headers, Content-Encoding negotiations, etc work as they do with HTTP; SPDY only replaces the way the data is written to the network.*" [19]

Internet access bandwidths have increased while pages have grown but the basis for much of SPDY's design was the belief that the main gains could be achieved by reducing the aggregate Round Trip Time (RTT) in a session. The comparison provided by Belshe [18] (see Fig. 4) pointed towards the relative importance of reducing RTT as opposed to increasing bandwidth beyond e.g. 3 Mb/s. In 2014 Akamai reported the global average connection bandwidth as 4.5 Mb/s [20].

HTTP/2, which started as a copy of SPDY in 2012, was almost fully accepted as an Internet standard by early 2015[14].[2] A very significant point is that unlike the change from HTTP 1.0 to HTTP 1.1, a SPDY implementation must support *all* the SPDY protocol features. This is achieved by using the "UPGRADE" header in HTTP i.e. if the client and server agree to switch to SPDY then all the new features must be supported. All SPDY traffic is encapsulated by SSL, and uses port 443. HTTP/2 has left open the possibility of non-SSL based sessions, but by March 2015 this option does not appear to have been implemented, and it is not clear that it will be. HTTP/2 also seeks to reduce the number of concurrent TCP connections from a browser to the same domain.

By 2014 most of the global web-based service providers including Google, Twitter and Facebook supported SPDY at the server side, and most of the popular browsers, Chrome, Firefox, Safari and IE, supported SPDY at the client side, so in effect, SPDY had already conquered a significant part of the web before being repackaged as the HTTP/2 draft standard.

The following subsections outline the key features introduced by SPDY and mostly adopted in HTTP/2 [21].

1) Frames, Streams and Multiplexing

The unit of communication in HTTP/2 is the *frame*. There are ten different frame types: DATA, HEADERS, PRIORITY, RST_STEAM, SETTINGS, PUSH_PROMISE, PING, GOAWAY, WINDOW_UPDATE, CONTINUATION.

A *stream* in HTTP/2 consists of bidirectional sequences of frames flowing between two endpoints (client and server). The server and client can send data simultaneously. Multiplexing allows for multiple streams of request and response frames (of maybe similar or different data) on a single TCP connection.

2) Prioritization, Dependency of Streams

Streams can be interleaved and prioritized. This allows an endpoint to allocate more resources to what is being prioritized when managing concurrent streams. Priority information can be used to select the appropriate streams for transmitting frames when there is limited sending capacity for any reason. A client can assign a priority number for a new stream in the HEADERS frame. Reprioritization of reserved streams can be regulated by the PRIORITY frames. This allows for more effective pipelining than HTTP 1.1 in that Head of Line blocking (see Fig 2c) can be avoided.

Streams can explicitly depend on the completion of other streams. This also affects the priority of streams. Dependency is assigned a weight between 1 and 256 inclusive. Dependent streams share the resources assigned to their parent in accordance with the weight assigned to them. Dependent streams move with their parent stream whenever the parent is reprioritized. A stream that is not dependent on any other stream is given a weight of 0 [14].

3) Binary Framing Layer

The binary framing layer in SPDY "dictates how the HTTP messages are encapsulated and transferred between the client and server"[22]. HTTP/2 has kept the same semantics, such as verbs and headers of HTTP 1.x. Changes occur in how these semantics are encoded, encapsulated and then transferred. In other words, their encoding in transit is what is different.

4) Server Push

A server can send pre-emptively (or "push") additional objects in addition to replying to requests from clients. For example, a server can send images, icons, CSS or JavaScript code before the client explicitly requests them. A client can however request that *server push* be disabled during a connection. Khalid et al. [23] have argued that this feature can be problematic in mobile devices because it can waste battery or bandwidth and proposes mechanisms for HTTP/2 that adjust the overall performance on mobile devices.

5) Header Compression

In HTTP 1.x, headers are typically repetitive and verbose. HTTP/2 compresses headers using the HPACK algorithm [24], based on Huffman encoding.

6) Flow Control

HTTP/2 Flow Control is used for both individual streams and for the connection as a whole. It is regulated through the use of the WINDOW_UPDATE frame; only DATA frames are subject to its effect. Receivers advertise how many octets they can receive for a specific stream or for the whole connection. The sender must respect the limits advertised by the receiver.

[2] SPDY and HTTP/2 are used interchangeably in many papers due to the great influence of SPDY on HTTP/2.

Flow control in HTTP/2 aims to make it possible to utilize network resources better by not allowing a particular stream to starve, and by dealing with slow/fast upstream and downstream connections adequately.

7) RTT and Liveness

PING frames have the highest priority. They are used to measure round trip time and check if the connection is still functional or the peer is still alive.

III. SUPPORTING HTTP IN THE NETWORKING CURRICULUM

This section suggests resources that can be used educationally to complement the accounts of HTTP in popular texts through contextualization and hands-on exercises.

A. Contextualisation

The story of HTTP evolution from HTTP 0.9 to HTTP/2 is in itself an educational topic, illustrating the standardization process in the W3C and IETF. Popular textbooks use HTTP 1.1 as a reference, some of them including the pipelining feature which has never been widely used in practice. It is recommended that the sections on HTTP in such texts are augmented by information on SPDY and HTTP/2. An accessible, if rather uncritical, overview of SPDY can be found in [17]. A short and readable account of the key differences between HTTP 1.0 and HTTP 1.1 can be found in [11]. Critical commentaries on SPDY and HTTP/2 can be found in [25, 26]. Table II gives a summary overview of key differences between the deployed versions of HTTP between 1995 and 2015.

Internet standards are published as RFCs. The nature of RFC content has been referred to as "…very technical, turgid and nearly incomprehensible" [27]. As a light-hearted poke at RFC 2068 (HTTP 1.1), RFC 2324 uses the same language style to describe the Hyper Text Coffee Pot Control Protocol (HTCPCP) [28], which amongst other features introduces the new error code 418 "I'm a teapot".

These types of textual materials can be used by lecturers as the basis for learning resources, or can be passed directly to students as study topics for essays. Branches can be followed if there is time in the curriculum. For example a particular criticism from [26] is that all HTTP/2 sessions are being run over TLS. Empirical studies have shown that there can be a significant cost of using SSL [29] – so when is a secure connection really (not) needed? Do public library opening hours and bus timetables need to be rendered immune from eavesdroppers?

Another criticism of HTTP/2, possibly best suited for more advanced students, is that it violates the established network design principle of *layering and abstraction* by replicating much of the functionality already provided by TCP at the underlying transport level. For example, both protocols support flow control, window size negotiation and pipelining. A further consideration is that SPDY introduces explicit state to HTTP, by way of session initiation and closedown, in a similar way that a TCP virtual connection is managed in its macro state.

Studies have compared the performance of SPDY to previous HTTP versions [30] [31]. These give mixed, sometimes contradictory, results in terms of SPDY outperforming older versions of HTTP or the opposite. SPDY has been studied on mobile devices[32] and on high latency Satellite networks [33].

Other factors such as Web page characteristics, server load and browser processing also play an important role in the overall perceived page load time of course [25].

Part of the wider context includes the topic of making the web faster. This can include Content Distribution Networks (covered in major textbooks); increasing TCP's opening window size [34], and domain sharding, whereby a browser is forced into making parallel connections due to deliberately placing web page components in different domains [35].

Table II: Summary of major differences in HTTP versions 1.0, 1.1 and 2

HTTP 1.0	HTTP 1.1	HTTP/2
"Stop and Wait", strictly sequential processing of requests and responses over TCP	"Stop and Wait", strictly sequential processing of requests and responses over TCP	Full duplex streams of binary frames over TLS/TCP
PDU: HTTP Message	PDU: HTTP Message	PDU: HTTP/2 Frame (10 Types)
New TCP connection opened for each Request/Response pair Browsers seek performance gain by opening multiple parallel TCP connections, even between client and server in same domain	Persistent TCP connections specified and adopted Pipelining specified but not mandatory and not adopted Browsers continue to open multiple parallel TCP connections within same domain	Aim: One persistent TCP Connection per domain Multiple concurrent streams within the TCP connection Pipelining mandatory Stream Multiplexing and Prioritization Dynamic stream dependencies and reprioritization
Caching, Content compression option	Caching, content compression option	Caching, Content compression
		Header Compression Server Push Flow Control

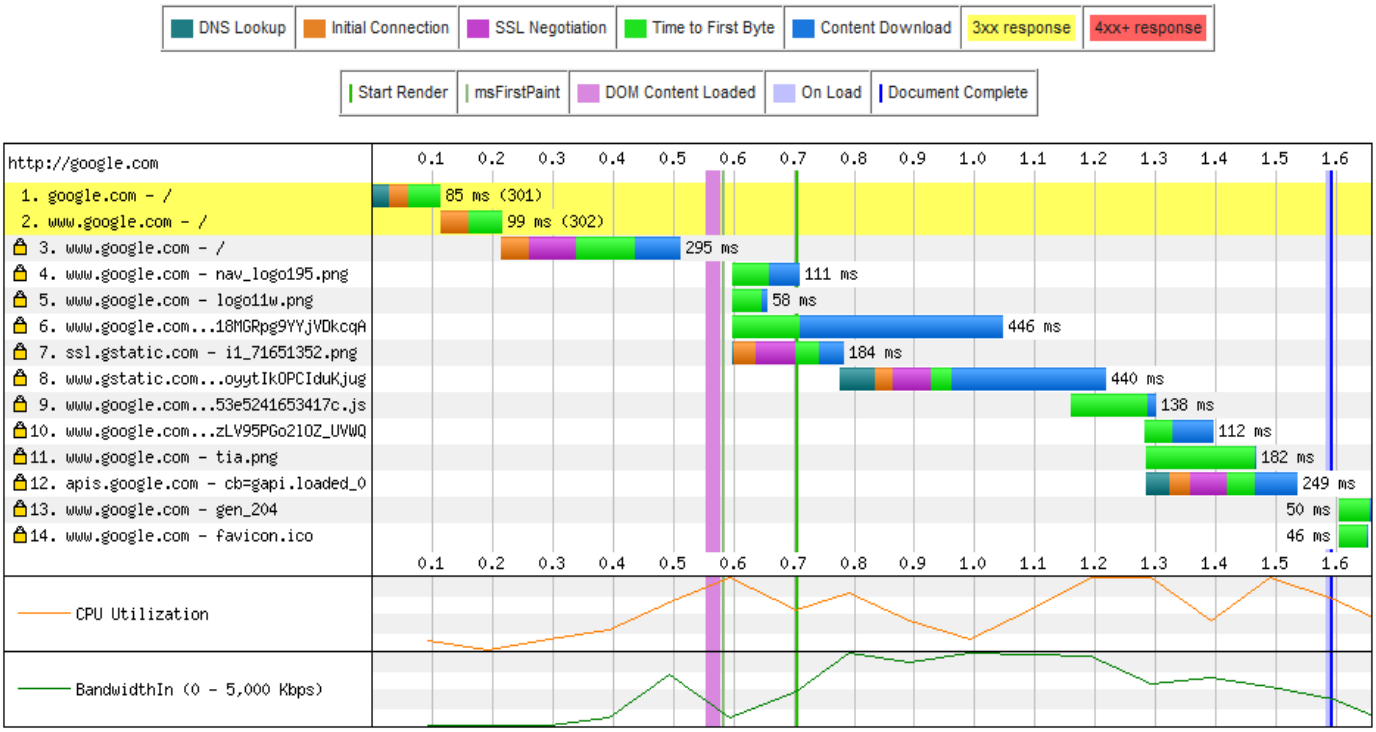

Figure 5: A waterfall view of page load time from `www.webpagetest.org` after accessing google.com

Web page content optimization is supported by systems such as ModPageSpeed [36], an executable Apache module that uses a complex set of rules to dynamically rewrites a page for particular connection.

David Wetherall has prepared a MOOC based on T&W 5th edition [1]; the videos can be accessed on demand, irrespective of the MOOC schedule. Video 8.8 [37] lasts for twenty minutes and addresses the future of HTTP, a topic not covered in the book. Around four minutes is spent on SPDY and HTTP/2 developments. The tentative nature of the discussion suggests the video was made around 2012. It is a useful high level introduction to the modern web.

B. Hands-on activities: observation and analyses

The use of Wireshark [38] in lab exercises has been popularized in supporting material by Kurose and Ross [3]. Recent Wireshark releases support both SPDY and HTTP/2 identification.

The *webpagetest* tool [39] is a free online service that is also useful educationally. Figure 5 shows a "Waterfall View" of the Page Load Time for google.com (from webpagetest's point of view onto the Internet). There are also facilities built-in to Chrome and Firefox that allow students to observe the components of PLT. These can optionally be displayed in a waterfall style (Fig.6). Note that a Firefox add-on [40] signals in the address bar that SPDY or HTTP/2 is in use.

The web page at **`spdycheck.org`** tests user-specified sites for SPDY, TLS, HTTP/2 and HTTP 1.1 support. Networking students can progress from understanding to creating by writing their own code to carry out these tests.

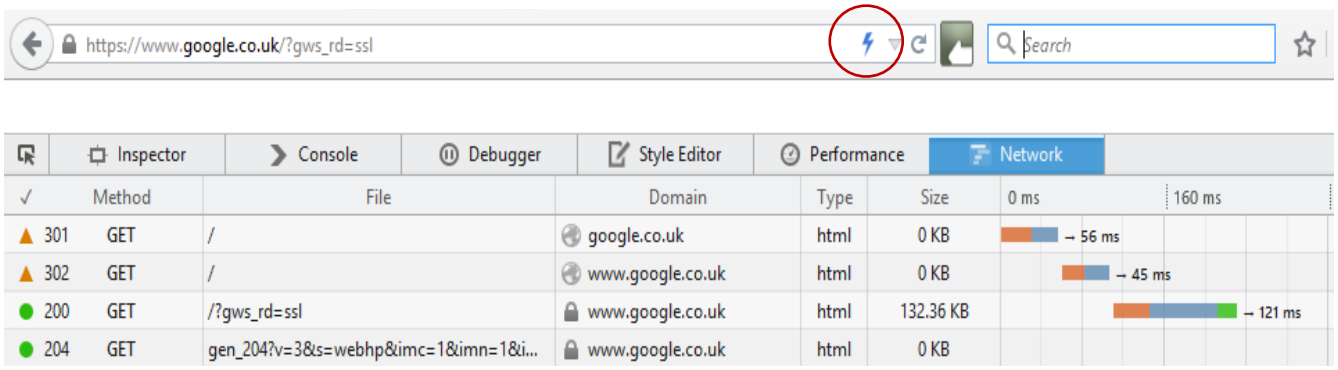

Figure 6: Screenshot of Firefox's built-in network monitoring facility; an add-on [40] shows when SPDY or HTTP/2 is in use (circled)

```
3868: SPDY_SESSION
clients2.google.com:443 (DIRECT)
Start Time: 2015-04-27 12:01:32.782

t=88844 [st=   0] +SPDY_SESSION  [dt=?]
                   --> host = "clients2.google.com:443"
                   --> proxy = "DIRECT"
t=88844 [st=   0]    SPDY_SESSION_INITIALIZED
                     --> protocol = "h2-14"
                     --> source_dependency = 3865 (SOCKET)
t=88844 [st=   0]    SPDY_SESSION_SEND_SETTINGS
                     --> settings = ["[id:3 flags:0 value:1000]","[id:4 flags:0 value:10485760]"]
t=88844 [st=   0]    SPDY_STREAM_UPDATE_RECV_WINDOW
                     --> delta = 10420225
                     --> window_size = 10485760
t=88844 [st=   0]    SPDY_SESSION_SENT_WINDOW_UPDATE_FRAME
                     --> delta = 10420225
                     --> stream_id = 0
t=88845 [st=   1]    SPDY_SESSION_SYN_STREAM
                     --> fin = false
                     --> :authority: clients2.google.com
                         :method: POST
                         :path: /service/update2
                         :scheme: https
                         accept-encoding: gzip, deflate
                         content-length: 1261
                         content-type: application/xml
                         user-agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML,
                     --> spdy_priority = 3
                     --> stream_id = 1
                     --> unidirectional = false
t=88845 [st=   1]    SPDY_SESSION_SEND_DATA
                     --> fin = true
                     --> size = 1261
                     --> stream_id = 1
t=88845 [st=   1]    SPDY_SESSION_UPDATE_SEND_WINDOW
                     --> delta = -1261
                     --> window_size = 64274
```

Figure 7: A trace from Chrome's built in network monitor showing a SPDY session

It is possible to get a breakdown of a SPDY or HTTP/2 conversation in Chrome by initially using the URL: **`chrome://net-internals/#spdy`**. This brings up the following information:

- **`HTTP/2 Enabled: true`**
- **`Use Alternate Protocol: true`**
- **`Force HTTP/2 Always: false`**
- **`Force HTTP/2 Over SSL: true`**
- **`Next Protocols: http/1.1,spdy/3.1,h2-14`**

If a live HTTP/2 session is then selected, the working of the protocol can be observed, including streams, priorities and flow control window size (see Fig 7). It is interesting to note that in some cases e.g. Facebook, SPDY appears to act as an encapsulating layer for HTTP 1.1 whereas in an all-Google HTTP/2 conversation (Fig. 7) there is no explicit mention of HTTP 1.1 although the familiar header fields are listed.

Entering **`about:config`** and then searching for **`spdy`** in the Firefox address bar will elicit the list in Table III.

Table III: Firefox SPDY parameters

`network.http.spdy.allow-push`	**`true`**
`network.http.spdy.chunk-size`	**`16000`**
`network.http.spdy.coalesce-hostnames`	**`true`**
`network.http.spdy.default-concurrent`	**`100`**
`network.http.spdy.enabled`	**`true`**
`network.http.spdy.enabled.deps`	**`true`**
`network.http.spdy.enabled.http2`	**`true`**
`network.http.spdy.enabled.http2draft`	**`true`**
`network.http.spdy.enabled.v3-1`	**`true`**
`network.http.spdy.enforce-tls-profile`	**`true`**
`network.http.spdy.persistent-settings`	**`false`**
`network.http.spdy.ping-threshold`	**`58`**
`network.http.spdy.ping-timeout`	**`8`**
`network.http.spdy.push-allowance`	**`131072`**
`network.http.spdy.send-buffer-size`	**`131072`**
`network.http.spdy.timeout`	**`180`**

Students can be asked to research and explain the meanings of these parameters, and can also change the settings and record the effects when interacting with the same web site.

C. *Hands-on activities: Simulators and Emulators*

For students with adequate time the next stage beyond observation and analyses is to use a simulator to modify traffic characteristics such as bandwidth, packet loss and delay, to see how that impacts on performance. A good starting point is to give the student a pointer towards Belshe's comparison of bandwidth vs RTT [18] with respect to impact on PLT (see Fig. 4) and ask them to see if they can reproduce these figures through simulation and measurement. Science is built on reproducible research results but in the case of Internet measurements, even simulations, reproducibility can be challenging.

There are various open source network simulation tools available, including ns3 [41] and Trickle [42]. Opnet is now called Riverbed Modeler [43] and is free for academic use.

In lab exercises the traffic shaping Linux kernel library (tc) [44] and NetEm [45] can be used to emulate delay and packet loss. Bandwidth control can be achieved using the Hierarchical Token Bucket control feature of the queuing discipline interface (qdisc) [46] in Linux. SPDY or HTTP/2 can be turned on and off in Chrome using the Chrome settings option. Sites including Facebook, YouTube and StatCounter can be used as test cases. In our experience it proved hard for any student to replicate the performance gains expected by moving to SPDY, but we should emphasize that this was an educational exercise rather than a robust piece of research.

IV. QUIC

Interestingly, when observing and analyzing live HTTP/2 connections we discovered the QUIC protocol [47] being deployed by Google.

> *"QUIC is an experimental protocol aimed at reducing web latency over that of TCP. On the surface, QUIC is very similar to TCP+TLS+SPDY implemented on UDP. Because TCP is implement in operating system kernels, and middlebox firmware, making significant changes to TCP is next to impossible. However, since QUIC is built on top of UDP, it suffers from no such limitations."* [47]

QUIC supports HTTP/2 functionality over UDP port 443. During a SPDY session the UPDATE header is used to switch to QUIC; this appears to be the current Google protocol of choice for short exchanges such as visits to sites which record advertising, analytics and marketing information. Entering **`chrome://net-internals/#spdy`** in the Chrome address bar reveals a comprehensive list of alternative QUIC based URLs for Google services.

Why QUIC? Part of the performance problem for SPDY and HTTP/2 lies in the behavior of TCP (see [48] [32]). A single TCP congestion avoidance window can put SPDY or HTTP/2 at a disadvantage compared with multiple HTTP 1.1/TCP connections each with a separate congestion window, which is often the case with HTTP 1.x.

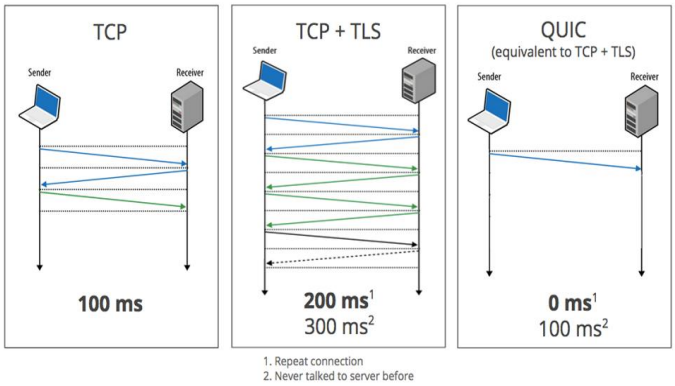

Figure 9: QUIC's Zero Round Trip handshake

A single lost packet will impact on *all* the multiplexed streams in a single TCP connection. QUIC is UDP based so avoids this. In addition, QUIC has a *zero round trip handshake* capability, see Fig. 9, conveniently avoiding the TCP handshaking and close down exchanges (see Fig. 3) that would increase the number of round trips.

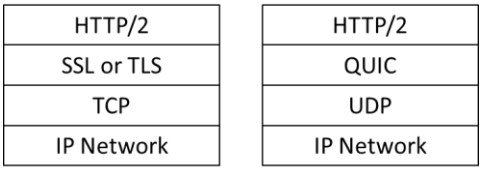

Figure 10: HTTP/2 alternative protocol stacks

However, UDP lacks congestion control and SSL functionality so QUIC seeks to replicate these within itself: QUIC has a pluggable congestion control algorithm option which is currently TCP Cubic and supports its own TLS-like security protocol, thus seeking to recreate the semantics of HTTP/2 over TLS/TCP without the performance drawback. Figure 10 summarizes using networking layered models. Recent versions of Wireshark can identify QUIC.

V. CONCLUSION

Areas of the Internet are undergoing a rapid rate of change. A pertinent example is the HTTP 1.1 application-level protocol which has been superseded by SPDY in many of the web transactions between popular browsers and major web sites since 2012. While it is testimony to the value of protocol layering that web users are largely unaware of this major change in HTTP it is not acceptable that computer networking students remain ignorant of it. It is incumbent on educators to ensure that the curriculum reflects such significant changes in this pervasive web protocol. Most of SPDY has now been adopted as the HTTP/2 standard but even the most recent editions of established computer textbooks have not caught up with HTTP/2. This paper makes a modest contribution towards filling the current gap by giving recommendations for resources that can be used to contextualize and obtain hands-on experience of recent developments in HTTP evolution.

ACKNOWLEDGMENTS

Thanks to Faizan Agha for digging deeper.

REFERENCES

[1] A. S. Tanenbaum and D. Wetherall, *Computer Networks*, 5th ed.: Pearson, 2010.

[2] L. Peterson and B. Davie, *Computer Networks: A Systems Approach*, 5th ed.: Morgan Kaufmann, 2011.

[3] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach*, 6th ed.: Pearson, 2012.

[4] T. B. Lee, "HTTP 0.9; http://www.w3.org/Protocols/HTTP/AsImplemented.html," W3C, 1991.

[5] T. B. Lee, "Basic HTTP; http://www.w3.org/Protocols/HTTP/HTTP2.html," 1992.

[6] T. Berners-Lee, R. Fielding, and H. Frystyk, "RFC 1945 Hypertext Transfer Protocol -- HTTP/1.0," IETF 1996.

[7] M. Andreesen, "NCSA Mosaic Technical Summary; http://web.archive.org/web/19991009182307/http://cbl.leeds.ac.uk/WWW/ps/mosaic.orig.ps," 1993.

[8] C. Allison, M. Bramley, and J. Serrano, "The World Wide Wait: Where Does the Time Go?," in *Euromicro98:Engineering Systems and Software for the Next Decade*, Vasteras, Sweden, 1998, pp. 932-940.

[9] A. Ruddle, C. Allison, and R. Nicoll, "Analysing the Latency of WWW Applications," *Software Practice and Experience*, vol. 33, pp. 1301-1322, 2003.

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "RFC 2068: Hypertext Transfer Protocol -- HTTP/1.1," 1997.

[11] B. Krishnamurthy, J. C. Mogul, and D. M. Kristol, "Key differences between HTTP/1.0 and HTTP/1.1; http://www8.org/w8-papers/5c-protocols/key/key.html," in *8th Int. World-Wide Web Conference; http://www8.org/w8-papers/5c-protocols/key/key.html*, 1999 pp. 659-673.

[12] Sandevine Report, "Encrypted Web Traffic More Than Doubles After NSA Revelations; http://www.wired.com/2014/05/sandvine-report/," in *Wired*, ed: Wired, 2014.

[13] A. Arnbak, H. Asghari, M. V. Eeten, and N. V. Eijk, "Security collapse in the HTTPS market," *Commun. ACM*, vol. 57, pp. 47-55, 2014.

[14] M. Belshe, R. Peon, E. Thomson, and A. Melnikov. (April 2014). Hypertext Transfer Protocol version 2.0 (draft-ietf-httpbis-http2-04); https://tools.ietf.org/html/draft-ietf-httpbis-http2-04.

[15] D. Stenberg, "HTTP2 explained," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 120-128, 2014.

[16] "Average Web Page Breaks 1600K; http://www.websiteoptimization.com/speed/tweak/average-web-page/," in *http://www.websiteoptimization.com/*, ed, 2014.

[17] B. Thomas, R. Jurdak, and I. Atkinson, "SPDYing up the web," *Commun. ACM*, vol. 55, pp. 64-73, 2012.

[18] M. Belshe, "More Bandwidth Doesn't Matter (much); https://docs.google.com/a/chromium.org/viewer?a=v&pid=sites&srcid=Y2hyb21pdW0ub3JnfGRldnxneDoxMzcyOWI1N2I4YzI3NzE2," 2010.

[19] M. Belshe, "SPDY Protocol; http://mbelshe.github.io/SPDY-Specification/draft-mbelshe-spdy-00.xml," 2012.

[20] Akamai, "The State of the Internet, 3rd quarter, 2014".

[21] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol version 2, draft-ietf-httpbis-http2-17; https://tools.ietf.org/html/draft-ietf-httpbis-http2-17," IETF February 2015.

[22] I. Grigorik, "Making the web faster with HTTP 2.0," *Commun. ACM*, vol. 56, pp. 42-49, 2013.

[23] J. Khalid, S. Agarwal, A. Akella, and J. Padhye. (2014) Improving the performance of SPDY for mobile devices.

[24] R. Peon, H. Ruellan, and H. W. _Group, " HPACK - Header Compression for HTTP/2 draft-ietf-httpbis-header-compression-12," 2015.

[25] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How speedy is SPDY?," presented at the Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, Seattle, WA, 2014.

[26] Kamp, P-H, "HTTP/2.0: the IETF is phoning it in," *Commun. ACM*, vol. 58, pp. 40-42, 2015.

[27] E. R. Harold, *Java Network Programming, 4th Edition: Developing Networked Applications*, 4th ed.: O'Reilly Media, 2014.

[28] L. Masinter, RFC 2324"Hyper Text Coffee Pot Control Protocol (HTCPCP/1.0)," IETF, 1st April 1998.

[29] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munaf, *et al.*, "The Cost of the "S" in HTTPS," presented at the Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, Sydney, Australia, 2014.

[30] Y. Elkhatib, G. Tyson, and M. Welzl, "Can SPDY Really Make the Web Faster?," *Proceedings of IFIP Networking 2014*, 2014.

[31] J. Padhye and H. F. Nielsen, "A comparison of SPDY and HTTP performance," Microsoft Research, 2012.

[32] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan, "Towards a SPDY'ier mobile web?," presented at the Proceedings of the ninth ACM conference on Emerging networking experiments and technologies, Santa Barbara, California, USA, 2013.

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

[34] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, *et al.*, "An argument for increasing TCP's initial congestion window," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 26-33, 2010.

[35] S. Souders, "Domain Sharding Revisited; http://www.stevesouders.com/blog/2013/09/05/domain-sharding-revisited/," ed, 2013.

[36] Google, "PageSpeed; https://github.com/pagespeed/mod_pagespeed," ed, 2015.

[37] D. Wetherall, "Future of HTTP; http://mediaplayer.pearsoncmg.com/_ph_cc_ecs_set.title.8-8_Future_of_HTTP__/ph/streaming/esm/tanenbaum5e_videonotes/8_8_http_future_cn5e.m4v," in *Computer Networks 5th edition supplementary video*, ed: Pearson, 2013.

[38] Riverbed, "Wireshark; https://www.wireshark.org/," ed, 2015.

[39] P. Meenan. (2015). *Web Page Test; http://www.webpagetest.org/*.

[40] C. Sun, "HTTP/2 and SPDY Indicator 2.2.1; https://addons.mozilla.org/en-us/firefox/addon/spdy-indicator/," ed, 2014.

[41] NSF_INRIA, "ns3; https://www.nsnam.org/," ed, 2015.

[42] ArchLinux, "Trickle; https://wiki.archlinux.org/index.php/Trickle," ed, 2014.

[43] Riverbed, "Riverbed Modeler Academic Edition (formerly OpNet IT Guru); https://splash.riverbed.com/community/product-lines/steelcentral/university-support-center/blog/2014/06/11/riverbed-modeler-academic-edition-release," ed, 2014.

[44] Linux, "Traffic Control; http://www.man-page.net/8/tc," ed, 2012.

[45] The_Linux_Foundation, "Netem; http://www.linuxfoundation.org/collaborate/workgroups/networking/netem," ed, 2009.

[46] Linux, "Queuing Discipline, qdisc; http://tldp.org/HOWTO/Traffic-Control-HOWTO/classless-qdiscs.html," ed, 2012.

[47] Google. (April 2015). *QUIC; https://www.chromium.org/quic*.

[48] Y. Elkhatib, G. Tyson, and M. Welzl. The Effect of Network and Infrastructural Variables on SPDY's Performance; http://arxiv.org/abs/1401.6508. *CoRR*, 2014.