TLS PERFORMANCE

Masterarbeit zur Erlangung des akademischen Grades

Master of Science

Verfasser: Richard Fussenegger, BSc

Vorgelegt am FH-Masterstudiengang MultiMediaTechnology, Fachhochschule Salzburg

Begutachtet durch:
Dipl.-Ing. Dr. Peter Meerwald (Betreuer)
Mag. (FH) Hannes Moser (Zweitgutachter)

Thal, 1. Juni 2015

Eidesstattliche Erklärung

Hiermit versichere ich, Richard Fussenegger, geboren am 27. Juni 1985 in Bregenz, dass ich die Grundsätze wissenschaftlichen Arbeitens nach bestem Wissen und Gewissen eingehalten habe und die vorliegende Masterarbeit von mir selbstständig verfasst wurde. Zur Erstellung wurden von mir keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Ich versichere, dass ich die Masterarbeit weder im In- noch Ausland bisher in irgendeiner Form als Prüfungsarbeit vorgelegt habe und dass diese Arbeit mit der den BegutachterInnen vorgelegten Arbeit übereinstimmt.

Thal, am 1. Juni 2015

Fussenegger Richard

Richard Fussenegger

1210695017

Personenkennzeichen

Kurzfassung

Verschlüsselte Kommunikation ist ein *Muss* im Internet, die neuen Protokolle SPDY und HTTP/2 verpflichten quasi sogar dazu. Eine wichtige Frage die bleibt, ist, wie stark die Verschlüsselung sich auf die Leistung der Webkommunikation auswirkt. Vor etwas mehr als zehn Jahren variierte der Mehraufwand für den Einsatz von Transport Layer Security (TLS) zwischen einem Faktor von 3,5 und 9 gegenüber einer unverschlüsselten Kommunikation. Die Anforderungen an verschlüsselte Kommunikation sind in den letzten Jahren stark angestiegen und sehr viele Verfahren haben sich als unsicher und problematisch erwiesen.

Schlüssellängen sind angewachsen und aufwendigere Verfahren kommen zum Einsatz für den Schlüsselaustausch. Es stellt sich somit die Frage ob schnellere Hardware, Webserver, Algorithmen und Implementierungen – wie damals angenommen – den Mehraufwand verschwindend gering machen, oder gegenwärtige Technologien nicht mit den erhöhten Anforderungen mithalten können. Dies wird anhand von einer umfangreichen Leistungsmessung evaluiert. Dabei soll im Besonderen der tatsächliche Einfluss von TLS ermittelt werden.

Schlagwörter:
Leistung, Sicherheit, Webserver, WWW, Internet, TLS, SSL

Abstract

Encrypted communication is a *must* in the Internet, the new protocols SPDY and HTTP/2 quasi oblige it. An important question that remains is how strong the encryption affects the performance of the web communication. Little more than a decade ago, the overhead for the usage of Transport Layer Security (TLS) varied between a factor of 3.5 and 9 compared to an unencrypted communication. The requirements for encrypted communications have increased in recent years, and many algorithms have proven to be insecure and problematic.

Key lengths have grown, and more elaborate methods are necessary for key exchange. The question arises whether faster hardware, web servers, algorithms and implementations —as assumed back then—can make the overhead negligible, or if current technology is not able to keep up with the elevated requirements. This will be evaluated on the basis of extensive performance tests. Special focus lies on determining the actual influence of TLS.

Keywords:
Performance, Security, Web Server, WWW, Internet, TLS, SSL

Acknowledgment

First, and foremost, I would like to thank my girlfriend for her sustainable support and her unconditional love.

I would also like to thank my advisor PhD Peter Meerwald for being extremely companionable, patient, and of great help while compiling my thesis.

Contents

1 Introduction

Performance and security are both crucial parts of any application. But a secure operation comes with additional costs. A simple example would be a safe compared with a wallet. The safe has high security, but it takes a long time to get the money. The wallet has little security, but it takes a short time to get the money. This effect is true for almost every operation. The problem is that most web applications cannot afford long processing times.

Research suggests that website performance is the most important key factor for users (Podjarny 2010; Everts 2014). One result of this user preference is that most web servers are still configured to handle connections over insecure communication channels under the pretext of efficiency arguments. Only 0.5 %[1] of all websites were offering encrypted transactions in 2013, and the HTTP Archive statistic reports that 16 % of all websites they observed during their elicitation in May 2015 were performed over a secured channel (HTTP Archive 2015). However, the world at large knows by now that security is very important in the World Wide Web (WWW) for various reasons.

In the past, the topic of security was ignored by the public, despite the fact that developers, hackers, and organizations were appealing urgently to improve security (Sebayang 2013; EFF 2011). This changed in the year 2013 when the mass media started to report on the National Security Agency (NSA) leakages made public by Edward Snowden (Gidda 2013). Many espionage activities from various intelligence agencies around the world are now being revealed at a constant rate (Holland 2014), and at the same time cybercrime is increasing (Brandt 2014).

Transport Layer Security (TLS) was specifically designed to provide secure communication over an insecure infrastructure, and, thus, to prevent eavesdropping and other inadvertent actions (Ristić 2014, 1). But, for the above reasons, broad deployment—let alone full encryption—of the WWW is still far away. Hence, users and servers are vulnerable to simple attacks by authorities and criminals alike. Hopefully, this aggravation will change in the near future.

Full encryption will become de facto mandatory with the new Hypertext Transfer Protocol 2 (HTTP/2) since major browsers, like Firefox from the Mozilla Foundation (Mozilla) and Chrome from Google Incorporation (Google), are not going to support clear, or unencrypted, communication (McManus and Hurley 2014; Chan 2014). In fact, Mozilla

1. Estimation based on (Durumeric et al. 2013, 4) and (Netcraft 2013) for all available IPv4 addresses connected to the Internet; excluding SNI enabled hosts.

even plans to deprecate any non-secure communication (Barnes 2015). Google expressed similar plans on various occasions, and wants to start displaying warnings for non-secured origins (Google, Inc. 2014).

Google already deployed their experimental HTTP/2 precursor protocol SPDY (pronounced "speedy") with mandatory encryption. Their decision to use secured communication channels for this protocol was initially based on compatibility issues with existing intermediaries (Google, Inc. 2009; Grigorik 2013a, 53; Wang et al. 2014, 389). Such intermediary nodes along a communication path may drop messages because of an unknown formatting, or alter and subsequently cripple the messages to a point at which they are unusable. Encrypted messages, on the contrary, are unreadable for intermediaries by definition, and alterations are impossible.

Encryption allowed Google the easy deployment of a new protocol within the existing infrastructure of the WWW and the performance of various live tests. However, additional cryptographic processing is necessary. There are many more things that get incapacitated or are harder to achieve if encryption is involved. This is the reason why the HTTPbis working group's discussion regarding mandatory TLS usage in HTTP/2 lead to no consensus (Stenberg 2015). Further, there are many other points in the new protocol that are criticized by various parties; a comprehensive list was compiled by (Kamp 2015).

To strengthen the encryption of the future an update to the TLS standard in form of version 1.3 is underway as well (Rescorla 2015b). The TLS working group's objectives are to improve security and performance. The re-evaluated text shall remove ambiguity, and introduce new, stronger cipher suites while prohibiting usage of insecure ones. All of these goals shall be reached with maximum backward compatibility. Thus, the minimization of the processing times involved in establishing a secure channel will only be achieved through optimizing, removing, or moving of certain components.

(Coarfa, Druschel, and Wallach 2002, 11) concluded in their research that TLS overhead will diminish with increasing central processing unit (CPU) performance, and that research should focus on more efficient web servers and the TLS connection setup phase. Nowadays, CPUs are significantly faster and feature multi-core architectures, protocols are finally evolving again addressing round-trip time (RTT) issues, and new web servers plus operating system (OS) kernels are available addressing the connection setup phase problems. If the conclusion of Coarfa, Druschel, and Wallach holds true after nearly twelve years, and if industry and research are on the right track to reach these goals, is unacknowledged.

The research question of this thesis is, therefore: *Does it hold true that new, optimized algorithms and their implementations as well as faster computers reduce the overhead of encryption to such a great extent that there is hardly a difference to unencrypted communication?*

1.1 Contribution

The primary focus of this thesis lies on evaluating the status quo and impending development of TLS with emphasis on performance. The contribution of this work to the field of research can be split up into three parts:

- Chronicle of Internet encryption and security development with attention to performance. This includes existing and upcoming technologies—protocols and standards—that improved and should improve speed.
- Theoretical and practical evaluation of the impact that encryption has on web communication through the development of a benchmarking plan based on existing and widely deployed technologies. The main question of this thesis will be answered through statistical analysis of the test results.
- Development of a program for key rotation that can be used together with a web server for de- and encryption of session tickets. This small program helps to improve both security and performance of one or more web servers.

1.2 Outline

Following this introduction is the methods section, which is devoted to related technologies as well as academic and non-academic research. The methods are important to locate and understand the role of encryption within web applications and its performance impact. Subsequent subsections deal with the internet protocol suite (IPS) layers in ascending order. A brief guide to cryptography follows as an introduction to the following subsections that take a closer look at the TLS ecosystem. Current as well as historic work of other researchers on the main topic of this thesis are discussed in the last subsection of methods.

The results section follows, which is dedicated to the presentation and discussion of the work that was performed as part of this thesis. The outcome will answer the research question. The first subsection is going to show the setup for the tests that is used, followed by a purposeful explanation of the HTTP Archive (HAR) file format. A statistical analysis of the collected test data is performed in the subsequent passage followed by the

last subsection which documents the development of a program for TLS session ticket key rotation.

The final and last section of this work is dedicated to the discussion of the performed work and an outlook for possible future extensions based on it.

1.3 Preliminary Remarks

All mathematical equations are following the same style by using only lowercase letters for variables and uppercase letters for sets. Names for mathematical variables are kept consistent throughout this work; see section 4 for reference purposes.

2 Methods

This section clarifies the fundamentals of underlying technologies in necessary detail. This knowledge is required in order to evaluate what problems new protocols and standards try to solve. Each subsection, therefore, includes a brief chronology of previous developments in which problems in regard to performance are highlighted.

Section 2.1 is a terse breakdown of limitations that are imposed by physics on computer networking, which in further consequence influence the speed of the Internet. The following section 2.2 gives a brief introduction to the IPS by clarifying which role the Internet Protocol (IP) and Transmission Control Protocol (TCP) play. Going further up the stack, HTTP follows, it builds the core of almost all web services and is of outermost importance for performance. A brief primer to cryptography is given before continuing with TLS. A chronology of the protocol's evolution concerning performance and security where applicable is given. The following section takes a detour by identifying TLS related technologies that are important for the encrypted ecosystem. The very last section dissects previous work on the subject, and builds the fundamental for the next part of the thesis.

2.1 Physical Limitations and Networking Primer

Various key figures directly impact the speed of any computer network and determine its performance (Cermak 2005). It is desirable for a web application that all of these measures are at their optimum for fast delivery.

- Latency is the time a packet takes from the sender to the receiver.
- Jitter is the variation in latency of packets over time.
- Error rate is the percentage of incorrectly transmitted bits.
- Bandwidth is the maximum throughput of a communication channel.
- Throughput is the total of packets successfully delivered.

The two most important key factors to understand in the context of a fast web application are latency and throughput/bandwidth because they can be directly influenced (Grigorik 2013a, 3). All other measures are dependent on the communication channel and hard to govern, especially in a huge network like the Internet. Of course, the latency and throughput/bandwidth can only be influenced on the controlled side (usually the server), and the other party's values will vary depending on various factors that are out of control for administrators and developers of websites.

2.1.1 Latency

Packets, which are sent over an internet[2], traverse multiple nodes and suffer from several types of delays at each node during that progress. This effect is known as latency, an inevitable principle of any network. The most significant delays are nodal processing, queuing, transmission, and propagation. Their sum gives the total nodal delay (d_{nodal}), and, thus, the latency of a communication path (Grigorik 2013a, 4–5; Kurose and Ross 2013, 35–47). The following paragraphs give further details about each delay, and are arranged according to their occurrence.

Nodal processing delay d_{proc} is the amount of time required to examine a packet's header. Bit-level error checks are usually performed by routers and the next destination for the packet to reach its receiver is determined. A router's processing time is typically within microseconds but may add up to milliseconds if complex payload modifications (namely network address translation (NAT) or deep packet inspection (DPI)) have to be performed (Ramaswamy, Weng, and Wolf 2004, 1630, Table I).

Queuing delay d_{queue} is the amount of time a packet is waiting before it can be transmitted onto the link. This delay usually occurs at routers, switches, and statistical multiplexers. Such a devices can only reconsign a single packet at a time, hence, placing all other incoming packets in a queue (buffer). Therefore, this delay may approach infinity if the rate of incoming packets exceeds that of outgoing ones. It follows that the link is now congested because the device only has a finite amount of buffer available to stack the incoming packets (Ramaswamy, Weng, and Wolf 2004, 1630, Table I; Kurose and Ross 2013, 41, Figure 1.18).

The congestion problem is illustrated in figure 1 with a 1 Gbit/s switch and three attached computers. Computer A and B continuously send 1 Gbit/s to computer C, thus, exceeding the switch's queue and congesting the link (Comer 2014, 30–31). The communication path is subsequently going to suffer from packet loss, which might trigger a domino effect because lost packets are eventually retransmitted by the sender. This means in effect that the sender will consequently send even more packets than before the packet loss was experienced. This effect is called congestion collapse. The TCP provides special algorithms to keep this issue as small as possible; see section 2.2.4 for more details (RFC 896, Nagle 1984).

2. An internet—note the lower-case *i*—is an amalgamation of networks through gateways, and short for internetworking, which derives from inter (between) and networking. A famous example for an internet besides the Internet would be the Advanced Research Projects Agency network (ARPANET), which is now a subnet of the Internet.

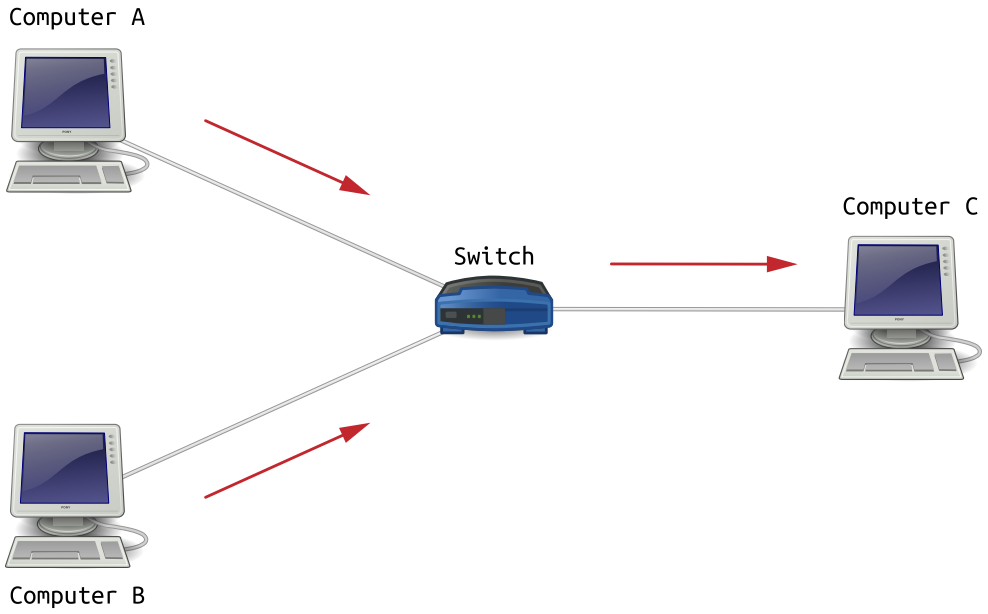


Figure 1: Switch with three computers attached, and arrows that indicate in which direction the streams move (Fussenegger 2014e; Kurose and Ross 2013, 36, Figure 1.16).

Transmission delay d_{trans} is the amount of time necessary to push (transmit) all of the packet's bits into the link. Packet-switched networks are commonly transmitting in a first-come-first-serve manner, meaning that a packet can only be transmitted when all other received packets have already been transmitted. The formula to calculate the transmission delay is defined as (Kurose and Ross 2013, 37):

$$d_{\text{trans}} = {}^{l}\!/_{r} \tag{2.1}$$

With l denoting the length of the packet in bit, and r the transmission rate in bit/s. As a result, if the same 100 Mbit file is to be transfered over a 100 Mbit/s and a 10 Gbit/s link, then it takes 1 s to push the file on the former link and only 10 ms on the latter (Grigorik 2013a, 5).

Propagation delay d_{prop} is the amount of time it takes for a signal's head to travel from the sender to the receiver. This value is derived from the distance between sender and receiver, and the medium that is used to transport the signal. The speed of the signal is dictated by the universal physical constant of the speed of light c. How fast light can travel depends on the medium it has to go through. This can be calculated by the index of refraction n, which is a dimensionless ratio that depicts "the apparent phase velocity of the transmitted beam from its nominal value of c" (Hecht 2002, 92, 4.2.3). The formula

to calculate the refractive index is, therefore, defined as:

$$n = {}^{c}\!/_{v} \tag{2.2}$$

$$v_{\mathrm{FOC}} = \frac{299{,}792{,}458\,\mathrm{m/s}}{1.5} = 199{,}861{,}638.666{,}667\,\mathrm{m/s} \approx 200\,\mathrm{Mm/s} \tag{2.3}$$

The n of a vacuum is one (by definition) and, thus, light will travel through it at the exact speed of 299,792,458 m/s (nominal value). The typical n of fiber optic cables (FOCs) is between 1.34 and 1.54 or worse, depending on the quality and the question whether the index refers to the core or the cladding (Ishigure, Nihei, and Koike 1996, 2049; Hecht 2002, 194). It is, therefore, safe to use a value of ∼1.5 for estimations, like in equation (2.3), which results in a speed v of ∼200 Mm/s for an average FOC, and allows the calculation of the actual propagation delay (Grigorik 2013a, 6):

$$d_{\mathrm{prop}} = {}^{s}\!/_{c_m} \tag{2.4}$$

$$d_{\mathrm{prop}} = \frac{6{,}800\,\mathrm{km}}{200{,}000\,\mathrm{km/s}} \times 1{,}000 = 34\,\mathrm{ms} \tag{2.5}$$

With s denoting the distance traveled, and c_m the speed of light in the medium (propagation speed). This results in a d_{prop} of 34 ms between New York City (United States) and Graz (Austria)[3] or a RTT of 68 ms through a direct FOC connection. Even in a vacuum the RTT would still take up to ∼46 ms without any of the other delay components considered, see equation equation (2.5). Since there is no direct connection and several nodes are between both locations, a real ping results in a RTT of ∼200 ms.[4] In other words, the d_{nodal} is ∼100 ms after all aforementioned delays are aggregated.

While 200 ms does not sound like much, research suggests that a 300 ms delay is already perceived by the user, and that there is a 1 s to 2 s barrier of maximum acceptable response time for systems, after which a user will perform a mental context switch (Grigorik 2013a, 7; Seow 2008, 40–45; Nah 2004; Shneiderman 1984; Miller 1968). The speed of signals in copper or twisted-pair cables is often slower because they are “subject to higher signal

3. The distance information was taken from http://www.entfernungsrechner.net/ and Graz was chosen because the author owns a server in that city with a gigabit uplink.

4. The ping was executed with http://startping.com/city/NewYork to https://www.movlib.org/ (Graz).

loss, and electromagnetic interference" (Grigorik 2013a, 9). However, telecommunication corporations around the world are eventually transitioning to optical fiber everywhere, including fiber to the home (FTTH) (Sawall 2014; OECD 2008; Hecht 2002, 196).

This upgrade will have a significant impact on user performance since the most substantial loss of time for routing packets predominates at the user's side—the well known last kilometer or mile. This last kilometer adds between 5 ms to 40 ms, and accounts for 40 % to 80 % of the whole incurred latency. This is because local Internet service providers (ISPs) have to deploy the cables through many households and aggregate the signal, which is then forwarded to the local routing node. From this point on, the packets travel through high-speed cables and routers until they reach the web server, which is usually also situated in a high-speed data center (Sundaresan et al. 2011, 142; Canadi, Barford, and Sommers 2012, 284).

2.1.2 Throughput

Another critical key factor in computer networks, besides latency, is end-to-end throughput. The instantaneous throughput,for example, is displayed in peer-to-peer (P2P) software while downloading a large file; it describes the rate at which the packets are received at any given point in time measured in bit/s. The average throughput $\overline{r}$ is made up of the total bit count x that was transferred, and the time t it took to transfer the file (Kurose and Ross 2013, 44). The formula is, therefore, simple:

$$\overline{r} = {}^{x}\!/_{t} \tag{2.6}$$

Certain Internet applications require a minimum throughput to work correctly, such as Web Real-Time Communication (WebRTC) audio requiring between 6 kbit/s to 510 kbit/s, and WebRTC video requiring 100 kbit/s to 2,000 kbit/s or more depending on the quality (Grigorik 2013a, 314–15). File transfers do not have minimum requirements, but small throughput will result in long download times. Mike Belshe from Google shows that the benefit of increasing maximum throughput (bandwidth) on the client side is approaching zero at around 8 Mbit/s, and that latency (RTTs) is more important for fast web applications than it might be for other Internet applications (Belshe 2010; Grigorik 2013a, 176–78).

Rank	Country/Region	Average Mbit/s
–	Global	4.5
1	South Korea	22.2
2	Hong Kong	16.8
3	Japan	15.2
4	Sweden	14.6
5	Switzerland	14.5
6	Netherlands	14.2
7	Latvia	13.0
8	Ireland	12.7
9	Czech Republic	12.3
10	Finland	12.1

Rank	Country/Region	Peak Mbit/s
–	Global	26.9
1	Hong Kong	87.7
2	Singapore	84.0
3	South Korea	75.4
4	Japan	69.0
5	Romania	67.0
6	Taiwan	64.2
7	Uruguay	63.3
8	Qatar	62.8
9	Israel	60.5
10	Latvia	60.2

Table 1: Average and maximum throughput, as observed from Akamai Technologies' content delivery network (CDN), globally and for the top ten countries and regions of the world (Belson 2014, 20, Figure 11–12)

Table 1 shows the average and maximum client throughput compiled by Akamai Technologies for various regions of this world. Most users already have enough throughput with a global average of 4.5 Mbit/s for high-speed web browsing if the observations of (Belshe 2010, 3) are used as a benchmark. But the situation, of course, is different for web servers: they usually should simultaneously serve content to as many clients as possible while still enabling all of them to utilize their throughput efficiently.

$$u_{r_{\max}} = r_{\mathrm{bl}} \big/ u \tag{2.7}$$

The bandwidth for each of u clients connecting to a server over a network is dictated by the throughput of the bottleneck link r_{bl}. Each client will typically have the same maximum throughput if fair queuing (RFC 970, Nagle 1985) with best-effort delivery is in use; see equation (2.7). Since the Internet core is sufficiently "over-provisioned with high speed links that experience little congestion" (Kurose and Ross 2013, 45), only two nodes can be the bottleneck: the client or the server.

2.1.3 Quality of Service

Quality of service (QoS) refers to the overall performance of all components of the preceding sections and networks in general. Packet switched networks—like any internet and the Internet—are based on best-effort delivery and, thus, guarantee no QoS whatsoever. This means that bit rates and delivery times are unspecified and depending on other network participants. But a minimum amount of QoS might be preferable for high throughput applications, like audio and video streaming or even file downloads (Comer 2014, 549).

Such a QoS system would have to guarantee that all network participants have the same characteristics at all times. It means in effect that if both computers A and B in figure 1 want to send a 1 Mbit/s stream to computer C over the 1.5 Mbit/s link, one of A and B would have to be declined by the switch. Otherwise, it would be impossible to satisfy the QoS of any of the participants; in fact, this is exactly what is done in telephone networks (Kurose and Ross 2013, 654).

> *When a network has sufficient resources for all traffic, QoS constraints are unnecessary; when traffic exceeds network capacity, no QoS system can satisfy all users' demands.*
>
> — (Comer 2014, 550)

A lot of research went into this field of study, and even standards like Asynchronous Transfer Mode (ATM) were developed, but nothing found broad deployment (Comer 2014, 549–56; Kurose and Ross 2013, 636–55). Instead, most ISPs use traffic shaping for rate limiting. The scheduling (outgoing data) and policing (incoming data) in use highly varies from ISP to ISP (Sundaresan et al. 2011, 144). The same might be true for some Internet hosting services (IHSs), and it is, therefore, important to read the terms of service (ToS) or ask how traffic is distributed among multiple servers.

Encrypted traffic, like TLS, is by definition unreadable for QoS implementations, and, thus, unclassifiable for the algorithms that try to automatically apply rules. How ISPs handle these situations is unclear, but throttling all encrypted traffic is reasonably hazardous because most top[5] and e-commerce websites use TLS to secure their users and applications. As a result, some parties demand that users of certain encrypted traffic from services like virtual private networks (VPNs) who use a lot of bandwidth should be considered harmful or even criminal (Andy 2014).

2.2 Internet Core Networking

The core of the Internet is, above all, driven by the IP and the TCP, which were first proposed by (Cerf and Kahn 1974). Other protocols exist, but none reached the same importance as those two, which becomes exemplified by the fact that the internet protocol suite (IPS) is commonly referred to as TCP/IP. Many changes were proposed and made to the core protocols since the first version from 1974, but they are essentially still operating in the same manner today (Grigorik 2013a, 13).

5. Referring to the "Alexa Internet Top Sites" at http://www.alexa.com/topsites.

However, the User Datagram Protocol (UDP) is getting more attention nowadays due to its low overhead for high bandwidth applications. This is the reason why, for example, WebRTC was standardized for UDP only. Google is experimenting with a complete replacement for TCP called Quick UDP Internet Connections (QUIC; Roskind 2013), and plans to submit it to the Internet Engineering Task Force (IETF) in the near future (Wilk, Hamilton, and Swett 2015). They are not the first to do so but might have the leverage to actually push it through, like they did with SPDY, which resulted in HTTP/2.

The IPS is often compared with the Open Systems Interconnection model (OSI model) but only defines four layers in contrast to the seven layers of the conceptual OSI model (Comer 2014, 50). The link layer is the lowest and does not describe any physical standards, instead, a functioning network infrastructure is simply assumed and not discussed in RFC 1122 or RFC 1123 (Braden 1989b, 1989a)[6]. Instead, the link layer standard merely describes protocols that act only on the link (local network segment) the host is connected to, also, only physical addresses[7] are used, and nothing is routed to outside networks (Comer 2014, 62–63; Kurose and Ross 2013, 49–53).

This is where the internet layer with IP and the transport layer with TCP take over followed by the application layer, which is also the last. The most important protocols of these layers are discussed in subsequent sections 2.3 to 2.5. There are numerous other protocols that assist to drive the Internet core. For instance, the Domain Name System (DNS) or the Border Gateway Protocol (BGP) that support participants to find each other in networks. Both of these protocols are insecure, manipulable, and can be used to bypass encryption by rerouting the traffic (Ristić 2014, 18–22). A very famous BGP incident, for instance, involved a Pakistani ISP that accidentally hijacked all YouTube traffic (Antony et al. 2008). Hence, both protocols are subject of past and ongoing research and new extensions, which use cryptography and other measures to secure them.

There are many more non-core network protocols that are commonly used in OSs, and might be misused to circumvent security measures as well. A recent attack involved such a protocol, the Network Time Protocol (NTP), to bypass HTTP Strict Transport Security (HSTS, see section 2.6.2) and perform a so called Secure Socket Layer (SSL) stripping attack. Essentially, this means that the time of the victim's computer was changed to a time far in the future to skirt encryption, and force the victim to visit an unencrypted network resource (Selvi 2014). Bypassing encryption to execute attacks is typical because

6. The lack of physical standards often tempts authors to add additional layers to the IPS, like seen in (Comer 2014, 53; Kurose and Ross 2013, 50).

7. Hardware media access control (MAC) addresses, not to be confused with message authentication code, which has the same abbreviation but is used in the context of cryptography.

it is harder to break underlying encryption than to simply remove it entirely (Ristić 2014, 151).

2.2.1 TCP/IP

IP provides the ability to route and address datagrams (packets) through gateways across network boundaries. Currently, two IP standards are deployed, namely version 4 (IPv4) and 6 (IPv6)[8]. TCP provides a reliable packet delivery over an unreliable communication channel. This means that it ensures "retransmission of lost data, in-order delivery, congestion control and avoidance, data integrity and more" (Grigorik 2013a, 13). The main focus of the protocol lies not on fast but on accurate delivery. Besides that, both protocols do not provide any security measurements, and allow any network participant, who has access to the link, to intercept and manipulate the data on the wire (Grigorik 2013a, 13; Ristić 2014, 2).

Transmission of the data and efficient division of it into segments to avoid network delays is what TCP is responsible for. Those segments are handed over to the IP interface, where each is wrapped in an IP packet and forwarded to the link interface, where it starts its way to the designated receiver. But TCP has to ensure that a communication with the desired receiver is actually possible before starting the transmission. Otherwise, it would be impossible to determine if all data will be reliably received. TCP uses a three-way handshake (3WHS) to achieve this.

Figure 2 illustrates TCPs 3WHS, where the client initially has to request a connection (`SYN`) from the server; this will take 34 ms, which is the delay that was previously calculated in section 2.1.1 from New York to Graz over a hypothetical direct FOC connection. The server will acknowledge the connection request (`SYN-ACK`) if it is listening on the port the connection request is sent to, and if connections are allowed. This adds another 34 ms only to send the acknowledgment back. From there on, sender and receiver are able to exchange application data (`ACK`), like downloading an image.

This whole process results in a significant overhead for establishing new TCP connections. Various researchers tried to propose a system to shorten this process, but the problem of further decreased security draws most attempts impractical. The most notable of these attempts was RFC 1644 (Braden 1994). (Radhakrishnan et al. 2011) were the first to introduce a system—called TCP Fast Open (TFO)—that works for recurring con-

8. IPv6 was developed by the IETF because of the long foreseen IPv4 address exhaustion. Deployment of the new standard is still slow despite many efforts of various parties, like the Internet Society (ISoc) with their http://www.worldipv6launch.org/ campaign.

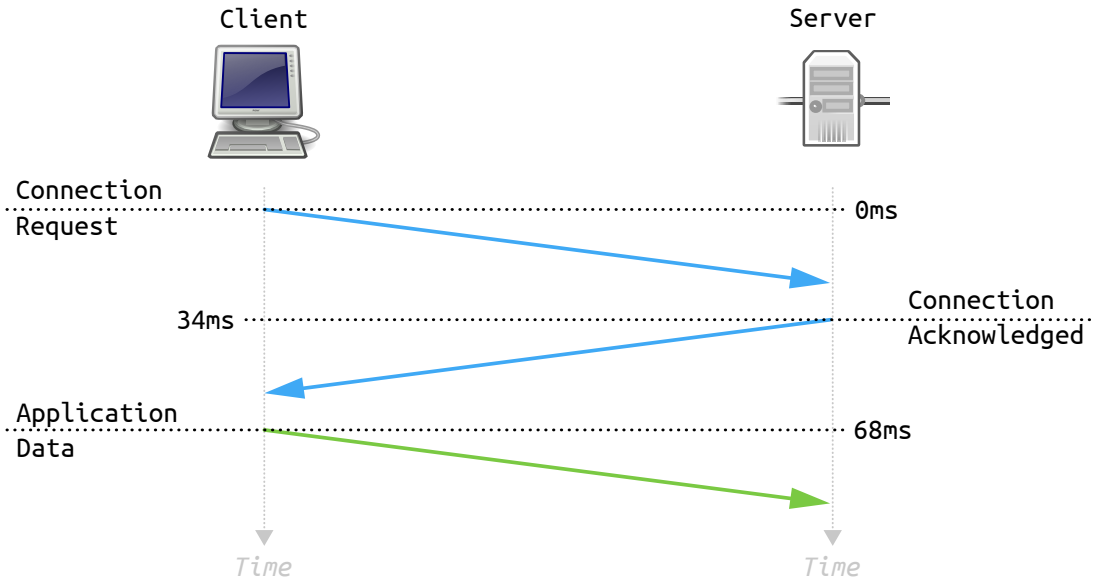


Figure 2: Simplified illustration of the TCP 3WHS with timing information from the example calculation in section 2.1.1 (Fussenegger 2014a).

nections by setting a cryptographic HTTP cookie without introducing new attack vectors. TFO proved to be very effective for subsequent connections over channels with high RTTs ranging from 5 % to 40 % in average for the whole page load time (PLT). This effect is less distinct with low RTTs (Radhakrishnan et al. 2011, 9, Table 1).

Currently TFO is only supported on Linux OSs starting from kernel version 3.7[9] for servers, enabled by default since version 3.13[10], and with IPv6 support since version 3.16[11]. Google's Chrome and Chromium browsers are currently the only user agents (UAs) that support the experimental RFC 7413 on Linux OSs (including ChromeOS and Android; Chan 2013a; Cheng et al. 2014). Nginx support is available since version 1.5.8 via the optional parameter `fastopen` to the `listen` directive.[12] It is not clear if the upcoming Microsoft Corporation (Microsoft) Windows OS or any of the future Apple Incorporation (Apple) OSs will implement this feature.

2.2.2 Flow Control

TCP—in contrast to UDP and other protocols—implements an end-to-end sliding window flow control mechanism, which ensures that the sender is not going to overwhelm the

9. http://kernelnewbies.org/Linux_3.7
10. http://kernelnewbies.org/Linux_3.13
11. http://kernelnewbies.org/Linux_3.16
12. http://nginx.org/en/docs/http/ngx_http_core_module.html#listen

receiver with too much data, thus, overflowing its network buffers. No other approach is possible since the internet layer with its IP does not provide any feedback regarding the current state of network congestion (Kurose and Ross 2013, 266). Each side of the communication has to include a receive window (*rwnd*)[13] value during the initial 3WHS and in subsequent `ACK` packets. During the inital handshake the initial receive window (*init_rwnd*) is set to the OS's default value, which highly varies between vendors, as illustrated in table 2.

OS	% > 15 kB	$\overline{rwnd}$
FreeBSD	91 %	51 kB
iPhone	66 %	87 kB
Linux	6 %	10 kB
Mac	93 %	270 kB
Windows 7	88 %	41 kB

Table 2: Advertised *init_rwnd* values of first HTTP request; the OS was extracted from the HTTP UA header line (Dukkipati et al. 2010, 29, Table 2).

The *rwnd* variable indicates how much data the receiver's buffer is able to hold, and can be adjusted at any time to account for buffer fill level changes. A value of zero has a special connotation and signals the sender that it should stop emitting data because the receiver's buffer is full and no more data can be processed at this time. The *rwnd* may become a bottleneck in various situations. For instance, for clients downloading or streaming resources from a server the *rwnd* might be too small and, therefore, never reaches the maximum throughput.

On the other hand, if a client is uploading a resource to the server, the server's *rwnd* might become the bottleneck, although it most certainly has a very fast link and sufficient resources to handle the incoming data. But the server might not be able to advertise its *rwnd* to the client because the field within the TCP's packet is limited to a maximum value of 65,535 B. This creates a hard limit for applications, and is highly problematic in cases when a long fat network (LFN, pronounced "elephan") is in use. A LFN is a network with a bandwidth-delay product (BDP) that significantly exceeds 10^5 bit (RFC 1072, Jacobson and Braden 1988, 1); a definition that essentially applies to most broadband technologies in use today if the RTT is not very low.

$$b \times d_{\text{nodal}} = (10.4 \times 10^6\,\text{bit/s}) \times (100 \times 10^{-3}\,\text{s}) = 1{,}040{,}000\,\text{bit} \tag{2.8}$$

13. Sometimes abbreviated as *rwin* in older documents.

$$rwnd_{\min} = \frac{9.8 \times 10^6 \,\text{bit/s}}{8 \times 1024} \times 0.1\,\text{s} = 119.629\,\text{KiB} \tag{2.9}$$

The average throughput in Austria is 9.8 Mbit/s (and only 26 % of all inhabitants are over 10 Mbit/s, values according to Belson 2014, 42, Figure 31, 43, Figure 33), while a realistic RTT would be 100 ms. This results in a BDP of $\sim 10^6$ bit; see equation (2.8). As a consequence, the minimum *rwnd* has to be ∼120 KiB to utilize the link's full throughput; see equation (2.9). This is why RFC 1323 introduced the window scale option, which raises the *rwnd*'s TCP field's hard upper limit to 1 GB (Jacobson, Braden, and Borman 1992, 8–11). This option is active on all modern OSs. Listing 1 illustrates how this value can be printed and changed on a Linux OS (Grigorik 2013a, 18–19, 28–30).

```
sysctl -- 'net.ipv4.tcp_window_scaling'            # print
sysctl --write -- 'net.ipv4.tcp_window_scaling'=1  # set
```

Listing 1: Printing and setting the TCP window scaling option.

One should not simply raise the *init_rwnd*'s value to increase throughput as it might have negative consequences in some situations. But (Dukkipati et al. 2010) displayed in their study that it is safe to increase the *init_rwnd* multiplier to at least ten. This will, in turn, allow participants to reach a higher throughput. Nonetheless, the greatest challenge for a heterogeneous network, like the Internet, is the communication through gateways and other intermediaries of unknown configuration that lie beyond its network boundaries, and are not taken into account by the flow control.

While flow control ensures that the receiver is not overwhelmed, it does not consider the involved intermediaries, which might be unable to cope, along the communication path. The throughput needs to be steadily adjusted to a constantly fluctuating network environment. (Jacobson and Karels 1988) were the first to introduce a solution to this problem in their work "Congestion Avoidance and Control". They proposed various new algorithms, whose most important aspects are discussed in the following sections (Kurose and Ross 2013, 269–72; Grigorik 2013a, 16–30; Comer 2014, 223–25).

2.2.3 Slow-Start

Slow-start (Jacobson and Karels 1988, 2–5)[14] extends on the flow control idea by introducing the congestion window (*cwnd*) variable. It is used on the sender's side to determine

14. Similar approaches to the problem were independently proposed by other researchers, for instance (Jain 1986), since it was a well known problem, as can be seen in (RFC 896, Nagle 1984, 1–3).

how much unacknowledged data can be in flight to the receiver before it must stop sending and wait for an acknowledgment (`ACK`). The *cwnd* value is—unlike the *rwnd* value—not advertised to the counterside. Each side maintains its own value, which is initialized with a system-specific value. Adjustments to this private variable are done in accordance with a simple rule: "the maximum amount of data in flight (not `ACK`ed) between client and the server is the minimum of the *rwnd* and *cwnd* variables" (Grigorik 2013a, 20).

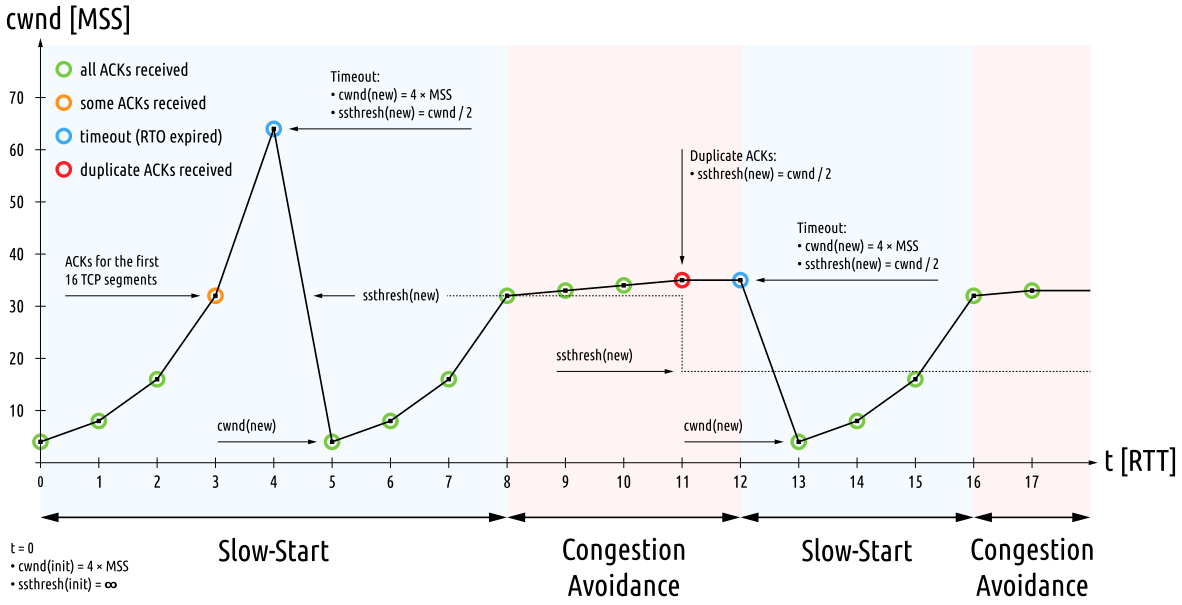


Figure 3: Illustration of TCP's slow-start and congestion avoidance (Fussenegger 2014b; Inacon GmbH 2010).

Figure 3 depicts a possible initial TCP communication with an initial congestion window (*init_cwnd*)[15] value of four. The exponential growth during the first RTTs is typical because both sides try to converge to the maximum throughput of the current connection as fast as possible. This phase ends if the *rwnd* is reached, a loss event occurs, or if the slow-start threshold (*ssthresh*) variable is greater or equal to the *cwnd*. It is important to note that the *ssthresh* variable has a very high value, infinity, or zero at the beginning, and is not set until the first loss event occurs after which it is set to $^{cwnd}/_{2}$ (Kurose and Ross 2013, 272–73).

$$t = \text{RTT} \times \left\lceil \log_2 \left(\frac{x}{init\_cwnd} \right) \right\rceil \tag{2.10}$$

15. *Init_cwnd* and *init_rwnd* values are always given as a multiplier for the MSS on Linux OSs.

The time that is necessary to reach the desired bandwidth can be calculated with equation (2.10), where x denotes the size in TCP segments (Grigorik 2013a, 21). The maximum segment size (MSS) must be known in order to convert the target *rwnd* to TCP segments. Therefore, an Ethernet network is assumed in the following example: the maximum transmission unit (MTU) is 1,500 B as defined in RFC 894 (Hornig 1984, 1)[16], and the MSS is 1,460 B (= 1500 − 20 − 20) after deducting presumed IPv4 headers. This results in 170 ms that are necessary to reach the desired throughput of 127 KiB from equation (2.9) if the previously calculated 34 ms from equation (2.3) are used as RTT; see the following equation (2.11).

$$t = 34\,\text{ms} \times \left\lceil \log_2 \left(\frac{\left\lceil \frac{127\,\text{KiB}}{1{,}460\,\text{B}} \right\rceil}{4} \right) \right\rceil = 170\,\text{ms} \tag{2.11}$$

This translates to five full round-trips between the client and the server, provided that all packets are actually able to go through and no additional processing time is necessary. In order to speed up the conversion any variable may be optimized. The RTT may be lowered by moving the participants closer to each other, or the *init_cwnd* may be increased. The former is closely tied to service costs and often only feasible for entities with enough capital, the latter may seem like something that can be easily done. But care must be taken when choosing the value of the *init_cwnd* because too great values might constantly overwhelm the underlying networks.

```
1 ip route | while read INTERFACE; do \
2   ip route change $INTERFACE initcwnd 10 initrwnd 10; \
3 done
```

Listing 2: Increase *init_cwnd* and *init_rwnd* on Linux with a kernel version of 2.6.33+.

However, the *init_cwnd* was first increased from one to four in RFC 2581, and was elevated anew to ten via RFC 6928 based on the proposition and worldwide tests from (Dukkipati et al. 2010; Allman, Paxson, and Stevens 1999; Chu et al. 2013). Linux has applied the proposed change of the *init_cwnd* to ten segments (IW10) since kernel 2.6.39[17], but the *init_rwnd* is still set to four. Listing 2 shows how both multipliers can be changed via the shell. The same code may be placed in `/etc/rc.local` to ensure that the values

16. The IETF prefers to use the term octet (unit *o*) to describe storage requirements within network protocols. This has historical reasons and is mainly used to avoid ambiguity, byte (B) and *o* are synonymous and both represent 8 bit. The usage of *o* is avoided in this thesis in favor of consistency.

17. http://kernelnewbies.org/Linux_2_6_39

persist during reboots.[18]

Another value that might has an influence on performance is slow-start restart (SSR), which is applied to idle TCP connections after a predefined period of time. The *cwnd* is discarded, and the whole slow-start process restarts. This is counter productive for long living TCP connections, which may idle for several seconds. Listing 3 illustrates how this value can be printed and changed on a Linux OS (Grigorik 2013a, 23).

```
1 sysctl -- 'net.ipv4.tcp_slow_start_after_idle'            # print
2 sysctl --write -- 'net.ipv4.tcp_slow_start_after_idle'=0  # set
```

Listing 3: Printing and setting the SSR option.

2.2.4 Congestion Avoidance

Congestion collapse was already mentioned in section 2.1.1 and is a serious problem for TCP with its automated retransmission of lost packets. But collapsing networks became a real problem within the growing internetworks long before TCP and the WWW, as the introduction of RFC 896 documents (Nagle 1984, 1–3). Many systems have been and are developed to deal with such effects. A comprehensive introduction to the topic can be found in (Kurose and Ross 2013), and is out of scope for this thesis.

The congestion avoidance—see red shaded areas in figure 3—comes into effect after the first loss event and after *cwnd* is greater or equal to *ssthresh*. This phase was also introduced by (Jacobson and Karels 1988, 8–11) and TCP's approach to avoid a possible collapse. The *cwnd* is not further doubled because it is assumed that this would directly provoke further congestion. Instead, it is only increased by a single MSS (RFC 5681, Allman, Paxson, and Blanton 2009) until the next loss event is encountered, and the whole process starts anew; see figure 4 for a finite state machine that illustrates the complete process. A considerable amount of different algorithms has been developed over the last few decades, where the recovery phase from packet loss is handled differently.

These algorithms include, but are not limited to, TCP Tahoe and TCP Reno (original implementations), TCP Vegas, TCP New Reno, BIC, CUBIC, and Compound TCP (Microsoft Windows). A new algorithm called Proportional Rate Reduction (PRR), which is not following the additive increase/multiplicative decrease (AIMD) principle, was proposed by (Dukkipati et al. 2011). This algorithm proved to decrease latency of short and bursty web transfers by 3 % to 10 %, while reducing timeout recovery as well by 5 %. PRR

18. Tested on Debian based systems, and might be different on other Linux distributions.

is the new default congestion avoidance algorithm in Linux since kernel 3.2[19] (Grigorik 2013a, 26–27; Kurose and Ross 2013, 274–78).

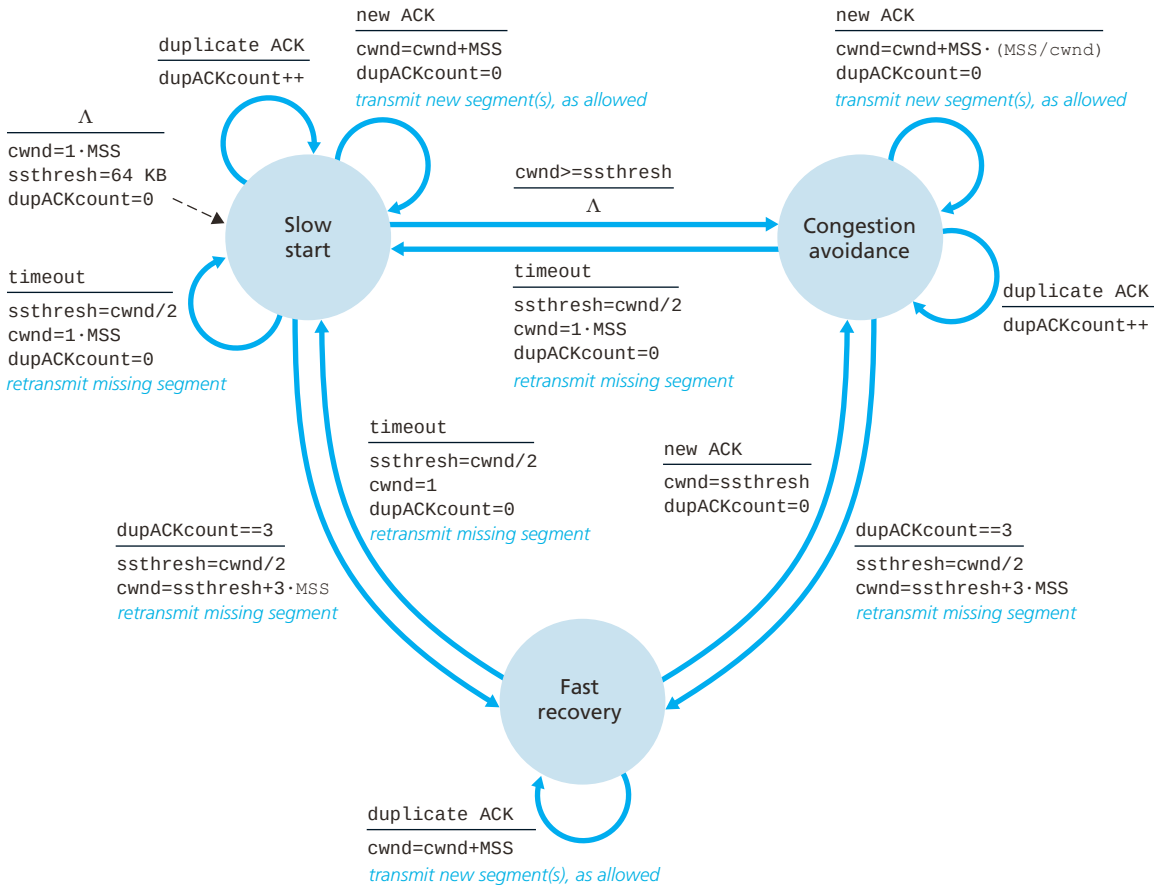


Figure 4: Finite state machine illustrating TCP's slow-start, congestion avoidance, and fast recovery coherences (recreated from Kurose and Ross 2013, 275)

2.2.5 Head-of-Line Blocking

TCP has to ensure that all packets always arrive in order. This means that the receiver has to keep all packets back in its buffer after a loss event and wait for the lost packet to arrive again. Otherwise, the packets would not be handled in order. The lost packet must be retransmitted before all subsequent packets can be sent. This problem is known as head-of-line blocking (HOL blocking) and results in unpredictable fluctuations in latency, commonly referred to as jitter. Web services like WebRTC (audio and video), which can deal with out-of-order delivery or packet loss, and are more delicate towards latency

19. http://kernelnewbies.org/Linux_3.2

variations, switched to UDP to overcome this phenomenon (Grigorik 2013a, 30–31).

2.3 Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) is one of the most important application layer protocols of the IPS and the Internet. It is, therefore, imperative to understand its mode of operation to achieve a fast web application. There are other protocols in use that work differently or build on top of HTTP, including WebSocket and WebRTC. However, these other protocols are not covered in this thesis. Almost all HTTP traffic is using the TCP and IP (see section 2.2) as communication protocols because of the reliable transport requirement that is guaranteed by their stack (Comer 2014, 528; Grigorik 2013a, 155; Kurose and Ross 2013, 98).

All versions of HTTP are working similarly at the core by functioning as a request and response protocol between client and server. The UA of the client submits a request message to the server, which returns a response. What is requested and responded varies and is implementation dependent, but a request or response may contain additional payload in its message body, such as an Hypertext Markup Language (HTML) document. How these request/response messages are comprised depends on the used HTTP version, but all versions operate through TCP streams that are established between the client's computer and the server.

2.3.1 HTTP/0.9: One-Line Prototype

The original HTTP, as such, was not defined, it was merely implemented as a proof of concept by Tim Berners-Lee for his vision of the WWW. The first outline of the prototype protocol can be found at (Berners-Lee 1991); it was very simple but sufficient.

The prototype features only a single method to request a response from a server—the `GET` method—followed by the full Uniform Resource Identifier (URI) or only its path of the desired document, terminated by two carriage return (CR) plus line feed (LF) combinations. The response is always an HTML page (Grigorik 2013a, 156; Gourley and Totty 2002, 16) or plain-text (Berners-Lee 1992a, 1992b), and the connection between the two parties is directly terminated after the document transfer is complete. The complete protocol is ASCII-only and running over a single TCP link.

Most of today's web servers still support this initial version of the HTTP, as can be seen in listing 4. The response returned by Google's server is HTTP/1.0, which is not only apparent because of the version indication in the very first line, but rather because

a real HTTP/0.9 response would contain a single HTML ASCII text stream without any headers or deviant character encoding. However, both of these features can be seen in the response of the example. Nonetheless, the server is willing to handle the request.

```
$> telnet google.com 80
Trying 2a00:1450:4001:802::1003...
Connected to google.com.
Escape character is '^]'.

GET /

HTTP/1.0 200 OK
Date: Thu, 02 Oct 2014 20:55:08 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
[...]

<!doctype html>[...]</html>Connection closed by foreign host.
```

Listing 4: HTTP/0.9 request (with HTTP/1.0 response).

2.3.2 HTTP/1.0: Common Usage

The WWW grew very fast after its inception, and it was clear that HTTP/0.9 was too simple and could not keep up with the growing requirements. Therefore, a first draft specification with basic ideas on how HTTP/1.0 could work was published (Berners-Lee 1992a). Web developers produced a large amount of experimental HTTP clients and servers that implemented variations of this specification. Then the developers waited for other web developers to adopt their ideas.

In May 1994, the First International Conference on the World-Wide Web (WWW1) was held in Switzerland, and the World Wide Web Consortium (W3C) plus the HTTP working group were founded to guide the evolution of future web standards. The goals of the working group were to improve and formalize HTTP as an IETF Internet standard. The RFC 1945 published by the group with informational status documented the common usage of various HTTP/1.0 clients and servers (Berners-Lee, Fielding, and Nielsen 1996).

A HTTP/1.0 request starts with the same line but adds two additional methods besides `GET`—namely the `HEAD` (illustrated in listing 5) and `POST` methods—while explicitly stating that these can be extended at will. The request's path is now followed by the client's HTTP version string because a server would have to assume an HTTP/0.9 client if it would be omitted. There are also many new header fields a request and response can

```
$> telnet tools.ietf.org 80
Trying 2001:1890:123a::1:2a...
Connected to tools.ietf.org.
Escape character is '^]'.

HEAD / HTTP/1.0
User-Agent: Mozilla/2.0 (compatible)
Accept: */*

HTTP/1.0 200 OK
Date: Fri, 03 Oct 2014 10:24:58 GMT
Server: Apache/2.2.22 (Debian)
Last-Modified: Thu, 16 May 1996 20:40:03 GMT
Accept-Ranges: bytes
Content-Length: 137582
Vary: Accept-Encoding
Connection: close
Content-Type: text/plain; charset=UTF-8

Connection closed by foreign host.
```

Listing 5: HTTP/1.0 request and response.

carry along. The most important changes between the two versions can be summarized as follows:

- Multiple new request methods.
- Request path must include trailing version information.
- Request can contain multiple header fields terminated by CRLF.
- Response is prefixed with status information.
- Response can contain multiple header fields terminated by CRLF.
- Response headers and body must be separated by two CRLF combinations.
- Response body may contain arbitrary data, and is not limited to HTML.

Request (and response) headers are still limited to ASCII characters, and a connection is still required to be directly terminated after the response was sent to the client. This means that every cycle in the communication requires an exclusive TCP connection, which "imposes a significant performance penalty on HTTP/1.0" (Grigorik 2013a, 159).

2.3.3 HTTP/1.1: First Standard

While the HTTP working group was still working on the documentation of HTTP/1.0, development of a real Internet standard was initiated as well under the name HTTP next generation (HTTP-NG). The result of this development was RFC 2068, which was already

(partly) adopted by many vendors in client, server, and proxy software alike by the time it was released. This fast adoption of the pre-standards—"often referred to as HTTP/1.0+" (Gourley and Totty 2002, 17)—lead to various problems (Krishnamurthy, Mogul, and Kristol 1999, 660; Fielding et al. 1997).

The protocol was, thereafter, overhauled a few years later, which resulted in RFC 2616. The updated standard aimed at resolving ambiguities, and improving interoperability and performance (Krishnamurthy, Mogul, and Kristol 1999, 661–66; Grigorik 2013a, 159; Fielding et al. 1999). But the evolution of the WWW went on, accompanied by the advent of many new techniques to deliver and enrich web pages, like the generally known asynchronous JavaScript + XML (AJAX) technology. The IETF, thereafter, instituted the HTTPbis working group in 2007 with the goal to refine RFC 2616.

The objective of this working group was the incorporation of various updates, fixing editorial problems and clarifying conformance requirements, but without developing or extending the protocol any further (IESG Secretary 2007). The work took almost seven years and resulted in various new RFCs. The two core specifications that define the protocol since then are RFC 7230 and RFC 7231. All previous specifications—RFC 1945, RFC 2068, and RFC 2616—were deprecated and obsoleted with the release of the RFC 723x series, which further comprised widely used techniques extracted by the HTTPbis working group from other RFCs plus a few amendments.

- Conditional requests (RFC 7232, Fielding and Reschke 2014b).
- Range requests (RFC 7233, Fielding, Lafon, and Reschke 2014).
- Caching (RFC 7234, Fielding, Nottingham, and Reschke 2014).
- Authentication (RFC 7235, Fielding and Reschke 2014a).
- New HTTP request methods (RFC 7236, Reschke 2014a).
- 308 moved permanent status code (RFC 7538, Reschke 2015).

This results in an extensive list of features that are available for HTTP/1.1, which is beyond the scope of this thesis, as exemplified by the sheer amount of RFCs defining the protocol. One performance problem, which results from the vast feature set, is that many header fields exist that have to be sent along with each request and response. The contents of those fields largely remain stationary across all messages, as illustrated in listing 8 with a diff-comparison between both listing 6 and listing 7 requests for different resources on the same web page. This problem derives from the statelessness of HTTP: the counter-side does not know what headers were sent with another request, even if it was sent over the very same TCP connection. But there are also a few important features

that improved performance (Krishnamurthy, Mogul, and Kristol 1999):

- Range requests allow receivers to continue aborted downloads.
- Compression allows the usage of compressed coding to minimize payload size.
- Persistent connections allow reusing of existing TCP connections.
- Pipelining allows to send multiple HTTP requests over a single TCP connection.
- Chunked transfer-coding allows to start sending without prior knowledge of the final size.

Parallel connections are not part of the specifications but mentioned and often used by UAs to speed-up the process of downloading several resources at the same time. All of these parallel connections share processing power and throughput on the participant's computer. The complete loading process is faster because delays are split, and the resources finish transmission at approximately the same time. But parallel connections are not always faster, especially, if throughput and processing capabilities are limited. Human perception might still create the feeling of a faster loading page because some resources appear earlier on the screen even if the complete download has not finished yet and takes longer as if the resources were fetched in a more traditional, serial manner (Gourley and Totty 2002, 88–90).

Browsers have built-in limitations and will not establish more than six[20] parallel connections to the same domain[21]. A general advice for HTTP/1.x websites to optimize the load times of pages is, therefore, domain sharding, where multiple domains are used to reference resources within the same page. This way, browsers are tricked into establishing more connections by scattering resources over multiple domains. The effect of domain sharding is illustrated in equation (2.12), where t_{DNS} denotes the time to perform the necessary DNS look-up, x the domains in use, and y the total resource count within the page (Stefanov 2012, 31–35; Souders 2009, 161–69).

$$t = (x \times t_{\mathrm{DNS}}) + (x \times \mathrm{RTT}) + \frac{y \times \mathrm{RTT}}{x \times 6} \tag{2.12}$$

This equation only applies to fresh connections, subsequent requests that are executed over an existing connection are reduced to simply $y \times \mathrm{RTT} / x \times 6$; presumed that resources are actually downloaded and not properly cached. Domain sharding, therefore, has numerous drawbacks: a DNS look-up is necessary for each domain, a new TCP connection has to be

20. http://www.browserscope.org/?category=network

21. There are no IP address restrictions, and often simple `CNAME` DNS entries are used (Souders 2009, 168; Grigorik 2013a, 199).

```
Host: www.ssllabs.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:32.0) Gecko/20100101
 ↪  Firefox/32.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: JSESSIONID=4 ... e); __utmt=1
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
```

Listing 6: First HTTP/1.1 request.

```
Host: www.ssllabs.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:32.0) Gecko/20100101
 ↪  Firefox/32.0
Accept: text/css,*/*;q=0.1
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://www.ssllabs.com/
Cookie: JSESSIONID=4 ... e); __utmt=1
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
```

Listing 7: Second HTTP/1.1 request to the same host as in listing 6.

```
--- http-11-request-1.txt        Fri Oct 24 17:04:07 2014
+++ http-11-request-2.txt        Fri Oct 24 17:04:41 2014
@@ -1,8 +1,9 @@
 Host: www.ssllabs.com
 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:32.0) Gecko/20100101
  ↪  Firefox/32.0
-Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
+Accept: text/css,*/*;q=0.1
 Accept-Language: en-US,en;q=0.5
 Accept-Encoding: gzip, deflate
+Referer: https://www.ssllabs.com/
 Cookie: JSESSIONID=4 ... e); __utmt=1
 Connection: keep-alive
 Pragma: no-cache
```

Listing 8: Unified diff between listing 6 and listing 7.

established to each host including slow-start—the sizes of the resources are not considered in equation (2.12)—and last but not least a new TLS tunnel has to be negotiated for each connection if encrypted communication is in use. This effect is further worsened if persistent connections are not supported (Grigorik 2013a, 198–200).

Persistent connections were perhaps the most important addition to HTTP/1.1. In fact, the semantics of HTTP were changed to always reuse established connections per default because of the importance of this feature (Gourley and Totty 2002, 98). The `Connection: Keep-Alive` HTTP header line was a non-standardized extension that worked differently but allowed supporting software to create something similar in the earlier protocol's revision (Thomson, Loreto, and Wilkins 2012; Gourley and Totty 2002, 91–97). A lot of software is still including this header line even if HTTP/1.1 is in use, as can be seen in listing 6 and 7.

Pipelining (or pipelined connections) is another performance optimization, which was introduced with HTTP/1.1 and can have a significant impact on load times. Apple, for instance, was able to reach a 300 % speed-up in their iTunes software by using pipelined, persistent connections on slower networks (Graessley 2012). This technique enables a client to send multiple HTTP requests over the same persistent connection before any response has arrived. In consequence, this allows parallel arrival and processing of the resources, while effectively eliminating request propagation latencies.

But there are several issues with pipelining. First, and most importantly, it must be supported by both—the client and the server. Most web servers support pipelining, but no major browser has it activated.[22,23,24] Secondly, multiplexing is not possible and the server has to send the responses back in the same order as the requests arrived. This means in effect that pipelined transactions work according to the first-in-first-out (FIFO) principle and add another layer that is suspect to HOL blocking (see section 2.2.5 as well).

This is most apparent if HTTP streaming is in use because it blocks by definition. The connection needs to stay open in order to allow the server to push data to the client without the data being requested by the client. Attempts have been made to fix this by extending HTTP/1.1 to support hints that the server can send to the client—indicating which resources are safe to be pipelined—but they were abandoned in favor of new multiplexing technologies like SPDY and HTTP/2; see figure 5 for a visual comparison of the different HTTP connections (Grigorik 2013a, 192–96; Gourley and Totty 2002, 99–100; Stefanov 2012, 34–35; Kurose and Ross 2013, 215–19; Nottingham 2011a; Fielding and

22. http://www.chromium.org/developers/design-documents/network-stack/http-pipelining
23. http://kb.mozillazine.org/Network.http.pipelining
24. https://bugs.webkit.org/show_bug.cgi?id=64169

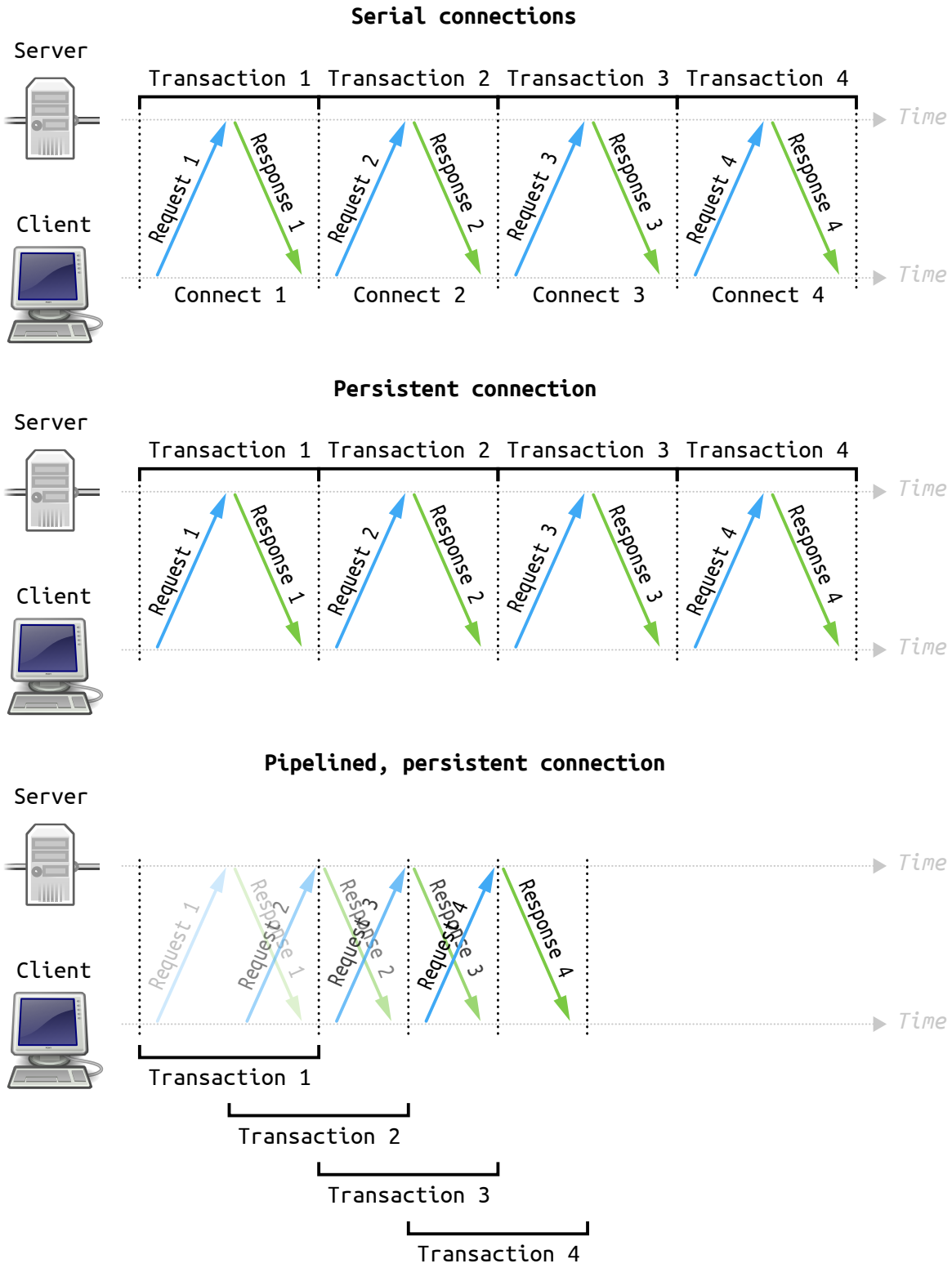


Figure 5: Comparison of the various HTTP connections (Fussenegger 2015a; Gourley and Totty 2002, 100).

Reschke 2014c, 2014d, 2014b; Fielding, Lafon, and Reschke 2014; Fielding, Nottingham, and Reschke 2014; Fielding and Reschke 2014a; Reschke 2014a, 2014b).

2.3.4 HTTP/2: Next Generation

In 2009, Google started their "Let's make the web faster" campaign because their experiments showed that performance is one of the most important key factors to satisfy users (Hoelzle and Coughran 2009; Brutlag 2009b, 2009a). In the same year, Google announced that one of their developer teams started working on an experimental protocol called SPDY as part of the campaign (Belshe and Peon 2009). A white paper followed this announcement with an explanation why a new protocol was necessary as well as a formalization of the goals for it (Google, Inc. 2009).

The main goals were a reduction of PLTs by at least 50 %, and avoidance of the need for changes of websites or in existing network infrastructure (Grigorik 2013a, 208). It was already explained in section 1 that encryption was necessary because Google would not have been able to deploy the protocol otherwise, but improving security became another goal of SPDY at a later point. Other industry leaders, like Mozilla and Amazon.com Incorporation (Amazon), joined the SPDY development, and the IETF instructed the HTTPbis working group at the beginning of 2012 to create a new HTTP standard (Winkler 2014, 1; Leiba and Morgan 2012).

The group used SPDY 2 as a working basis since the goals were very similar, but they developed several changes and improvements until the final protocol was released in May 2015 as RFC 7540. The new revision promises shorter latencies, higher throughput, and more efficient network resource usage. Further improvements are promised through the inclusion of multiplexing and push technologies as well as compression of headers (Grigorik 2013a, 207). The latter is performed with the exclusively developed HPACK algorithm and was defined in a dedicated specification, which was released along with HTTP/2 as RFC 7541. SPDY also features header compression, but more on that later.

SPDY and HTTP/2 do not alter the semantics of HTTP. This does not mean that the protocols are backward compatible, adoption of client and server software is necessary to support them. However, web applications and network intermediaries, which are not directly dealing with constructing or parsing HTTP messages, do not require any changes. This is achieved through modification of the way how data is formatted (framed) and transported between the peers, and should ensure a fast adoption of the new protocols for obvious reasons. In fact, this approach already worked out for the experimental SPDY protocol after only a few years with decent support in client and server software.

The major version increment of the new revision is due to the new binary framing layer. The decision to migrate from a textual to a binary protocol was based on the facts that the former is more complicated to parse, requires more space, and is more error-prone, as empirical knowledge with HTTP revealed. Additionally, text requires a character encoding and various workarounds to deal with whitespace, capitalization, line endings, or blank lines. This is exemplified by the various notices and warnings in regards to parsing and interpreting of HTTP messages in RFC 7230 (Fielding and Reschke 2014c, 19–34; Grigorik 2013a, 207–11).

Both protocols only use a single TCP connection to exchange data between peers in order to minimize the negative effects of establishing multiple connections and to introduce multiplexing. In HTTP/2 each such stream contains one or more messages which are sets of aforementioned frames. They are the smallest unit and indicate the type of data (see section 6. of Belshe and Peon 2015, 31–49) they encapsulate with some additional data in their header, which at minimum needs to include the stream identifier. This requirement is necessary because such a HTTP/2 TCP connection can contain an arbitrary amount of bidirectional streams, messages, and frames that can be interleaved and reassembled. The latter is only possible via these identifiers.

This new binary framing layer of HTTP/2 with the multiplexed streams over a single TCP connection is the most important change of the protocol in the context of performance. It removes the HOL blocking of HTTP entirely, but more improvements were incorporated. HTTP header redundancy was already mentioned and illustrated in section 2.3.3, and the SPDY authors addressed this issue by utilizing the deflate algorithm to compress headers. The representation of redundant header fields was very effectively reduced, but researchers found that it leads to information leakage, which allows attackers to perform a so called session hijacking attack, known as Compression Ratio Info-leak Made Easy (CRIME) (Peon and Ruellan 2015, 3; Rizzo and Duong 2012).

A new algorithm called HPACK was developed for HTTP/2 to counteract the CRIME attack and defined in a dedicated specification (RFC 7541, Peon and Ruellan 2015). The new approach uses two mechanisms to reduce the redundancy and size of HTTP headers: for one thing, it uses an (optional) static Huffman encoding to compress string literals, which is considered to mitigate the CRIME attack (Peon and Ruellan 2015, 21; Tan and Nahata 2013; Huffman 1952), and ,for another thing, it utilizes indexed lists of previously sent name-value header field pairs, which allows index references and, in further consequence, the omission of the entire header name and value.

A dynamic and static table of previously encountered header fields are maintained

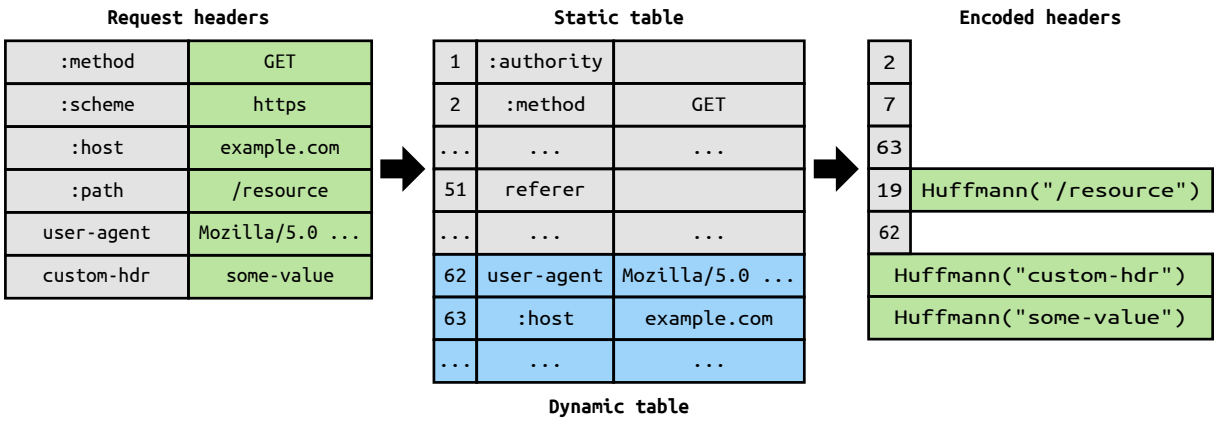


Figure 6: HTTP/2 header compression HPACK (Fussenegger 2015b; Grigorik 2015).

and updated on both the client and server side. Both tables share their address space, and while the static table is used for pre-defined standard headers (see appendix A. Peon and Ruellan 2015, 25–26) the dynamic table is used for others. An index is assigned to each header that is appended in the dynamic table, which can be used for the previously mentioned index references in later frames of the same connection. A simplified illustration of this approach can be found in figure 6. The combination of the binary framing layer with fixed-length fields and HPACK for compression reduces the overhead of the protocol to a minimum for both compact representation and simple parsing (Belshe and Peon 2015, 12–14; Peon and Ruellan 2015; Grigorik 2013a, 211–17, 222–24, incorporating updates from the online version).

HTTP/2 introduces its own flow control mechanism because TCP's flow control (see section 2.2.2) is not granular enough, and does not provide the necessary application programming interfaces (APIs) for the regulation of individual streams. The specification does not mandate a particular algorithm for this feature and provides only the format and semantics for an implementation. All problems that were previously presented in the context of TCP's flow control need to be taken into account for HTTP/2's flow control as well. This means that HOL blocking may occur within a connection if it is not handled properly, and deadlocks may occur if critical frames are not processed (Belshe and Peon 2015, 22–24; Grigorik 2013a, 218–19).

Stream dependencies, priorities, and weights allow a peer to suggest an endpoint how it would prefer resource allocation for the streams of a connection. A prioritization tree can be built from this information, which can be used to optimize the delivery of data. For instance, HTML documents and Cascading Style Sheets (CSS) are very important for rendering a web page because the Document Object Model (DOM) and CSS Object

Model (CSSOM) need to be built from them in order to render the page. This means in effect that a client could prioritize them over other resources and make everything else dependent on their successful delivery. Efficient, automated prioritization was and is subject of ongoing research (Belshe and Peon 2015, 24–28; Grigorik 2013a, 215–17).

Server push is another feature that can be utilized to improve the performance of web applications, but also for many other purposes and completely new applications. It breaks out of the request/response semantic of HTTP, and permits one-to-many workflows, which previously were only possible through various hacks. A server can push multiple resources for a single request of a client, but how many and what kind of resources is controlled by the client. This is an important security feature, and allows conditional processing based on the client's cache, for instance, as it can decline any stream. Resource inlining via data URIs would be an example of forced push without any caching opportunity, which is elegantly solvable with server push (Belshe and Peon 2015, 60–63; Grigorik 2013a, 219–22).

In order to use HTTP/2 for communication an upgrade process is necessary because peers have to assume an HTTP/1 counterpart. A complete section of the specification is dedicated to all three possible cases of upgrades (Belshe and Peon 2015, 8–10). The most common case is going to be the upgrade during the TLS handshake via the Application Layer Protocol Negotiation (ALPN) extension (defined in RFC 7301 Friedl et al. 2014) because major browsers will not support unencrypted connections with HTTP/2, as was already explained in section 1.

The TLS handshake always has to happen before any application data may be transmitted. This means that the upgrade will not add additional round-trips or any other unwanted side-effects that affect performance. The exact upgrade process, especially of the other two cases, is not important at this point as it is an implementation detail that is covered by the client and server software. But it is important to note that HTTP/2 connections without TLS may break even with the prior knowledge that the endpoint supports the new protocol simply because intermediaries always change, and one or more might not be able to handle the new protocol in subsequent connections. TLS with ALPN is the safe choice for best utilization of the new features (Grigorik 2013a, 224–26).

All in all HTTP/2 looks promising, but future adoption has to show if the protocol will be a success or not. Research has to provide algorithms that improve stream handling, especially prioritization and flow control. Further discussion of SPDY is omitted at this point in favor of HTTP/2 simply because it was an experimental protocol from the very beginning, and Google already announced that they will drop SPDY and support only

HTTP/2 in the future (Bentzel and Béky 2015). It is sufficient to note that the latest SPDY revision is very similar to HTTP/2, and that it was always used by Google to test new ideas for its successor.

2.4 Cryptography Primer

Although this thesis is not about cryptography per se, a few very basic concepts need to be explained in order to understand how TLS and the former SSL achieve their goals of confidentiality, integrity, and authenticity. The goal is always for two parties—A and B that are called Alice and Bob (bound by tradition Rivest, Shamir, and Adleman 1978, 121) —to communicate securely over an insecure channel. Various cryptographic primitives are necessary to achieve this goal, and to assure the aforementioned three properties. There are a few terms that are commonly used in this context, which can be clarified without further in-depth explanation (Paar and Pelzl 2010, 5):

- Clear- or plaintext x is the original data.
- Cipher is the algorithm in use for performing encryption or decryption.
- Ciphertext y is the encrypted data.
- A cryptographic key k might be involved where indices indicate the type.
- Key space $\mathcal{K}$ is the set of all possible keys from which a key k can be derived from. It is possibly the most important aspect of many cryptographic systems and usually determines its strength.

2.4.1 Random Number Generators

Random number generators (RNGs) play a very important supporting role and are, of course, no cipher. “In cryptography, all security depends on the quality of random number generation.” (Ristić 2014, 14) The randomness of these generators is, therefore, absolutely crucial for all ciphers which rely on an initialization vector (IV). There are three types of RNGs (according to Paar and Pelzl 2010, 34–36):

- True random number generators (TRNGs) that generate real randomness and are usually implemented as physical devices. For instance, a microphone placed nearby a heavily trafficked street or computer mouse moved by a human could provide such randomness.
- Pseudo random number generators (PRNGs) or deterministic random bit generators (DRBGs) that generate statistical randomness based on a seed value and its extended

type via an algorithm. Examples for this kind of RNGs include the special device `/dev/urandom` on Linux OSs.

- Cryptographically secure PRNGs (CSPRNGs) that have the additional requirement of producing unpredictable randomness. A good example would be the Yarrow algorithm used in some Unix-like OSs for `/dev/random`.

Computers in general are not good at producing randomness since their inputs are limited to certain states, which are often predictable. This situation is worsened if virtualized environments are in use since these systems have no access to real physical hardware (Everspaugh et al. 2014). Some CPUs feature built-in TRNGs—like the `RdRand` (Bull Mountain) instruction on Intel Ivy Bridge processors—but many projects do not trust these sources because of backdoor concerns.[25,26,27,28]

2.4.2 Hash Functions

Hash functions are very important primitives in cryptography, and, probably, the best known type among web developers and administrators. They exist in various flavors and are classified as either keyless or keyed. The input can be of variable length, while their output is always a short, fixed-length bit-string: the so called digest, hash value, or simply hash. Keyed hash functions, obviously, require a key, which is of a fixed-length and leads to further differentiation because a key may be private or public. However, all incarnations are generally working very similar by utilizing a construction function that repeatedly calls a compression function (Al-Kuwari, Davenport, and Bradford 2010, 16).

The usage of hash functions is manifold in cryptography, and they are necessary for digital signatures, message authentication codes (MACs), or to securely store passwords. Another typical usage example is file comparison or integrity checks with a digest known in advance. Hash functions are usually extremely fast and produce a small output even for hundreds of megabytes. Modern parallelizable hash functions promise even faster execution times on todays multi-processor architectures. In order for such functions to be useful in the context of cryptography, three requirements have to be met (according to Paar and Pelzl 2010, 296; Ristić 2014, 10):

- Preimage resistance (or one-wayness): given a digest, it must be computationally infeasible to find or construct data that produces the same digest.

25. https://plus.google.com/+TheodoreTso/posts/SDcoemc9V3J
26. http://www.theregister.co.uk/2013/12/09/freebsd_abandoning_hardware_randomness/
27. http://www.theregister.co.uk/2013/09/10/torvalds_on_rrrand_nsa_gchq/
28. https://www.freebsd.org/news/status/report-2013-09-devsummit.html#Security

- Second preimage resistance (or weak collision resistance): given a digest and the data it was produced from, it must be computationally infeasible to find different data which produces the same digest.
- Collision resistance (or strong collision resistance): it must be computationally infeasible to find two different data that produce the same digest.

Examples for well known hash functions are MD (family), SHA (family), RIPEMD (family), and Whirlpool. Some of them are cryptographically broken and should not be used anymore for security sensitive applications.

2.4.3 Symmetric Cryptography

Symmetric cryptography (also known as private-key, symmetric-key, secret-key, or single-key cryptography) ciphers have two defining properties. First—and most importantly—one single and identical secret key is used by Bob to encrypt and Alice to decrypt the document and vice versa; see figure 7. The second property is that the decrypt and encrypt functions are very similar or even identical. There are, furthermore, two modes of operation: stream and block ciphers. The first mode, as the name already suggests, takes a stream of data, often byte-wise (Ristić 2014, 7), and encrypts/decrypts continuously. A well known example for a symmetric stream cipher would be the Rivest Cipher 4 (RC4).

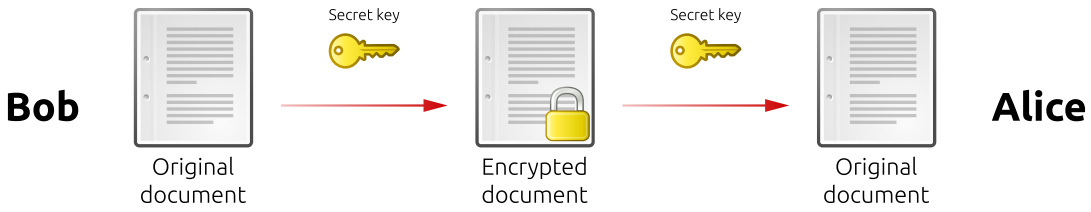


Figure 7: Illustration of symmetric cryptography (Fussenegger 2014c).

A block cipher, on the other hand, takes a fixed amount of data (often 128 bit (8)) and encrypts/decrypts each block successively. Data which does not fill such a block has to be padded in order to fill it. Block ciphers can build the basis for many other cryptographic primitives, like the already explained hash functions. Well known examples for block ciphers include the Advanced Encryption Standard (AES), Blowfish, Camellia, Data Encryption Standard (DES), International Data Encryption Algorithm (IDEA), and Rivest Cipher 5 (RC5). Both types of symmetric algorithms have the positive feature of being very fast compared to other cryptographic algorithms; even for large amounts of data. However, while these algorithms are fast and secure they also have various disadvantages. The secret key, obviously, has to be distributed between Alice and Bob,

which has to happen over a secure channel because otherwise a malicious party could tap the key and decrypt the data as well.

Additionally, a single key will most certainly not suffice for a large group of u users, otherwise everyone could decrypt each message. This means that each user pair needs its own key pair. As a result, each user would have to store $u - 1$ keys as visualized in figure 8. With this result, the total keys in circulation can be deduced by multiplication with the number of users participating in the system. From this it follows that even a small group of ten users would already require forty-five key pairs, as can be seen in the following equation:

$$k_{\text{total}} = \frac{u \times (u - 1)}{2} \tag{2.13}$$

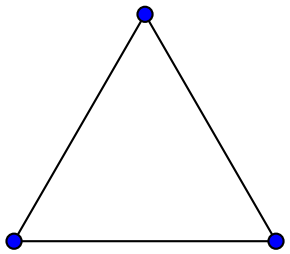


Figure 8: Complete graph visualizing symmetric cryptography key distribution for three users (Benbennick 2006).

Further, a group of two thousand users would already require approximately two million key pairs. There are various approaches to tackle this, like Kerberos, which is a sophisticated protocol that is indeed widespread. However, all of those approaches have various problems, like communication requirements, secure channel during initialization, single point of failure, and no perfect forward secrecy (PFS) (see section 2.4.8; Paar and Pelzl 2010, 150–51, 336–41). The last "problem" is that messages are repudiable since all involved users have the same key.

Alice could simply encrypt a message and claim that it was sent by Bob, and he would not be able to prove the opposite. This could also be a feature—for instance in Off-the-Record (OTR) messaging (Borisov, Goldberg, and Brewer 2004)—but it is a serious problem in e-banking or e-commerce systems. The transfer of funds or the ordering of a product has to be non-repudiable, which is not possible with symmetric cryptography (Paar and Pelzl 2010, 4–6; Ristić 2014, 4–9; Thomas 2000, 22–24).

2.4.4 Message Authentication Codes

Message authentication codes (MACs, also known as cryptographic checksum or keyed hash function) can provide integrity and authenticity assurances through the usage of symmetric cryptography. This is necessary because Alice needs the ability to validate that the message she received is the unaltered original sent by Bob. It was already clarified in section 2.2.1 that any party with access to the communication channel is also able to intercept and manipulate data that is being sent. Consequently, every such party has the ability to alter messages in the channel. MACs help to solve this problem by attaching a cryptographically secure authentication tag a to the message.

The a is generated by the sender Bob via a key k, enabling Alice to verify the integrity and authenticity of the message. This way, she will know to discard the message if the tag does not match. Of course, the k has to be transmitted to Alice in advance over a secure communication channel. Otherwise, Alice is not able to decrypt the attached a. The summed up properties of this supporting cryptographic primitive are, therefore (according to Paar and Pelzl 2010, 321):

- Checksum, by appending an authentication tag.
- Symmetric keys, shared by all participants.
- Message integrity, making transit manipulation detectable by receiver.
- Authentication assurance of valid origins.
- Repudiable, as the receiver cannot verify that the sender was a particular person.
- Plus all properties from the hash functions:
  - Extremely fast.
  - Arbitrary input size.
  - Fixed-length output.

Symmetric block ciphers are historically the basis for MACs, but other modes of operation are possible. The name hash message authentication code (HMAC) is used if a cryptographically strong hash function forms the basis. Some of the fastest implementations are based on universal hashing, like VMAC (Krovetz and Dai 2007). Another specialty of MACs is that they can operate with multiple different cryptographic primitives, which is actually used in TLS, where the data is split in half and the computed digests are XORed to compose the actual MAC (Paar and Pelzl 2010, 319–28; Ristić 2014, 10–11).

It is important to note that MACs are not encrypting or decrypting anything, they only provide integrity and authenticity. Confidentiality, or the encryption of the message,

happens in a separate step. When this step is performed is up to the implementation. However, it may be combined in a single cipher that is known as authenticated encryption with associated data (AEAD). Three such approaches are currently in use in different protocols. The following list recapitulates them in a mathematic notation, where f_{mac} denotes the MAC generation function, and f_{sym} the symmetric encryption function (according to Krawczyk 2001, 311):

- Authenticate-then-encrypt (AtE) or MAC-then-encrypt (MtE)
  1. $a = f_{mac}(x)$
  2. $y = f_{sym}(x, a)$
  3. y is transmitted.
- Encrypt-then-authenticate (EtA) or encrypt-then-MAC (EtM)
  1. $y = f_{sym}(x)$
  2. $a = f_{mac}(y)$
  3. y and a are transmitted.
- Encrypt-and-authenticate (E&A) or encrypt-and-MAC (E&M)
  1. $y = f_{sym}(x)$
  2. $a = f_{mac}(x)$
  3. y and a are transmitted.

EtA/EtM is currently considered to be the best approach and was standardized for TLS in RFC 7366, but it is not implemented in any widely adopted software as of yet. The default approach for TLS is, therefore, still AtE/MtE with Galois/Counter Mode (GCM) from RFC 5288, being the best available AEAD cipher which is used in practice (Ristić 2014, 44–45; Gutmann 2014; Salowey, Choudhury, and McGrew 2008).

2.4.5 Asymmetric Cryptography

Asymmetric cryptography (also known as public-key cryptography) utilizes a key pair per user, where one key is public and the other is private. The principle behind the system is very easy to describe: a night safe of a bank is accessible to everyone who can access the foyer of the bank (public key), but the contents of the night safe are only accessible to the persons who have the key to it (private key). Data encrypted with Alice's public key k_{public} is only decryptable with Alice's private key $k_{private}$, whereas data encrypted with $k_{private}$ is decryptable by anyone in possession of k_{public}; see figure 9.

The greatest advantage of this approach is that k_{public} can be transmitted over an

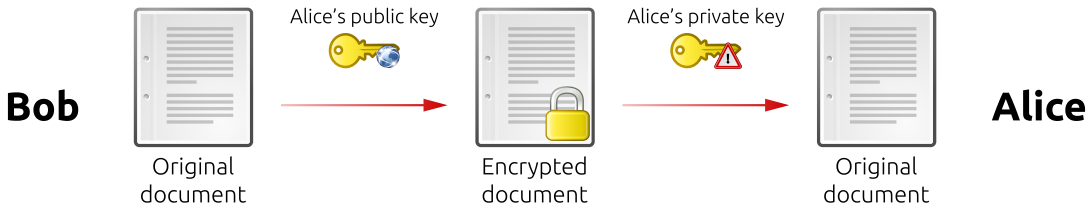


Figure 9: Illustration of asymmetric cryptography (Fussenegger 2014d).

insecure communication channel because any malicious party receiving k_{public} can only encrypt messages for the owner of $k_{private}$. A side-effect of asymmetric encryption is the fact that $k_{private}$ makes sure that the message is non-repudiable, a property that none of the aforementioned ciphers was able to provide. Bob can sign the data encrypted with Alice's public key with his private key, and Alice can use that signature after decrypting the data to validate his digital signature (Paar and Pelzl 2010, 4–6, 152–54; Ristić 2014, 12–13).

2.4.6 Digital Signatures

Digital signatures extend upon the MAC idea by utilizing asymmetric instead of symmetric cryptography, and in turn provide non-repudiability for certificates, documents (contracts, emails, etc.), software updates, and much more. There are a few different signature schemes: the best known and most widely used form is the Rivest Shamir Adleman Cryptosystem (RSA), and a newer kind is the Elliptic Curve Digital Signature Algorithm (ECDSA). In fact, only these two are supported in TLS (Paar and Pelzl 2010, 259–64; Ristić 2014, 13–14). The following properties are provided by digital signatures (according to Paar and Pelzl 2010, 264):

- Identity authentication, by establishment and verification of participants.
- Access control, based on a provided signature.
- Availability, by assurance that a system is reliably available.
- Auditing, which provides evidence about events (via logs for instance).
- Physical security, by protection against tampering.
- Anonymity, by protection against identity theft or misuse.

How digital signatures can be validated within the ecosystem of TLS and the Internet is discussed in more detail in section 2.6.4, as this yields multiple challenges and is a highly controversial topic.

2.4.7 Choosing Key Sizes

It was already mentioned that the strength of many ciphers is determined by the key space $\mathcal{K}$. Choosing an appropriate size is probably the easiest decision during the deployment of cryptographic services. But simply choosing the greatest available $\mathcal{K}$, while tempting—since it seems to promise highest security—might not be the wisest decision. The resulting huge keys come with additional computational complexity during generation, sending, and validation that must not be disregarded.

Various standardization bodies[29] try to address this problem by giving recommendations based on multiple factors, which are out of scope for this thesis to discuss. See (Dent 2010) for a high level introduction to the topic or (BSI 2015), (ANSSI 2014), and (Smart 2013) for non-US publications. The problem is further exacerbated by the fact that some ciphers are insecure and should not be used at all, and that key sizes are not easily transferable from one type of cipher to another.

RFC 7525 was especially created to address these issues for the TLS ecosystem and gives recommendations for choosing proper ciphers and key sizes. It concludes that 128 bit of security (2^{128} operations) are considered to be adequate for generic web services. Table 3 gives a compact overview of key sizes used and supported in most TLS implementations. The low security row in the table should not be used, for reasons explained in detail in the RFC, especially since Mozilla already begun phasing out weak keys in its Network Security Services (NSS) library (which affects their Firefox and Google's Chrome browser; Wilson 2014, 2015; Sheffer, Holz, and Saint-Andre 2015).

Security	Symmetric	Asymmetric	Elliptic curve (EC)	Hash
Low	96	1,024	192	192
Default	128	2,048	256	256
High	256	4,096	512	512

Table 3: Commonly used key sizes within the TLS ecosystem (Ristić 2014, 17–18, 236–37; Sheffer, Holz, and Saint-Andre 2015).

2.4.8 Perfect Forward Secrecy

Perfect forward secrecy (PFS) is an important term in cryptography that describes an attribute of key exchange algorithms. Traditionally, static RSA and Diffie–Hellman key exchange (DH) are used for transportation of the intermediate keys that will be used to

29. Interactive overview of various recommendations: http://www.keylength.com/

encrypt the data in flight. This means in effect that the security of all connections depends on the server's private key. A compromise of that key consequently allows a third party to decrypt any communication that was ever performed with this key, not to mention the obvious other implications that such a compromise would lead to.

Alternative key exchange algorithms generate a new key for each session without using deterministic algorithms. Thus, a compromise of the server's private key in the future will not allow the adversary to decrypt previous communication. Such a system is commonly referred to as PFS, and currently two such methods are available in TLS: The older is known as ephemeral Diffie–Hellman key exchange (DHE or EDH), which was first proposed to the public by (Diffie and Hellman 1976) and further refined in (Diffie, Hellman, and Merkle 1980). The system is based on the discrete logarithm problem and requires the generation of strong domain parameters (also known as Diffie–Hellman parameters) that are basically just prime numbers.[30]

The session's dynamic encryption key is derived from those parameters, and the ephemerality is secured by randomly chosen integers from both peers that are necessary for exponentiation operations (Paar and Pelzl 2010, 206–8; Bernat 2011). The size of the domain parameters directly translates to the security of each connection. Most servers use 1,024 bit, which translates to about 80 bit of security. This amount of security might suffice since the keys are only in use for a very short time, but generation of a key with the same amount of bits as the server's RSA certificate is generally recommended (Ristić 2014, 244–45).

The exponentiation operations of the DHE method result in bad performance compared to static methods. (Law et al. 1998) were the first to propose a new method that is based on algebraic structures of elliptic curves over finite fields, which achieves the same security with smaller keys, and provides a performance that is comparable to RSA. This method is known as Elliptic Curve Diffie–Hellman key exchange (ECDHE), and works very similar to DHE. The main difference is that there are less domain parameters to exchange between both peers, and that all operations are based on predefined named curves. Those curves were first defined in RFC 4492, with a small extension in RFC 5246, and addition of new curves in RFC 7027 (Blake-Wilson et al. 2006; Dierks and Rescorla 2008, 78; Merkle and Lochter 2013).

The creation of efficient and secure curves for key exchange is very difficult, and some of the existing curves are suspect to backdoors from the NSA and the National Institute of

30. They can be generated with OpenSSL and its `openssl dhparam` command (see https://www.openssl.org/docs/apps/dhparam.html).

Standards and Technology (NIST) because they are manipulable with special seed values (Bernstein and Lange 2013). Various parties are, therefore, urging to move to new curves defined by independent researchers instead of relying on the NIST standardization body. One such new curve would be (Bernstein 2006), which was already drafted for inclusion in TLS (Josefsson and Pégourié-Gonnard 2014).

However, PFS is a very important property of modern secure communication, and it is expected that the upcoming TLS 1.3 standard will only support key exchange algorithms that provide it.

2.4.9 Post-Quantum

Mentioning key sizes for cryptographic systems without bringing up quantum computing is not possible any more because of the renowned factorization algorithm from (Shor 1994). This quantum algorithm is able to find the prime factors of an integer in polynomial time. Thus, it is able to break various cipher schemes, like RSA and elliptic curve cryptography (ECC), in a very short time since they are based on the assumption that it is computationally infeasible to factor such large numbers. This circumstance has lead to a completely new field of study called post-quantum cryptography (Bernstein, Dahmen, and Buchmann 2009).

On the one hand, this field consists of researchers trying to invent new ciphers that can withhold quantum computers, and, on the other hand, there are those researchers who try to create such quantum computers. This ongoing research sparked various cipher schemes that are considered to be quantum safe. But, most importantly in the context of this thesis, they also promise increased performance compared to current ciphers (Bos et al. 2015). However, as of now no such cipher was standardized or included in the TLS protocol.

2.5 Transport Layer Security

Transport Layer Security (TLS) is the de facto standard for securing communication in the Internet. The protocol is a wrapper around other application layer protocols and is accordingly situated in the same layer of the IPS—in strong contrast to what the actual protocol's name might suggest.[31,32] TLS was specifically designed for transport protocols

31. The name often yields confusion, and authors are tempted to place TLS and SSL within the transport layer of the IPS, as seen in (Oppliger 2009, 67, Figure 3.1).

32. Some authors argue that TLS and SSL add an additional layer to the IPS stack, as seen in (Thomas 2000, 8).

that provide reliable data delivery, like TCP. But it was also adapted for unreliable datagram protocols like UDP under the name Datagram Transport Layer Security (DTLS). However, all TLS revisions have the same goals (according to Ristić 2014, 2; Dierks and Allen 1999, 4; Dierks and Rescorla 2006, 5; 2008, 6):

- Cryptographic security, the major target.
- Interoperability between independent protocols and software.
- Extensibility by being a framework that does not depend on specific implementations.
- Relative efficiency by attaining all goals at tolerable performance costs.

TLS offers security through cryptography, which is "the science and art of secure communication" (Ristić 2014, 4) and is almost as old as mankind itself.[33] It follows the key principles of Information Security (InfoSec) by honoring confidentiality, integrity, and availability (CIA triad)[34]. If TLS is correctly configured, this means in effect that the cryptographic part keeps information secure and protects against eavesdropping (confidentiality), ensures safe transport and prevents alterations (integrity), and is always active if in use (availability). The triad is almost always extended with authenticity in the context of TLS because of the X.509 dependency.

However, in theory it would be possible to use TLS without any form of authentication by using anonymous cipher suites (see section 2.5.1), pre-shared keys (RFC 4279, Badra et al. 2005), or Kerberos cipher suites (RFC 2712, Medvinsky and Hur 1999). But—as hinted—this is almost never the case in practice, and the cryptography part of TLS is extended by proving identities of one or both participants and, in turn, protects against forgery and masquerading. But building trust is a complicated thing and the most criticized part of the whole TLS ecosystem; further details can be found in section 2.6.4.

It is important at this stage to point out that TLS will only provide point-to-point encryption (P2PE) and not real end-to-end encryption (E2EE). Meaning that only the transport of the message is secured between the client and the server that communicate. True multi-node E2EE would have to be implemented in the wrapped application protocol; the Internet Protocol Security (IPSec) was standardized by the IETF for this purpose (Meyer 2014, 3). TLS uses asymmetric cryptography to achieve the goal of authenticated P2PE, and to exchange a key for symmetric encryption of subsequent message exchanges between the client and the server.

All revisions of the protocol are multi-layered and consist of several higher-level subprotocols that must be encapsulated in the TLS Record Protocol. Two phases are necessary

33. https://en.wikipedia.org/wiki/History_of_cryptography
34. Not to be confused with the Central Intelligence Agency (CIA).

in order to securely communicate data between the participants. The negotiation phase (handshake, explained in more detail in the next section) comes first in order to agree upon the security parameters, cipher suites, validate identities (server only or both sides), and a pre-master secret by exchanging keys or by choosing arbitrary parameters (depends on the chosen cipher). It follows the application phase in which real data can be exchanged securely (Ristić 2014, 25–32; Meyer 2014, 3–4; Grigorik 2013a, 47–52; Oppliger 2009, 75–6; Thomas 2000, 67; Dierks and Allen 1999; Dierks and Rescorla 2006, 2008).

The following terms are important to distinguish and are used throughout the rest of this thesis in the context of TLS (based on Meyer 2014, 5):

- Connection is a communication channel between two peers without security.
- Session is a connection with agreed upon cryptographic parameters for security.
- Channel is a unique session with a particular set of keys and active security.

2.5.1 Cipher Suites

Cipher suites are a standardized set of cryptographic algorithms that are uniquely identifiable by two bytes in TLS, and commonly referred to via their constant's name. Such a suite always consists of four algorithms that are used for key exchange, authentication, encryption, and hashing. An example for such a suite would be one of the current recommended cipher suites: `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384` (byte string `0xC0,0x30`). The prefix `TLS` is always present in a TLS cipher suite and indicates the associated standard, `ECDHE` describes the key exchange algorithm, `RSA` the authentication algorithm, `AES_256_GCM` the encryption algorithm, and `SHA384` the hash function. Suites are used to enable easy addition and deprecation of used algorithms in the TLS protocol, and to avoid hard-coded cryptographic primitives within the standard.

2.5.2 Negotiation Phase

The TLS negotiation phase must precede every secured connection attempt in order to create an encrypted channel. However, the TCP handshake must always initially be completed to ensure that a reliable connection between both network participants exists before any TLS specific data may be transmitted. Multiple round-trips between both peers are, therefore, necessary, which imposes a significant overhead compared to an unencrypted channel. Some attempts that try to lower this overhead were already mentioned earlier, namely TFO and QUIC. Of course, the connection setup might differ with other protocols such as DTLS, but it generally applies to the usual HTTP Secure (HTTPS) traffic.

Further, the TLS Handshake Protocol is used during the negotiation phase to optionally authenticate one or both peers, to securely generate a shared secret for encryption of application data, and to ensure that the channel is secured against any third parties (Dierks and Rescorla 2008, 3–4). Figure 10 illustrates the full TLS handshake for a completely new connection, and extends figure 2 with the same timing values from section 2.1.1. The following list summarizes the messages that are usually exchanged between both peers during the negotiation phase.

1. The client can send the `ClientHello` message after the TCP connection is established, which starts the negotiation phase. Includes supported protocol versions, a RNG value, a session identifier (always empty during initial negotiation phase), and lists of supported ciphers and compression methods. The client may also request extensions in this message since RFC 3546, which is part of the protocol since TLS 1.2 (Blake-Wilson et al. 2003; Dierks and Rescorla 2008, 44–46).
2. The server answers with the `ServerHello` message, which contains the chosen protocol version, an RNG value, a session identifier (may be empty if not supported), the chosen cipher suite and compression method. The `ServerCertificate` message directly follows for trust establishment; see section 2.6.4. A `ServerKeyExchange` message might follow with additional parameters for the generation of the pre-master secret, this depends on the used cipher. Another optional message at this point would be the request to the peer for a certificate if mutual authentication is required, but this is usually not the case while browsing the Internet. The end of this transmission's messages is indicated by the `ServerHelloDone` message, which must be the last.

Another full round-trip was required for this message exchange, and the total time adds up to 136 ms. Assuming that all the information fitted into a single packet, as longer keys often require more transmissions. This should be taken into account for best performance. Also note that nothing was encrypted so far, and everything was sent in cleartext. The client now wants to validate the server's certificate if it sent one, which has to happen outside of the TLS system, and is covered in section 2.6.4. The negotiation phase eventually continues if the client trusts the server and everything seems valid:

3. The client sends its certificate if requested and available, but it always has to send the `ClientKeyExchange` message at this point. This contains additional parameters for the generation of the pre-master secret or is simply empty if no additional parameters are required by the used cipher. The `ChangeCipherSpec` protocol's message follows,

which instructs the server that the peer is now switching from cleartext to ciphertext. The final `Finished` message is created, which contains a MAC of all messages the client received and sent so far.

4. The server decrypts the parameters from the `ClientKeyExchange` payload, and validates the integrity of the complete negotiation phase based on the MAC the client transmitted. It also switches from cleartext to ciphertext, and informs the peer with the `ChangeCipherSpec` protocol's message about the change. The server's `Finished` message follows, which again contains a MAC of all messages it received and sent so far.

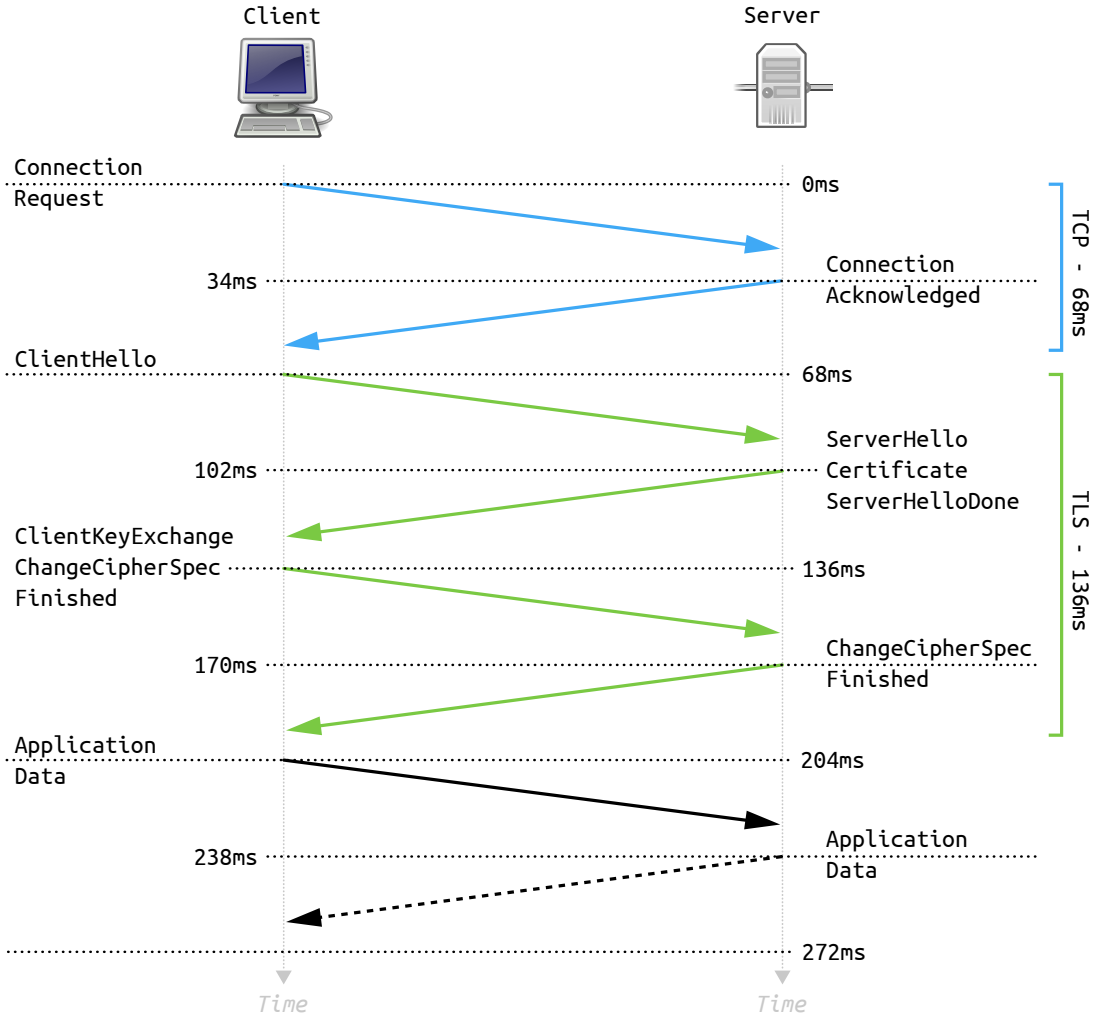


Figure 10: Full TLS 1.2 handshake (Fussenegger 2015c).

The total time for the message exchange is now at 204 ms and at least three full round-trips had to be completed, or in other words: a 6-way handshake. Any further commu-

nication will be encrypted and decrypted with a symmetric key for confidentiality, and a MAC for integrity based on the master secret that was agreed upon during the negotiation phase. Asymmetric cryptography was only used to establish a secure communication channel since it would be too slow for efficient communication. Session resumption or tickets (section 2.6.1) should be used to avoid the full negotiation phase, and perform an abbreviated handshake as depicted in figure 11.

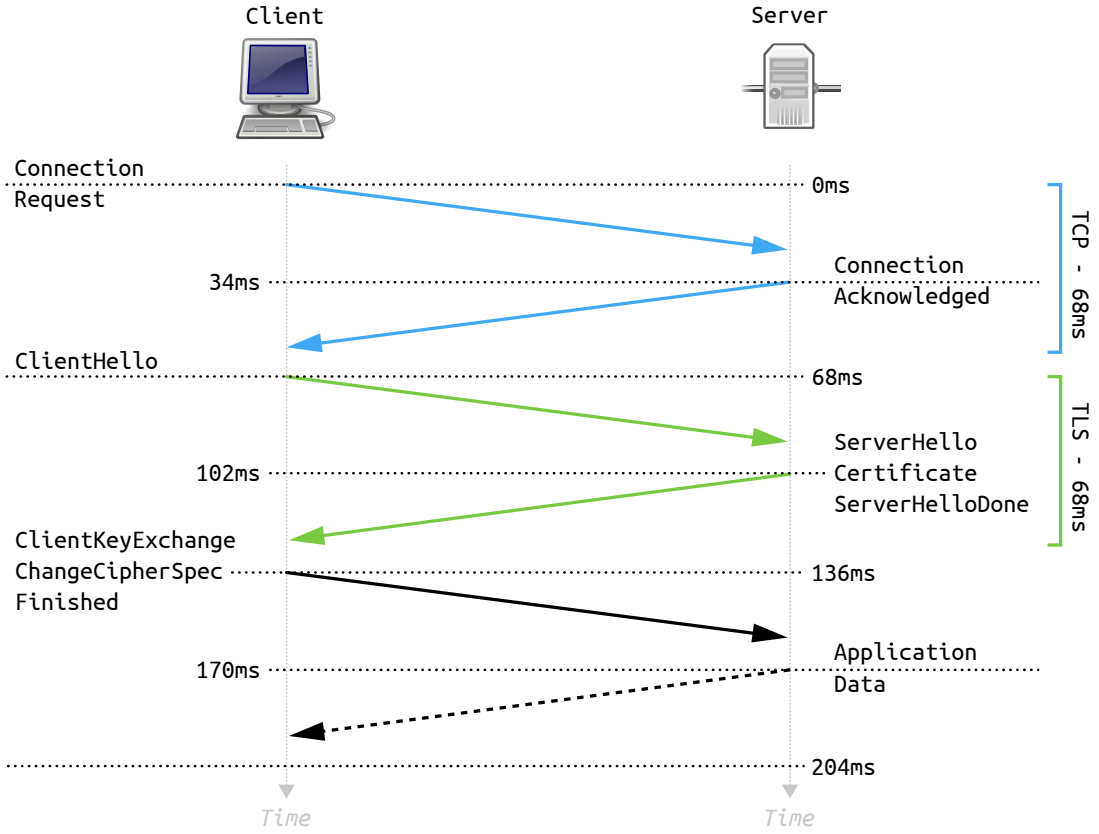


Figure 11: Abbreviated TLS 1.2 handshake (Fussenegger 2015d).

Session resumption has to be enabled on the server side, and an identifier will be included in the `ServerHello` message during the negotiation phase. A client may submit this session identifier in its `ClientHello` message the next time it wants to establish a TLS channel or if it wants to update the agreed upon cryptographic primitives in use. The server then checks for the identifier in its appropriate cache, and reuses that session information. The parameters that are used to secure the channel are always different since the random values from both hello messages differ. This abbreviated handshake is one of the simplest techniques to speed-up TLS connections by shaping off an entire round-trip for subsequent requests, plus it is available in almost all implementations (Ristić 2014,

25–32; Meyer 2014, 3–4; Grigorik 2013a, 47–52; Oppliger 2009, 75–76; Thomas 2000, 67; Dierks and Allen 1999, 23–46; Dierks and Rescorla 2006, 24–51; 2008, 26–63).

2.5.3 SSL & PCT: Encrypt the Web

The fully patented SSL 2.0 protocol specification was first published to the public in 1995 (K. E. Hickman and ElGamal 1995; K. E. B. Hickman and ElGamal 1997). The earlier revision SSL 1.0 was never released "due to conceptual weaknesses with regards towards security" (Meyer 2014, 29). The situation did not improve much with SSL 2.0, as detailed in RFC 6176. But the design of the handshake phase of these protocols was already similar to the current system described in the previous section, only the `ChangeCipherSpec` protocol and its messages were missing. And even the earliest known documents of SSL already included the session resumption feature (Meyer 2014, 30; Turner and Polk 2011).

Microsoft released its own specification called Private Communications Technology (PCT) to address some security flaws of SSL, and to have a similar technology in their Internet Explorer (IE) browser.[35] They also released two versions of this specification in a very short time. The intention of the first version of PCT was to fix the aforementioned security flaws, and the second revision added new features. Microsoft created PCT as an open standard from the very beginning, which ultimately lead to the transition from SSL to TLS and future standardization of the protocol through the IETF (Benaloh et al. 1996; Dierks 2014). But Netscape Communications Incorporation (Netscape) released another revision of the protocol before that, which is still in wide use today.

SSL 3.0 was a complete overhaul of its precursor and released in late 1995. It was a solid protocol with strong ciphers at that time, and it built the basis for all future work from the IETF (Ristić 2014, 3). The redesign mainly affected the handshake phase—which was the weakest point of SSL 2.0—with the introduction of the `ChangeCipherSpec` sub-protocol. The handshake was further extended by authenticating all data that has to be sent in cleartext before the actual encrypted session starts (Meyer 2014, 35–37). However, SSL 3.0 is comprehensively broken today, like its precursors. The faults are so severe that it is about to be deprecated by the IETF, and support for it must be deactivated in existing deployments (Barnes et al. 2015; Möller, Duong, and Kotowicz 2014; Duong and Rizzo 2011).

35. See https://en.wikipedia.org/wiki/Browser_wars for more background information.

2.5.4 TLS 1.0: Open Standard

TLS 1.0 is the first version released under the standards body of the IETF. The name SSL was dropped and exchanged with TLS as part of the takeover agreements between Netscape, Microsoft, and the IETF (Dierks 2014). The transition of the protocol was very slow; the working group started their work in 1996, and the final release of TLS 1.0 was RFC 2246 in January 1999 (Ristić 2014, 4; Dierks and Allen 1999). However, the version number in the TLS record was only incremented by a minor number and set to `3.1` while not being backward compatible:

> *The differences between this protocol and SSL 3.0 are not dramatic, but they are significant enough that TLS 1.0 and SSL 3.0 do not interoperate (although TLS 1.0 does incorporate a mechanism by which a TLS implementation can back down to SSL 3.0).*
>
> — (Dierks and Allen 1999)

The major changes to the protocol included a detailed specification of a pseudo random function (PRF) for key derivation. The `Alert`, `CertificateVerify`, and `Finished` messages were changed, and the `ChangeCipherSpec` became mandatory before the `Finished` message to mitigate the drop attack (Schneier and Wagner 1996). The last major change was the replacement of the MAC with a HMAC (Meyer 2014, 37; Dierks and Allen 1999).

2.5.5 TLS 1.1: Improving Security

RFC 4346 was released in April 2006 to account for various security issues found in TLS 1.0. The addressed vulnerabilities included the prevention of attacks against the Cipher Block Chaining (CBC) operation mode through explicit inclusion of IVs for such ciphers. This also made the key derivation process simpler since only four keys needed to be derived from the master secret in contrast to the previous six. The handling of padding errors was changed to mitigate the (Bleichenbacher 1998) attack, and the usage of the so called export ciphers—which are weak by design[36]—was prohibited. Another change included that connections that were terminated without a `close_notify` were still resumable (Meyer 2014, 38–39; Dierks and Rescorla 2006, 5).

36. Ciphers which have the `EXP` or `EXPORT` key in their name; see https://www.openssl.org/docs/apps/ciphers.html.

2.5.6 TLS 1.2: Current Standard

The latest and current revision of the protocol was released as RFC 5246 in August 2008, and it mainly addressed security issues like TLS 1.1 did. AEAD ciphers were introduced, which also forced the reintroduction of IVs in the key derivation process. The MD-5 and SHA-1 combination was removed from the PRF as well as digitally signed elements in favor of configurable hash functions. This change also simplified both parts because a single hash function could be used on the whole data. TLS extensions were merged into the protocol and made RFC 3546 obsolete.

SSL 2.0 was deprecated while IDEA and DES ciphers were removed completely, plus a few other changes. The protocol became much more flexible through the removal of hard-coded cryptographic primitives. This should ensure that emerging vulnerabilities and new ciphers of the future were easier to compensate and implement with the introduction of extensions rather than the development of a new revision as it was the case in the past (Meyer 2014, 39–40; Ristić 2014, 4; Dierks and Rescorla 2008, 4–5).

2.5.7 TLS 1.3: Next Generation

The need to revise the protocol once again arose from a wave of new attacks on the protocol itself and its surrounding infrastructure, and the publication of SPDY and its successor HTTP/2 with their de facto requirement for encryption. Performance came more into focus in comparison to earlier revisions of TLS because the transition to a fully encrypted web with those new protocols requires higher efficiency. The IETF working group started in the beginning of 2014 with a formalization of the primary design goals for TLS 1.3 (Turner and Morgan 2014).

One objective is to encrypt more data of the negotiation phase to minimize observable cleartext messages for passive and active adversaries. The initial phase should be further optimized by reducing the amount of round-trips that are necessary to establish a secured channel. Full handshakes aim on as little as a single round-trip (1-RTT) and no round-trip (0-RTT) for the abbreviated handshake in the best case. Further goals are of course updates to used algorithms, ciphers, and mechanisms to reflect the current state of cryptographic research and to avert latest attacks.

The protocol is still based on SSL 3.0 and even maintains backward compatibility because the `ClientHello` message will not change. However, support for SSL 3.0 itself will be prohibited for reasons explained in section 2.5.3. Overall, the changes go much further than in any of the previous protocol revisions but are still in flux with an estimated

final release in 2015 or 2016. Nonetheless, some of the major changes directly targeting the main goals can already be summarized at this point because their inclusion is effectively necessary even if details are not agreed upon yet.

An overhaul of the negotiation phase and the TLS Handshake Protocol was inevitable in order to support extended encryption and a 1- to 0-RTT flow. It is important to note at this point that the RTT count the working group is referring to does not include TCP's initial round-trip. This was not overseen by the authors, it simply is not part of TLS, which assumes a reliable transport but does not care how this has to operate. However, (Rescorla 2015a) is a collection of ideas for new handshake flows that can provide the targeted goals. It also contains possible problems that have to be considered, and builds the basis for the new negotiation phase of TLS 1.3.

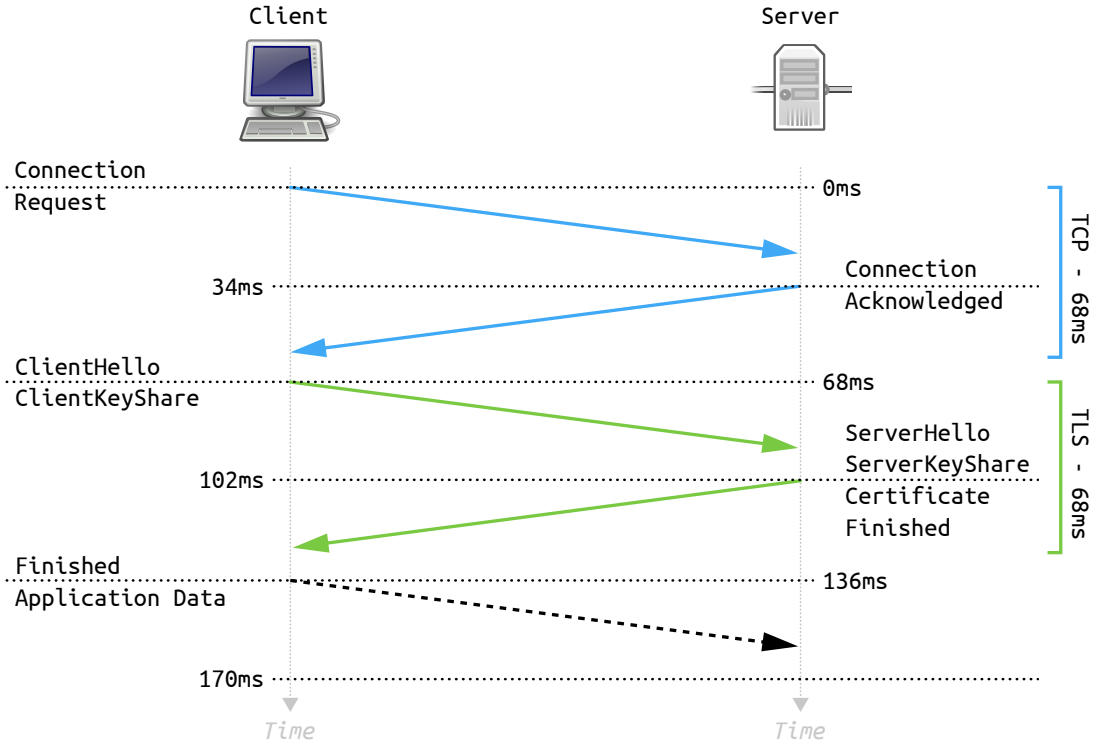


Figure 12: Full TLS 1.3 handshake (Fussenegger 2015e; Rescorla 2015b, 31).

Only the initial `Client-` and `ServerHello` messages from TLS 1.2's full handshake will remain in cleartext and their structure stays the same. Additionally, two new cleartext messages are added on both sides for the same flight, the `Client-` and `ServerKeyShare`, as depicted in figure 12. They contain additional cryptographic parameters for calculation of the pre-master secret, but which parameters exactly will depend on the cipher chosen by the server. This means in effect that the client will always send the information before

it actually knows which cipher the server will choose. This is why it can include multiple parameters for different ciphers or even send an empty key share if it has not enough information to include anything.

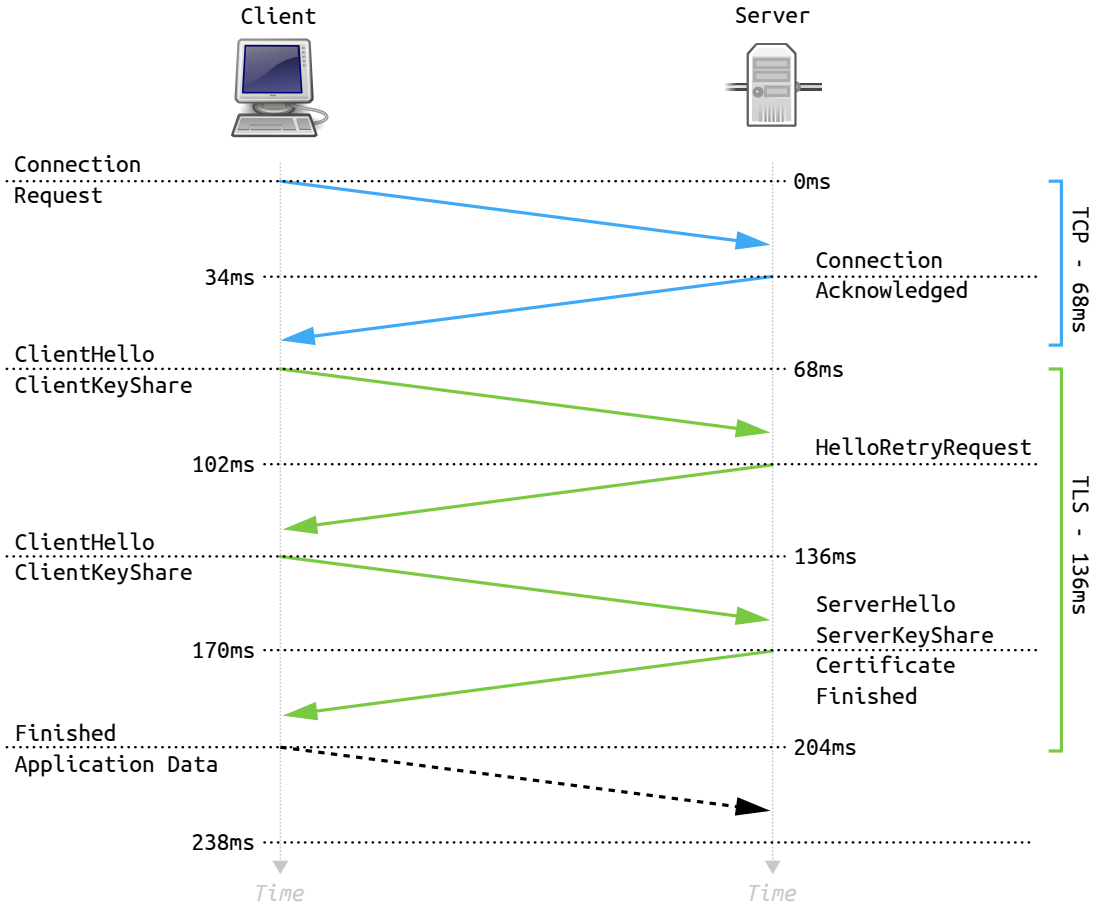


Figure 13: Full TLS 1.3 handshake with mismatched parameters (Fussenegger 2015f; Rescorla 2015b, 32).

The fact that the client may send nothing or unusable parameters forced the authors to implement a special message: the `HelloRetryRequest` server message, illustrated in figure 13. Upon receiving of this message the client has to resend the same `ClientHello` and `ClientKeyShare` messages but extend the content of the latter with the requested cryptographic parameters. From here on, the flow is the same again for both handshake situations, and the server will answer with its `ServerHello` and `ServerKeyShare` messages.

In the same flight the server can already include various optional but encrypted messages followed by the mandatory `Finished` message. The client may also answer with additional optional messages, but it can directly include encrypted application data after

the mandatory `Finished` message. The abbreviated handshake, illustrated in figure 14, as it is planned now still requires a round-trip for session resumption, but the first application data may already follow the `Finished` message of the client in the second flight (Rescorla 2015b, 29–32).

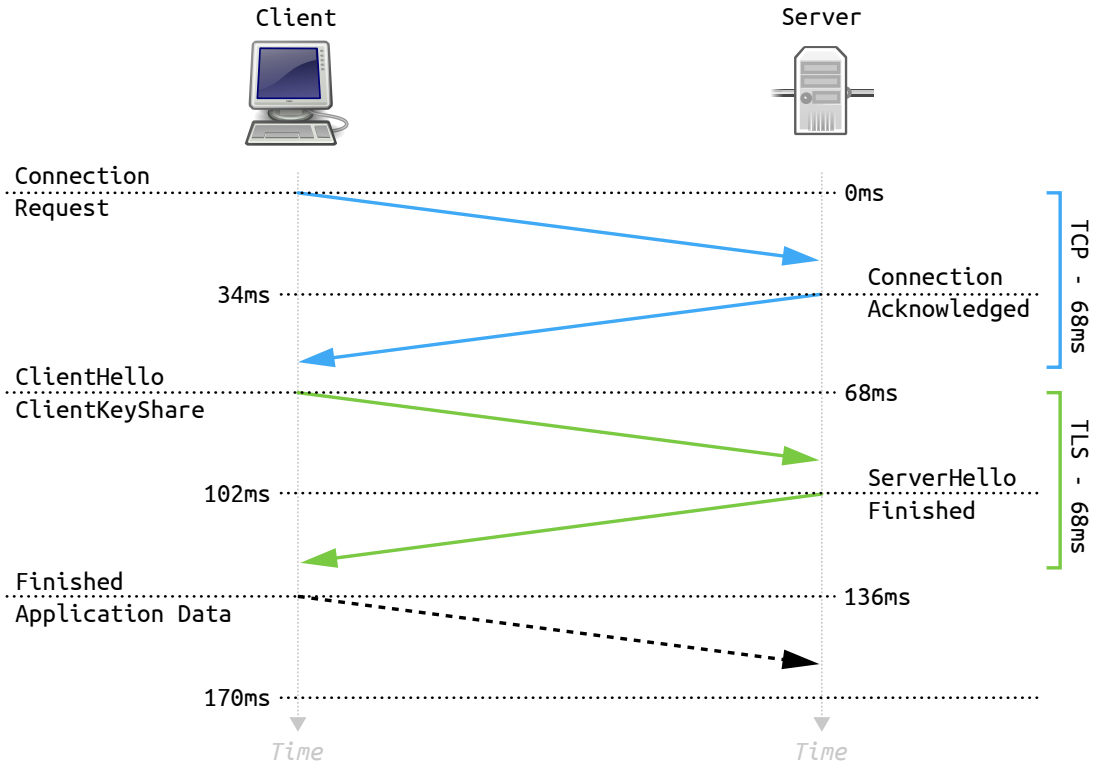


Figure 14: Abbreviated TLS 1.3 handshake (Fussenegger 2015g; Rescorla 2015b, 33).

The extended encryption of the TLS 1.3 Handshake Protocol counteracts various privacy leakages that are present in current revisions. For instance, the host to which a client connects is not visible anymore if Server Name Indication (SNI) is in use. SNI allows hosting of multiple HTTPS websites with a single IP address and was first defined in RFC 3546 (Blake-Wilson et al. 2003, 8–10). Other data that might be of interest to third parties for fingerprinting or reassembling of communication, namely timestamps and sequence numbers, were either dropped altogether or are encrypted. The shortened phase also requires the deprecation of previous static RSA and DH in favor of dynamic DHE and ECDHE algorithms that automatically provide PFS.

Other changes currently included in the draft contain the removal of non-AEAD ciphers, compression, custom DHE groups, the `ChangeCipherSpec` protocol and its single message, and much more. None of these changes are directly targeting performance, and an in-depth explanation of them can be found in (Rescorla 2015b). Recapitulatory, it can be

said that the upcoming protocol trims a lot of overhead from the various messages during the negotiation phase, which in turn minimizes the chances that a packet grows too big and requires fragmentation. Further, security, privacy, and simplicity are improved, which should in turn support implementors and researchers during protocol evaluations.

2.6 TLS Extensions and Related Technologies

The following sections will describe important TLS extensions and related technologies that are essential for the ecosystem of encrypted communication in the Internet.

2.6.1 Session Tickets

Session tickets were first defined in RFC 4507 and updated in RFC 5077. This TLS extension enables a server to offload a session's state to the client instead of keeping it in a local cache, as was described in section 2.5.2 with session resumption. The state is encrypted and encapsulated in a so called ticket, which the server forwards to the client. The client can resubmit the ticket to the server at a later point if it wishes to resume the session. The main advantage of this approach is that a server does not require any kind of cache management. In theory it also simplifies sharing of session states among multiple servers and makes scaling of clusters easier.

Only the key that is used to encrypt the encapsulated tickets needs to be shared among the whole cluster, in contrast to the complete cache. Synchronization of the key yields its own problems, very similar to the ones described in section 2.4.3, that need to be solved. The most severe problem, which generally results in the recommendation to disable session tickets altogether, is the lifetime of the key. PFS can only be guaranteed if the key is frequently rotated since a compromise of the key would otherwise result in a compromise of all past communication. An attacker who recorded all encrypted messages would obviously be able to decrypt them with the appropriate session ticket key.

Frequent rotation is not supported by any major platform, and administrators and developers are left to implement their own solution if they wish to use session tickets and provide PFS. The latter being available by default in the upcoming TLS 1.3 specification if static key exchange algorithms are really dropped in the final revision. A possible open source solution for session ticket key rotation for nginx is presented in section 3.3 (Ristić 2014, 58–59; Grigorik 2013a, 57; Salowey et al. 2006, 2008).

2.6.2 HTTP Strict Transport Security

HTTP Strict Transport Security (HSTS) is a mechanism that was introduced to improve security of websites that are only accessible via secure connections and was defined in RFC 6797; see listing 9 for an example. The idea is simple: the server includes an HTTP header that indicates that this domain, and optionally all its subdomains, should always be visited via the appropriate secured protocol (usually HTTPS). A compliant UA will convert any request and Uniform Resource Locator (URL) that is not using the secure protocol to use one. Any certificate errors will yield a hard-fail since requesting of a non-secured resource is not allowed. This technique mitigates active and passive network attacks and deployment bugs.

```
1 Strict-Transport-Security: max-age=262974383; includeSubdomains; preload
```

Listing 9: HSTS example header with max-age, subdomains, preloading attributes.

Major browsers additionally support HSTS preload lists.[37] These lists are hard-coded into the browser, and hosts found on that list are impossible to access via non-secure protocols. HSTS can be used to minimize unnecessary HTTP-to-HTTPS redirects and improve performance because UAs will alway directly access the server via the appropriate protocol (Ristić 2014, 285–93; Grigorik 2013a, 72; Hodges, Jackson, and Barth 2012).

2.6.3 False Start

False Start is a TLS protocol extension (Langley, Modadugu, and Moeller 2015) that allows peers to transmit application data together with the `ChangeCipherSpec` and `Finished` messages. This means in effect that the negotiation phase is reduced to a single round-trip even if the client visits the server for the first time or the previous session has expired. False Start is available for all TLS revisions since it does not alter the protocol; only the timing when application data is sent is affected. A client knows the encryption key with transmission of the `ClientKeyExchange` message, thus, it can already encrypt application data. See figure 15 for an illustration of the altered TLS negotiation phase with earlier transmission of encrypted application data.

Subsequent processing is performed to confirm that the handshake records were not tampered with, which can be done in parallel. This TLS extension was, once again, proposed by Google after performing several tests against all SSL/TLS hosts known to their own search index. Preliminary tests of them showed that False Start can reduce the

37. https://hstspreload.appspot.com/

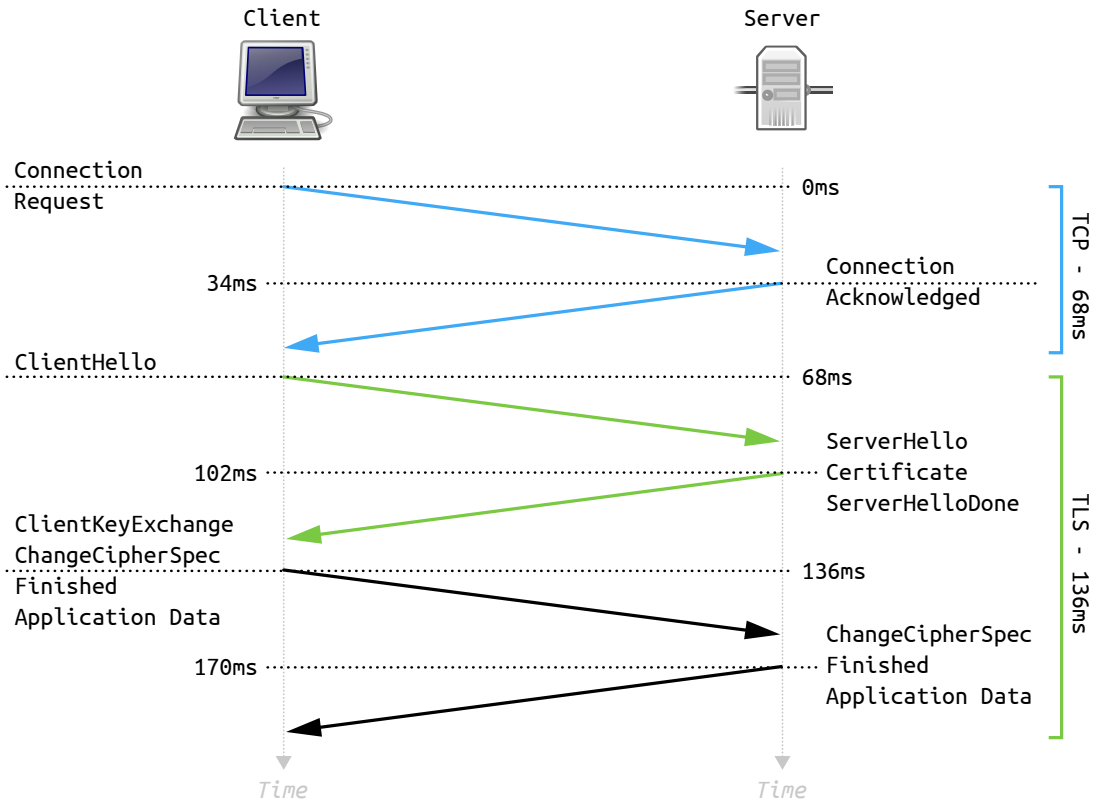


Figure 15: TLS False Start handshake (Fussenegger 2015h).

latency of the negotiation phase by up to 30 % (Belshe 2011). Though, their tests also revealed that some endpoints—mainly TLS terminators, which are hardware devices that proxy unencrypted data to backend HTTP servers—fail if application data is sent to early (Langley 2012).

That is why web browsers will only use False Start if certain requirements are met:

- Chrome and Firefox require Next Protocol Negotiation (NPN) or its successor ALPN and a cipher suite with the PFS attribute.
- Safari requires a cipher suite with the PFS attribute.
- IE uses a blacklist of known sites that break with False Start, and a timeout to repeat the handshake if it failed.

Cipher suites with PFS and the announcement of available protocols via the NPN or ALPN TLS extensions will enable False Start across all browsers. This means in practice that a web server, like Apache HTTP Server (Apache) or nginx, has to be compiled against a recent OpenSSL version (1.0.1a or higher) that will automatically enable the necessary extension, and pair it with a suitable cipher suite selection (Grigorik 2013a, Chapter 4 from the online version; 2013b).

2.6.4 Public Key Infrastructure

Public key infrastructure (PKI) is a system that issues, distributes, and validates digital certificates. It was already mentioned that TLS does not mandate how the authentication certificate of a peer should be interpreted. However, the need to validate the authenticity of a peer is important in order to ensure that two parties who have never met are actually the ones they claim to be. In the context of the WWW this usually means that the server should proof that it is a valid origin for the data it provides and not an attacker who is trying to spoof passwords, for instance.

An IETF working group developed the Internet PKI especially for the WWW and formalized it in RFC 5280 (Cooper et al. 2008). The model depends on certificate authorities (CAs) that issue certificates for various purposes. A certificate encapsulates an entity's public key, various meta information, and is signed with the digital signature of a CA. The included meta information decides what purpose a certificate may be used for, including signing of other certificates, documents, emails, or domain names. Each CA has its own self- or cross-signed certificates, known as root certificates because of the chaining requirement.

A server certificate should always be signed by an intermediate CA's digital signature

and not directly by the CA itself for security, technical, and administrative reasons. The most important and obvious reason is that a CA's root key is very important and should never be compromised; otherwise, all certificates that were ever signed by it would have to be invalidated. Other reasons include cross-certification, compartmentalization, and delegation to other organizations. The chaining of certificates often yields confusion, and some include no chain or the complete chain when a client requests the server's certificate, which leads to invalid paths.

A path should always lead to a root certificate, but it should not include it. Including the root certificate only wastes bandwidth because a system will not trust a CA simply because its root certificate is part of a chain. The system will trust the CA because it is part of its own trust store. Such a store is a collection of root certificates that is provided by the system. Operating systems usually provide such trust stores to installed software, but any vendor may maintain its own collection, like Mozilla does. Missing intermediate certificates will result in an error since the system is unable to complete the chain to the appropriate root certificate.

The validation process starts if a complete chain is present. The system will first validate the expiration date, as server certificates are usually valid for one to two years, and if it is not expired continue to ask the CA if the certificate was not revoked. Revocation usually happens if the owner of a certificate reports that it should be revoked. This could be the case because of a compromise of the associated public key or simply because it is not needed any longer. There are currently two standards available for certificate revocation. The older is known as certificate revocation list (CRL) and is also defined in RFC 5280, and the newer is called Online Certificate Status Protocol (OCSP) and defined in RFC 6960.

CRLs are a long list of certificate serial numbers that are revoked but have not expired. Every CA maintains one or multiple CRLs, and they are made available to systems over the Internet. The main problem with these lists is that they are often very large, which makes real-time lookups slow. OCSP was designed to solve that problem by allowing a system to check the revocation of a single entity. The actual lookup is performed by a so called OCSP responder, which is operated by the CA and again available over the Internet. This system introduces new problems in form of poor performance, single points of failure, and privacy. This can be mitigated with OCSP stapling, which allows the inclusion of an OCSP response directly in the TLS handshake.

Revocation is still problematic, even with the two systems in use. This is why many UAs perform a soft- instead of a hard-fail if a OCSP responder is unreachable, which

means that they treat the certificate as not revoked. Google goes even further and has disabled revocation checking in their Chrome browser (Ristić 2014, 63–112; Grigorik 2013a, 57–62). There are many more problems with the current Internet PKI system, all of them related to trust and security, and not to performance (Ristić 2014, 77–78; Cooper et al. 2008; Santesson et al. 2013).

2.6.5 Web Servers

The term web server may refer to a computer that is connected to a network or to a single software that is responsible for delivering web pages to clients, and processing or storing instructions sent by them. In this thesis, web server always refers to the software and server is used to refer to the network connected computer. Web server software play an integral role because they are responsible for handling all incoming and outgoing requests, orchestration of the correct system calls at the appropriate time, and much more. Performance is always directly related to the web servers in use, and many software is available[38] with different approaches to handle all of these requirements.

Apache HTTP Server (Apache) is historically the best known web server software with the biggest market share. It is implemented in user land—like most web servers—and features different architectures from which an operator can choose. The older and mostly deployed architecture is process-based, and tries to reduce latency while increasing throughput to ensure reliable processing of requests. The second architecture is a mixture of multiple processes and threads. It was introduced to increase the amount of simultaneously servable clients, and as an answer to newly emerging event-based web servers that featured much better performance than Apache did at that time.

Nginx (pronounced “engine x”) is such an event-based web server. It was developed by Igor Sysoev for Rambler, the biggest Russian search engine, because existing web servers were not fast enough. The main focus is high concurrency, performance, and low system resource usage. It utilizes an asynchronous event-driven mechanism to handle requests for highest throughput, and a highly modular codebase to improve extensibility and maintainability. It found broad deployment because of its outstanding performance, and constantly moves upwards in the NetCraft web server survey[39] statistics.

38. https://en.wikipedia.org/wiki/Comparison_of_web_server_software
39. http://news.netcraft.com/archives/category/web-server-survey/

2.6.6 Security Libraries

Implementing cryptographic functions and logic is not a trivial task, in fact most security issues are related to incorrect implementations and bugs. This became particularly evident with the serious and often discussed Heartbleed vulnerability in OpenSSL. This programming mistake of a single developer enabled anyone to read the memory of a machine that is connected to the Internet, and possibly steal the private key that is in use. This circumstance was further aggravated by the fact that OpenSSL is the most used security library in non-Windows software—like web servers—to implement SSL and TLS (Ristić 2014, 162–63).

Other libraries[40] like NSS from Mozilla, GnuTLS from the Free Software Foundation, SChannel from Microsoft, or the Java Secure Socket Extension (JSSE) exist, but none of them is a simple drop-in replacement for OpenSSL, and, of course, they are not free of bugs either. Such drop-ins are important because a lot of big software is hard-coded against the public OpenSSL API, like Apache and nginx. Altering such implementations to use a completely new API is often complicated or simply not feasible. This is why other projects were established after the Heartbleed discovery, namely LibreSSL and BoringSSL that shall maintain backward compatibility while improving security and performance.

The former is a fork of OpenSSL that was started by various programmers of the OpenBSD Project, and the latter is a fork of Google. Both have the goal to streamline the codebase and they exchange modifications with each other. Their efforts also result in improved performance, although much more would be possible and might be available in these libraries in the future (Grigorik 2014, time index 5:25 to 6:40). As of now, it seems as if LibreSSL is doing a better job as a drop-in replacement than BoringSSL, which is mostly driven by the interest of Google to use it for its own software rather than being a dedicated library for all kinds of other software, like LibreSSL (Stenberg 2014).

2.7 Previous and Related Work

This section is dedicated to analyzing and discussing previous and related works—both academic and non-academic—that deal with the same scientific question as this thesis. A lot of research was conducted on the performance of SSL and TLS since their inception, and only the most relevant publications that directly deal with performance and not security were considered. However, there is no claim on completeness.

40. https://en.wikipedia.org/wiki/Comparison_of_TLS_implementations

2.7.1 Historic Work

One of the first studies conducted on the performance of SSL was (Goldberg, Buff, and Schmitt 1998a) that evaluated the impact of session resumption. They concluded that abbreviated handshakes result in a 15 % to 50 % reduction in the time it takes to establish a channel. Additionally, they observed that the time for establishment of a new TCP connection is only slightly lesser than the time it takes to resume an SSL session. The results of this work are still relevant today, and enabling session resumption for TLS is the simplest method to speed-up encrypted websites. Modern UAs even wait for the first negotiation phase to finish before establishing more connections to a host in order to reuse the session, and to avoid the costs of the full handshake (Grigorik 2013a, 56).

Another publication of the same authors followed a few months later, where they compared the performance of HTTP and HTTPS in general (Goldberg, Buff, and Schmitt 1998b). This work must be considered historic because of the benchmarked ciphers and web servers, although the used RC4 algorithm with 40 bit and 128 bit is still in use today by several websites. The reason is simple: it was promoted as the best mitigation against the Browser Exploit Against SSL/TLS (BEAST) and Lucky Thirteen attack (Lucky 13), despite the fact that the stream cipher was already considered problematic for a long time. RC4 is broken since 2013 and prohibited via RFC 7465 for usage in TLS (Ristić 2014, 197–98, 218–23; Popov 2015). However, the work concludes that encrypted communication is similar in terms of speed to its unencrypted equivalent with a transfer rate (B/s) drop between 17 % to 20 %.

TLS 1.0 was released in 1999 and (Apostolopoulos, Peris, and Saha 1999) were the first to benchmark the new protocol. Their results are again only of historic nature because of the used ciphers, security libraries, and web servers. Their approach was a traditional load test, where they measured the number of connections the servers could handle in a second. Such a benchmark is more meaningful than the transfer rate measurements of the previous work because it is simpler to compare it with real world deployments that are not within a controlled laboratory environment. Apostolopoulos et al. conclude that the impact of encryption is huge with a drop from ∼250 unencrypted requests per second down to ∼10 without session caching, which is ∼94 % less. Finally, they propose that some of the computational overhead should be shifted from the server to the client in order to accelerate the negotiation phase with the argument that the server needs to handle many more handshakes than the client does.

Another historical work was (Kant, Iyer, and Mohapatra 2000) with special attention on CPU impact of SSL operations as well as the total throughput of HTTPS transactions.

They used various hardware configurations and different files while measuring several characteristics of the CPU (cache, prediction) and the time to first byte (TTFB). The total drop of the observed TTFBs was ∼85 % during their experiments of encrypted to unencrypted transactions. Furthermore, they observed that the involved operations are highly CPU bound and conclude that faster or multiple CPUs have a significant impact on SSL web server performance.

(Coarfa, Druschel, and Wallach 2002) conducted a comprehensive study of the performance impact on the various processing stages of TLS through removal of relevant code and insertion of no operation (NOP) instructions. This approach should simulate a perfect operation that has no overhead, and allowed them to estimate the maximum expectable improvement for the overall system based on Amdahl's Law. Apache was the only tested web server, but they also used RSA accelerators and compared them to CPUs. The findings of Coarfa et al. showed that such cards are not preferable compared to dual CPU servers, which often outperformed them, although RSA was the algorithm that was accountable for the highest relative CPU cost of all involved cryptographic operations. They also conducted throughput benchmarks, similar to the experiments of (Apostolopoulos, Peris, and Saha 1999), and saw a drop from ∼1,650 unencrypted requests per second down to ∼320, which is ∼80 % less. Increasing the CPU speed or count yielded significant improvements, and they deduced that faster or more CPUs would solve the problem of cryptography in the future.

(Zhao et al. 2005) conducted an in-depth study of the operations that a web server has to perform during SSL transactions. This work also illustrates well how long old protocols and procedures are of interest, even in the scientific field. TLS, the SSL successor, has already been released for over six years when this work was created purely addressing the old revision. However, Zhao et al. focused on the time that is spent on the various cryptographic operations during the negotiation and application phases. They found out that the majority of total processing time is spent on SSL during HTTPS transactions with an overhead of 70 %.

The takeaways from these historic works are that SSL and TLS imposed an immense performance degradation that was almost always highly CPU bound. Especially the asymmetric cryptography part was costly to compute, which is also the reason why RSA accelerator cards were often used in benchmarks. Today, such cards are not very common and CPUs are considerably faster while featuring multiple cores with special built-in instructions for various cryptographic algorithms.

2.7.2 Current Work

Many more studies and papers were published, but most of them concentrated on particular aspects of HTTPS performance, like a single algorithm or how expensive operations could be off-loaded to the client. Other systems to encrypt the WWW were also proposed; most notable are Obfuscated TCP (ObsTCP, which was rejected by the IETF in favor of TLS) and tcpcrypt. The release of SPDY resulted in an increased interest in performance among researchers, although not particularly in the impact of encryption.

Google was, of course, first to publish benchmarks and announce that SPDY would improve PLTs by a factor of two (Belshe and Peon 2009). A few years later, another work followed, where a mobile device was used to load several web pages over HTTPS and SPDY. This experiment resulted in a PLT improvement for SPDY of 23 %, which is a factor of 1.3 (Welsh, Greenstein, and Piatek 2012).

A few months later, (Podjarny 2012) published a blog post where he compared HTTP, HTTPS, and SPDY over a reverse proxy without removing many of the HTTP/1 optimizations that are known to hinder SPDY. The results showed that SPDY is only 4.5 % faster than HTTPS and 3.4 % slower than HTTP.

Microsoft also published benchmarks between SPDY (with and without encryption), HTTP, and HTTPS. Their explicitly declared preliminary results show that HTTP and HTTPS often keep up with SPDY, especially in low RTT environments, and that the impact of encryption should be further studied in future work (Padhye and Nielsen 2012).

(Servy 2013) conducted server load benchmarks with Apache and again with HTTP, HTTPS, and SPDY. He used his own commercial load testing tool to perform the benchmarks with a single domain and a small HTML document that contained fifty to one hundred small images. This is important to note because such a scenario is best suited for SPDY. The results show that the server was able to handle many more successful transactions while using fewer resources with SPDY; HTTPS had the worst performance.

The first academic work was published in 2013 by a researcher team from AT&T Labs. Their study also focused on the impact of SPDY on mobile networks with special attention to various short-comings of TCP. HTTP and SPDY showed similar performance in their tests, and the researcher team concluded that the problems are, on the one hand, the web pages that are optimized for HTTP/1, and, on the other hand, that TCP is not well suited for mobile networks (Erman et al. 2013).

It is clear from the varying results that Google attests SPDY better performance than other parties. This is a circumstance that was also identified by (Wang et al. 2014) who then performed a thorough study with the goal to clarify the situation. They

captured the PLT of 200 Alexa Internet top sites with and without encryption for both protocols. Their conclusion is that SPDY only yields an improvement of 7 % over all tested scenarios, additionally they state that "the overhead of SSL is too small to affect SPDY's performance." (Wang et al. 2014, 398)

2.7.3 The Cost of the "S" in HTTPS

The latest work, which solely focused on the overall performance of TLS, was released in December 2014—after the decision to write this thesis was made—by a group of eight researchers with fundings from the European Union: "The Cost of the 'S' in HTTPS". The first section of the paper is dedicated to the growth of HTTPS traffic in recent years. The evaluation of (Naylor et al. 2014) is based on log files from a European ISP with approximately 25,000 users. TStat was used to analyze all requests, which is a software capable of extracting information from the TLS negotiation phase, and it illustrates how probably private data leaks through the initial cleartext messages of the protocol are (Mellia, Carpani, and Lo Cigno 2003). This circumstance will be addressed with the release of TLS 1.3.

However, the third section of the paper concludes that 50 % of all HTTP transactions (Naylor et al. observed) are encrypted. This high amount is mainly due to the fact that most top visited websites like Google Search, Facebook, and YouTube are fully encrypted. Most interestingly, although only mentioned in a footnote, 55 % of the clients offered SPDY, but only 5.5 % of the hosts actually supported it (Naylor et al. 2014, 134–35). It must be mentioned at this point that the presented graphs are not very meaningful and difficult to interpret, especially the ones presented in subsequent sections. Simpler graphs would have been supportive to reach a broader audience (UNECE 2009, 17–30).

The next section of "The Cost of the 'S' in HTTPS" is split into two subsections: in the first, the total PLT of the Alexa Internet top 500 websites is investigated, and, in the second, the costs of the TLS negotiation phase are discussed. The PLT tests were executed with PhantomJS, a headless web browser based on the Qt WebKit engine with a scriptable JavaScript (JS) API, on an unspecified Linux OS. Each of the 500 websites was loaded twenty times without caching via HTTP and HTTPS over two different connections, which were third generation (3G) mobile and fiber connections, both unspecified. The so collected data shows that TLS adds extra latency ranging from 500 ms to over 1.2 s for almost all websites on the mobile connection, and more than 500 ms for 40 % on the fiber connection.

It is safe to assume that Naylor et al. loaded the homepage of each website, but it

is unclear how they handled redirects. This is especially important with websites that are supporting only encrypted traffic with a proper HTTP-to-HTTPS redirect. Such responses are generally not comparable with the response of a real page. Not only does the payload differ, the redirect is usually issued from the web server, and no back-end processing is involved. PhantomJS does not support HSTS and does not have preloading lists, which means that it will connect to the HTTP website and download the small payload or follow the redirect. The former would result in faster and the latter in slower or almost equal PLTs for the HTTP benchmarks.

The remarks of Naylor et al. continue with a comparison of TCP connection reuse between unencrypted and encrypted transactions. They observe that connection times incur a significant impact from TLS, and, more interestingly, that many websites use fewer connections and serve fewer resources if accessed via HTTPS. The evaluation of the TLS negotiation phase follows for which the researchers extracted one million examples from one day of the ISP dataset. Furthermore, they extracted five popular services from the set and revealed a relation between the RTT (geographical distance, see section 2.1.1) and the time it takes to establish an encrypted channel. While most results are somewhere in the midfield all services exhibit very long timings for some transactions, which is most likely due to network congestion (Naylor et al. 2014, 135–36).

Data usage is the subject of the next section of "The Cost of the 'S' in HTTPS", which is again split into two parts. On the one hand, the average overhead of TLS is determined through evaluation of the channel usage (transferred data) that results in only 4 % of the total volume, and, on the other hand, the impact on proxies is evaluated. They provide different services like caching (ISP savings) or optimizations (user savings). The datasets of Naylor et al. indicate that ISPs would see a large increase in traffic, while users would not see a significant change in data consumption (136–37).

The last section that is based on experiments evaluates the impact of encryption on the energy consumption of mobile devices. Collected data indicated that TLS has almost no impact, and that proxies deteriorate the consumption. Naylor et al. state that the benefit the proxy provides outweighs the increased consumption because the proxy rewrites the requested *webm* to an *mp4* video format that the cellphone is able to decode in hardware, which it cannot do if the video is requested over an encrypted channel. Evolving web technologies, like the HTML5 *<video>* tag, allow UAs to select the format that is best suited for them. The UAs have to be the solution for such use cases and not proxies. Relying on third-parties also comes with high risks, especially if the third-party is entirely opaque to the user (Kravets 2014; Thoma 2014).

The authors of "The Cost of the 'S' in HTTPS" discuss the potential impact of losing in-network services with emphasis on their positive effects and usages. Malicious or conscious negative usage either to harm and eavesdrop the users or to increase income through authorities, criminals, or ISPs is not mentioned (Naylor et al. 2014, 137–38; Kravets 2014; Thoma 2014). Furthermore, they argue that: "[…] losing the ability to block tracking cookies hurts privacy, which is one of the goals of using TLS to begin with." (Naylor et al. 2014, 138) This is a very common misconception about the goals of encryption in general, though, TLS 1.3 is actually going to improve privacy (a bit) (Sniffen 2014).

2.7.4 Ongoing Work

The academic world was and is highly interested in the performance of cryptography, but work which is benchmarking complete systems is rare and mostly of theoretical nature. Administrators and developers are much more interested in increasing the performance of their applications, and hackers and organizations are much more interested in a fully encrypted WWW. Blog posts, discussion boards, mailing lists, and other documents in the Internet contain the bulk of the work done within this field of research. This is also due to the fact that the results are very transient and volatile.

The following is a collection of publications and applications of recent years that try to collect, summarize, and evaluate TLS. This list completes this section of the thesis with further interesting material and tools that can help in administrating or developing (encrypted) web applications:

- "High Performance Browser Networking" (Grigorik 2013a) is a book that contains a complete chapter on TLS, and also investigates other important aspects of web applications in terms of performance. It was a major source for this thesis.
- "Is TLS Fast Yet?"[41] features a collection of information around TLS, and is a follow-up to the previously presented book.
- "Bulletproof SSL and TLS" (Ristić 2014) is a book that is not exclusively about performance, but it features some chapters on the topic, and it contains a very good high-level introduction and overview of everything involved in the TLS ecosystem.
- "Qualys SSL Labs"[42] features multiple interesting projects:
  - An online tool to test and grade a web server's SSL/TLS implementation.

41. https://istlsfastyet.com/
42. https://www.ssllabs.com/

  - An online tool to test a UA's SSL/TLS implementation.
  - "SSL Pulse"[43] is used to publish implementation surveys of thousands of SSL/TLS web servers across the WWW.
- "ImperialViolet"[44] is Adam Langley's weblog, where he constantly publishes articles around TLS and related technologies. He is a senior staff software engineer at Google, and works on encryption on both the server and client side.

2.8 Benchmarking Approaches

It is obvious that measurements must be carried out to find out how much encryption affects the performance of web applications. Unencrypted communication must first be quantified in order to compare it to its encrypted counterpart. In addition, new protocols and extensions must be considered, which might counteract the speed degradation induced by encryption. Several approaches to perform these measurements were used by the authors of the works that were presented in the previous section 2.7. All of them used valid approaches and combinations are also possible, but each approach comes with its own difficulties and combinations will add them up.

2.8.1 Performance Tests

Performance tests, as conducted by (Goldberg, Buff, and Schmitt 1998b; Apostolopoulos, Peris, and Saha 1999; Coarfa, Druschel, and Wallach 2002; Servy 2013; Zhao et al. 2005), are usually performed "to determine the responsiveness, throughput, reliability, and/or scalability of a system under a given workload." (Meier et al. 2007, 15) The results help to identify bottlenecks, establish baselines, support tuning efforts, or can be used to determine compliance with requirements. Load and stress tests are subcategories of performance tests. The former is used to test a system under normal and anticipated production workloads, and the latter is used to test beyond those workloads. The results from a stress test are used to determine under which conditions a system will fail and how it fails (15–16).

Multiple (virtual) clients are created that load one or more URLs while responses and metrics of the server system are being collected during the execution. This is the general approach to perform performance tests and it seems fairly simple. In reality it is not that simple because server systems are very complex and indeterministic, which creates

43. https://www.trustworthyinternet.org/ssl-pulse/
44. https://www.imperialviolet.org/

a close link between the test's setup and the tested system. Hence, results are not easily comparable among different hard- and software configurations, and it is complicated to make a general statement (Nottingham 2011b).

Most load testing tools—like ab[45], Apache JMeter[46], siege[47], httperf[48], wrk[49], and so forth[50,51]—are designed to request a single URL with extremely high concurrency while collecting or allowing the collection of insightful statistics of the tested system and to perform meaningful stress tests. However, such an approach is not well suited to determine the protocol added overhead for complex entities, like web pages with their closely intertwined dependencies, especially not from a user's perspective. After all, protocols like SPDY and HTTP/2 were created to minimize the user experienced latency, and not to improve a web server's performance or response time (Grigorik 2012).

Performance testing is also offered by cloud based system, like Blitz[52] or Dynatrace[53], and there are combinations of local and cloud based systems, like LoadUI[54]. These services generally work in the same manner as the previously discussed programs. Their benefit is that the hardware for the generation of the virtual users is guaranteed and already deployed. Another benefit is the immediate generation of meaningful statistics in form of graphs instead of spreadsheets, like most of the previous programs offer it. A problem that remains with these services is that the tested server must be accessible via the Internet.

Another solution for performance tests is to record a real web browsers requests and rerun it several times; examples for such software include NeoLoad[55] and WAPT[56]. This approach goes beyond the simple "URL hammering" and/or URL randomization of previous solutions and is meant to reflect real world usage. All third-party services suffer from one or more of the same problems: proprietary software, black boxes, and very high costs. Nevertheless, all of these services are viable solutions for companies without the hardware or expertise to build their own performance tests.

45. http://httpd.apache.org/docs/trunk/en/programs/ab.html
46. http://jmeter.apache.org/
47. https://www.joedog.org/siege-home/
48. http://www.hpl.hp.com/research/linux/httperf/
49. https://github.com/wg/wrk
50. http://tsung.erlang-projects.org/
51. https://nghttp2.org/documentation/h2load.1.html
52. https://www.blitz.io/
53. http://www.dynatrace.com/
54. http://www.loadui.org/
55. http://www.neotys.com/product/overview-neoload.html
56. http://www.loadtestingtool.com/

2.8.2 Page Timing Tests

Page timing tests, as conducted by (Goldberg, Buff, and Schmitt 1998a; Kant, Iyer, and Mohapatra 2000; Coarfa, Druschel, and Wallach 2002; Welsh, Greenstein, and Piatek 2012; Padhye and Nielsen 2012; Erman et al. 2013; Wang et al. 2014; Naylor et al. 2014), are another possible approach to the problem. Statistics are collected on the client side. The results are not as closely linked to the setup and system as in the previous approach because hard- and software are not operating at their limits, as it is the case during performance tests. Additionally, it is also possible to draw inferences from client side timing information about the impact of the involved operations on the server side, but the data does not reveal where the expense exactly occurred.

The results of these tests remain stable, even if hard- and software or versions and configurations are changed on the server. Measurements on the server side, on the other hand, might be influenced by such changes. Of course, changes on the client side might affect the measurements as well. One solution for this problem could be a proxy, like the BrowserMop Proxy or Fiddler, through which every transaction is tunneled, but proxies add an additional layer of abstraction and interfere with the results per definition. The problem with proxies is further aggravated by the fact that web browsers apply an upper connection limit if they detect them (Duell et al. 2012; Smith 2012, 20; Chan et al. 2010).

Accordingly, it is unavoidable, provided that the communication should not be intervened with, that the web browser collects its own statistics; even if this means that different versions might log different times. Most web browsers offer the export of timing data in form of HTTP Archives (HARs) via their developer tools. HARs contain details of all requests the web browser had to perform in order to render the web page, and are explained in more detail in section 3.1.3. A service which supports the collection of statistics in this manner is WebPagetest.[57]

This software is developed as an open source project, and also offered for local installation via so called private instances. Agents (computers which execute the web browsers) are currently only supported on Windows, while Linux support is still experimental and only available for very few web browsers. It offers the ability to collect extended statistics and to capture screenshots and videos. The agents are actually rendering the web pages in contrast to PhantomJS, which was used by (Naylor et al. 2014), and behave exactly like a real user's web browser would do.

Actual rendering also results in additional involved subsystems, like graphic cards, and rendering processes might distort the results further because of reflows and other

57. http://www.webpagetest.org/

operations that are all highly indeterministic. Nonetheless, results reflect real world usage, and testing with different browsers can increase the statistical significance of the collected results.

2.8.3 Other Approaches

Other approaches include a theoretical examination of one or more of the involved operations as well as benchmarking isolated subsystems. While these approaches allow the optimization of parts of the complete system, they do not provide any information of a real world system. Benchmarking isolated subsystems in particular can be very trivial, while being extremely linked to the actual system and setup. For instance, to determine how many Elliptic Curve Diffie–Hellman (ECDH) operations a computer can perform a simple invocation of `openssl speed ecdh` is sufficient. After all, it would also be impossible to answer how encryption actually impacts the performance of web applications.

3 Results

The results section is dedicated to the presentation of the solution that should answer the research question of this thesis and the analysis of the collected data. The first section 3.1 sketches an overview of the exact test setup and approach that was used to perform the timing measurements. Section 3.2, where the collected data is analyzed and presented. The last section 3.3 contains the presentation of a program that can be used to securely rotate TLS session ticket keys for the nginx web server.

3.1 Setup

The chosen approach to find out which impact encryption has on web applications are page timing measurements because they provide the most information about both the client and server side at the same time. The measurements should provide information about current technologies that are in use by many web applications. WebPagetest would be the obvious choice for this task, but the lack of Linux support is a drawback because TFO is currently not implemented in the Windows kernel, but it is an interesting new extension that is worth experimenting with. The fact that it was already formally defined in RFC 7413 allows the assumption that Microsoft will implement the feature in the near future.

Additionally, the extended features of WebPagetest are not required for the evaluation, and while varying web browsers would be interesting for the measurements only Chrome currently offers support for TFO. Furthermore, both Firefox and Chrome use NSS on Linux for cryptographic operations. The difference should, therefore, be minor. Direct export of HARs is not offered by Chrome or by any other browser. A simple and automatable solution that works well in Linux is available in form of the chrome-har-capturer Node.js command-line interface (CLI) program. It utilizes Chrome's remote debugging protocol to load URLs and collect the tracing information.

The setup, therefore, consists of a client running Linux Mint[58] with the MATE Desktop,[59] due to its low resource consumption, and a server running Ubuntu Server[60] because it offers the latest kernel and software; more details can be found in table 4. Installation of the OSs was done manually; of course, Vagrant[61] could have been used, which would either

58. http://www.linuxmint.com/
59. http://www.minted.com/
60. http://www.ubuntu.com/server
61. http://www.vagrantup.com/

already contain the necessary software or automatically start the provisioning. However, the complete provisioning is automated and the only required installation is *git* to check out the repository[62] and *make* on the server to start the provisioning.

	Client	Server
Distribution	Linux Mint	Ubuntu Server
Version	17.1	15.04
Desktop	MATE	—
Architecture	64-bit	64-bit
Kernel	3.13.0-38-gen.	3.19.0-15-gen.

Table 4: Client and server OS overview.

Virtual machines (VMs) are used for both the client and the server in order to run multiple tests at the same time. One client can only perform tests sequentially because parallel execution could affect the collected data, like it would happen during a performance test. Of course, a virtualized environment has several drawbacks compared to real hardware, but negative effects should be minor because the hardware is not under high load. However, the final decision to use VMs was solely based on the time factor. Many iterations are necessary to gather statistically significant data—at least one hundred (Meier et al. 2007, 180)—which results in very long execution times. Multiple VMs allow parallel execution without interferences.

Care must be taken while choosing a desktop OS. For instance, Ubuntu Desktop[63] with its Unity environment had various issues with graphics, which might be related to the Nvidia card of the host system as well, and resource consumption during an initial test run of this setup. Chrome had to be started with disabled 3D acceleration because it still had issues even after Ubuntu reported no graphic errors. This was not the case with Linux Mint. Additionally all visual aids were disabled to minimize the impact of graphic related issues further.

A fast Microsoft Windows computer is used as host system, and VirtualBox[64] as virtualization software to run the clients and servers. Hardware details of the host can be found in table 5, and both the client and server VM configurations can be found in appendix A. Intel's Core i7-3770K CPU features four real cores plus four virtual cores via HyperThreading. Two cores with 2 GB of main memory are assigned to each client

62. The complete code currently resides in a private Bitbucket repository and will be moved to GitHub after the final exam; it is also included on the enclosed volume.
63. http://www.ubuntu.com/desktop
64. https://www.virtualbox.org/

Component	Type	Comment
Operating System	Windows 7 Professional 64-bit	
Motherboard	Gigabyte GA-Z77X-D3H	
Processor	Intel Core i7-3770K	
Main Memory	16 GB	
Graphics	Nvidia GeForce GTX 680	
Hard Drive	Samsung SSD 830	System
	2×WD1001FALS RAID-0	VMs

Table 5: Host system hardware overview.

VM and each server has a single core with 1 GB. The remaining two cores are left to the host system. Each VM has full access (no execution cap) to the CPU without additional virtualization layers, which is possible through the Intel VT-x EPT CPU extension.

The server's system specification was chosen because it reflects common virtual private server (VPS) setups, and the client was simply assigned twice as much of all resources, which was kept after verifying that it is enough for the used desktop OS to run smoothly during the tests. For comparison, the previously mentioned Ubuntu Desktop VM had access to four cores and 8 GB main memory. CPU usage on the host system was constantly going up and down between 12 % to 28 %, while main memory usage for all VirtualBox processes combined was fixed at ∼360 MB; see figure A.3 and figure A.4 for screenshots.

Network adapter two is used to connect a client with its server, network adapter one is used to connect to the Internet via the host system. An active Internet connection is only required during the initial provisioning and not for the actual tests, but it was useful during development.

The provisioning on both the client and server is started via *make client-vbox* and *make server-vbox* respectively. This will compile and install the latest PHP release[65] first, and create symbolic links for the main executable followed by the installation of the VirtualBox Guest Additions, which are required to resolve any compatibility issues. Provisioning continues after rebooting, which is necessary for full initialization of the additional VM software with the server.

3.1.1 Server

After the reboot, the command *make server* needs to be executed on the server. Kernel and already installed software are upgraded to their latest versions followed by the extraction

65. https://github.com/fleshgrinder/php-compile

of previously collected HARs of web pages that are going to be tested. An MD5 hash is generated for each web page's resources based on the path and query of their URLs. The hash is used as local filename because path and query combinations of many URLs are extremely long, many hierarchies deep, and contain problematic characters that would make it very difficult to create or delete these directories and files on a Windows system, which will be used for the data analysis. There are more reasons why this hashing approach was chosen but more on this later.

A few problematic HTTP headers are removed from the resources. Starting with the content length field, which is recomputed by the web server based on the real local size of the file. Content security policy and reporting headers are removed because they could block loading of resources or try to send reports to the real website. Public key pins and associated headers are removed because testing is done with self generated certificates and the browser would not accept them otherwise. All raw cookie headers are removed, and the already parsed cookie objects of the HAR are used, instead, because they are easier to work with in the object oriented program. HSTS headers are removed because HTTP support is required for later comparison of unencrypted with encrypted transactions. Status is removed because it is redundant and already available via a dedicated property in the HAR, which is again used by the backend application. Finally, date and vary are removed, because they are set by the web server based on the actual response it sends.

Payloads of the resources are either extracted and decoded, if present in the HAR, or separately downloaded. The loop starts over for all processed resources and their cookies, headers, and payloads are being prepared for storage by replacing all previously collected top-level domains (TLDs) with the local TLD. This is necessary to circumvent the HSTS preloading list of Chrome for certain domains, and it is easier to generate a catch-all zone file for the DNS, which will be installed after the extraction because it only has to match the TLD and is entirely domain agnostic.

Four versions of each web page are created: two are HTTP or HTTPS with sharded domains, and two are HTTP or HTTPS without sharding (they only use the domain of the web page itself). It would also be possible to create pure protocol-less URLs in all payloads, but it was found that this approach breaks some JS files. These files can contain conditions that would be altered and consequently broken by a simple string or regular expression replace approach. Finally, all resources are stored in a local directory, and the collected domains are written into a file for further usage in later steps.

The stored resources consist of two or three files. The first is a serialized PHP file containing an object with the resource's cookies, headers, redirect URL (if applicable),

```
34 location / {
35     # The PHP front controller script will redirect nginx to serve the file.
36     location ~ \.payload$ {
37         internal;
38         try_files $uri =404;
39     }
40 
41     # Pass any request to the PHP script and let it determine what is to be returned.
42     include fastcgi.ngx;
43 }
```

Listing 10: Nginx catch-all location block.

and the HTTP status code. The second file is the payload itself, while the third file is the compressed payload if the original resource was compressed. Precompressing the payloads avoids compression during runtime, which would add additional processing time. All of this abstraction is necessary to create an nginx configuration containing only a single location block that is reusable for any web page; see listing 10. Nginx forwards the requests to a PHP script that computes the MD5 hash, deserializes the resource's meta information, sets the cookies and HTTP headers, and returns the path to the payload file back to nginx for delivery.

Script based and purely static web pages are also supported instead of the explained HAR based extraction process, which is provided for addition of new test cases based on existing web pages. It is important to note that the creation of a web page's HAR is actually not as easy as it might seem, especially if the web page is responsive and/or contains a lot of dynamic content. The web browser, which is used, might not request some of those files at this point, for instance a banner which is generated based on the viewport, but those missing files will result in a 404 error during the test runs. Manual checking for errors is, therefore, necessary before a new test case can be used.

The DNS server bind9 is installed and configured in the next step. It was explained in section 2.3.3 that every domain look-up consumes some time, which should be controllable during the tests as well. This means that the bind9 on the server acts as primary DNS for the client instead of using the *hosts* file on the client for the local TLD look-ups, which would have virtually no impact. Linux OSs do not cache DNS information by default and require additional packages to do so. Of course, previously mentioned HSTS preloading reasons apply too but could have been solved via the *hosts* approach as well. Nginx is compiled from source to get the latest updates of the experimental SPDY module; see

```
nginx version: nginx/1.9.0
built by gcc 4.9.2 (Ubuntu 4.9.2-10ubuntu13)
built with OpenSSL 1.0.2b-dev xx XXX xxxx
TLS SNI support enabled
configure arguments: ... --with-cc-opt='-O2 -pipe -m64 -march=native'
  ↪ --with-http_gzip_static_module --with-http_ssl_module --with-http_spdy_module
  ↪ --with-openssl-opt='-O2 -pipe -m64 -march=native -DOPENSSL_NO_HEARTBEATS
  ↪  enable-ec_nistp_64_gcc_128' --with-openssl=/usr/local/src/openssl
  ↪ --with-md5=/usr/local/src/openssl --with-md5-asm
  ↪ --with-sha1=/usr/local/src/openssl --with-sha1-asm --with-pcre=/usr/local/src/pcre
  ↪ --with-pcre-jit --with-zlib=/usr/local/src/zlib --without-http_access_module
  ↪ --without-http_auth_basic_module --without-http_autoindex_module
  ↪ --without-http_empty_gif_module --without-http_geo_module
  ↪ --without-http_memcached_module --without-http_proxy_module
  ↪ --without-http_referer_module --without-http_scgi_module
  ↪ --without-http_split_clients_module --without-http_ssi_module
  ↪ --without-http_upstream_ip_hash_module --without-http_userid_module
  ↪ --without-http_uwsgi_module
```

Listing 11: Extended nginx version information with highlighted OpenSSL configuration flags (irrelevant parts omitted for brevity).

listing 11 for detailed *configure* flags.[66,67]

Generation of two CAs with the system's OpenSSL follows after the configuration of the network with a fixed IP address: one for ECDSA with a secp384r1 EC plus SHA256 for hashing, and one for RSA with 4,096 bit plus SHA348 for hashing. Furthermore, two intermediate CAs are created for each root, the ECDSA uses a secp256r1 EC, RSA 2,048 bit, and both use SHA256 for hashing. Root certificates need to be created with stronger key sizes than other certificates because they are usually very long-lived, which means up to thirty years. Intermediate certificates are created with best practice values (Ristić 2014, 248–49, 252–53); see listing 12 for involved commands.

Generation of the nginx server configurations follows because they require the certificates of the root and intermediate CAs to be in place for the generation of the host certificates and keys. A server configuration for each web page, protocol, and sharding type is created. This means that there are ten configurations in total:

1. HTTP with sharding.
2. HTTP without sharding.
3. HTTPS with sharding and ECDSA certificate.
4. HTTPS without sharding and ECDSA certificate.
5. HTTPS with sharding and RSA certificate.

66. https://github.com/fleshgrinder/nginx
67. https://github.com/fleshgrinder/nginx-compile

```
# Generate ECDSA CA
cd ecdsa/ca
mkdir -p certs db private
touch db/index
openssl rand -hex -out 'db/serial' 16
echo 1001 > db/crlnumber
openssl rand -out 'private/.rnd' 32768
openssl ecparam -genkey -name secp384r1 -out 'private/key'
openssl req -out 'csr' -new -rand 'private/.rnd' -key 'private/key' -config 'ca.conf'
 ↪  -batch
openssl ca -config 'ca.conf' -in 'csr' -out 'crt' -selfsign -notext -batch -extensions
 ↪  ca_ext

# Generate ECDSA intermediate CA
cd ecdsa/sca
...
openssl ecparam -genkey -name prime256v1 -out 'private/key' -rand 'private/.rnd'
openssl req -out 'csr' -new -rand 'private/.rnd' -key 'private/key' -config 'sca.conf'
 ↪  -batch
openssl ca -config '../ca/ca.conf' -in 'csr' -out 'crt' -notext -batch -extensions
 ↪  sca_ext

# Generate RSA CA
cd rsa/ca
...
openssl req -out 'csr' -new -rand 'private/.rnd' -newkey rsa:4096 -sha384 -keyout
 ↪  'private/key' -nodes -config 'ca.conf' -batch
openssl ca -config 'ca.conf' -in 'csr' -out 'crt' -selfsign -notext -batch -extensions
 ↪  ca_ext

# Generate RSA intermediate CA
cd rsa/sca
...
openssl req -out 'csr' -new -rand 'private/.rnd' -newkey rsa:2048 -sha256 -keyout
 ↪  'private/key' -nodes -config 'sca.conf' -batch
openssl ca -config '../ca/ca.conf' -in 'csr' -out 'crt' -notext -batch -extensions
 ↪  sca_ext
```

Listing 12: OpenSSL certificate generation commands. The dots indicate navigation and the initial repeated commands for the creation of the necessary files. The used configuration files are available in the repository or in the enclosed volume and not included in the work because of their length.

6. HTTPS without sharding and RSA certificate.
7. SPDY with sharding and ECDSA certificate.
8. SPDY without sharding and ECDSA certificate.
9. SPDY with sharding and RSA certificate.
10. SPDY without sharding and RSA certificate.

HTTP/2 is not yet supported by nginx, but the release was announced for the end of 2015 (Garrett 2015). A few alternatives are available, for instance H2O[68] and nghttp2,[69] but both SPDY and HTTP/2 open only a single TCP connection. This means in effect that the SPDY results can be conferred to HTTP/2. The keep alive times of 10 s and TLS settings with the `ECDHE-RSA-AES128-GCM-SHA256` and `ECDHE-ECDSA-AES128-GCM-SHA256` cipher combination—which offer PFS for False Start and are the two faster of four ciphers recommended by RFC 7525 (Sheffer, Holz, and Saint-Andre 2015, 11)—plus a shared session cache are the same for all configurations; see listing 13 for the relevant parts of nginx's main configuration that was used for all tests.

Software	Version	Origin
bind	1:9.9.5.dfsg-9	Package
nginx	1.9.0	GitHub
nginx's OpenSSL	1.0.2b-dev	GitHub
nginx's zlib	1.2.8	GitHub
OpenSSL	1.0.1f-1ubuntu11	Package
PHP	5.6.7	GitHub

Table 6: Exact server software version information.

Provisioning of the server is finished with all configurations and services in place. It will start a remote service that accepts commands from the client, for instance to download the root certificates. Detailed version information for reference is listed in table 6.

3.1.2 Client

Client provisioning continues with the invocation of *make client* after the server is ready, and the necessary reboot for the VM Guest Additions was performed. This will upgrade all packages and the kernel to the latest version, and continue with the installation of the current stable Google Chrome followed by the NSS tools. Previously generated root certificates are imported into the certificate database of the current user with the

68. https://h2o.examp1e.net/
69. https://nghttp2.org/

```
35      aio                             off; # Does not work properly in VM.
36      etag                            off;
37      fastcgi_cache_path              /tmp/ngx-fastcgi-cache levels=1:2
         ↪  keys_zone=FASTCGI_CACHE:64m inactive=10m;
38      gzip                            off;
39      gzip_proxied                    any;
40      gzip_static                     on;
41      gzip_vary                       on;
42      if_modified_since               off;
43      ignore_invalid_headers          on;
44      msie_padding                    off;
45      output_buffers                  1 512k; # Should match read ahead value.
46      postpone_output                 1460;
47      read_ahead                      512k; # Should match output buffers.
48      reset_timedout_connection       on;
49      sendfile                        off; # Does not work properly in VM.
50      tcp_nodelay                     on;
51      tcp_nopush                      on;
52
53      client_body_timeout             10s;
54      client_header_timeout           10s;
55      keepalive_disable               none;
56      keepalive_timeout               10s;
57      lingering_time                  10s;
58      lingering_timeout               10s;
59
60      spdy_headers_comp               0; # To slow in such a fast environment.
61      spdy_keepalive_timeout          10s;
62      spdy_recv_timeout               10s;
63
64      ssl_buffer_size                 1360;
65      ssl_ciphers                     ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-
         ↪  SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384:DHE-RSA-
         ↪  AES128-GCM-SHA256:DHE-DSS-AES128-GCM-SHA256:kEDH+AESGCM:ECDHE-RSA-
         ↪  AES128-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-
         ↪  AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-
         ↪  AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-
         ↪  AES128-SHA:DHE-DSS-AES128-SHA256:DHE-RSA-AES256-SHA256:DHE-DSS-AES256-SHA:DHE-
         ↪  RSA-AES256-SHA:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!3DES:!MD5:!PSK;
66      ssl_dhparam                     dhparam-2048.pem;
67      ssl_ecdh_curve                  secp384r1; # prime256v1
68      ssl_prefer_server_ciphers       on;
69      ssl_protocols                   TLSv1.2;
70      ssl_session_cache               shared:SSL_SESSION_CACHE:64m;
71      ssl_session_tickets             off;
72      ssl_session_timeout             10s;
73
```

Listing 13: Relevant parts of the nginx main configuration.

NSS tools, which will be used by Chrome to establish trust and validate the provided certificate chain. Node.js and chrome-har-capturer are installed followed by the network configuration with a fixed IP address. Provisioning of the client is finished, and the tests are automatically started for all web pages that reside in the appropriate directory; detailed version information of the software can be found in table 7.

Software	Version	Origin
Google Chrome	42.0.2311.135 (64-bit)	Package
iproute2	3.16.0-2ubuntu1	Package
libnss3-tools	2:3.17.4-0ubuntu1	Package
PHP	5.6.7	GitHub

Table 7: Exact client software version information.

It was discovered, after the first week of tests[70] and during the analysis of the collected data that chrome-har-capturer (0.4.1, but 0.5.0 does not address the issues) logs incorrect timing information.[71] Listing 14 is a unified diff that illustrates the significant deviation of the reported timing information compared to a HAR that was manually exported with Chrome. The program for fully automated tests was already finished, and a transition to WebPagetest meant more effort than developing a drop-in solution that would export the HARs via Chrome for the Node.js program.

It was mentioned earlier that Chrome does not offer the ability of a direct HAR export, but the *chrome.devtools.network* API for browser extensions offers a method to retrieve a complete archive. An extension that utilizes this API is only invoked if the developer tools are opened by a user; there are no CLI flags or any other means provided by Chrome for this. The Linux automation tool *xdotool* was used to overcome this hurdle. It waits for the web browser to start and sends the *F12* keyboard shortcut to open the developer tools, like a real user would do, at which point the extension is being executed, the page requested, and finally the HAR is downloaded to the local file system. A download is necessary because an extension is not allowed to access the local file system.

This workaround introduced some minor bugs. For instance, *xdotool* sometimes sends the keyboard shortcut before Chrome is fully initialized. Chrome simply ignores the signal in that case, subsequently the developer tools are not opened and the extension is not executed. A process timeout was, therefore, defined that terminates Chrome and restarts

70. Which were executed with confidence, since chrome-har-capturer has considerably high usage statistics for such a specialized software and is recommended in forums and mailing lists.

71. Reported issue: https://github.com/cyrus-and/chrome-har-capturer/issues/19

```
--- chrome-capturer.har        Wed May 27 10:25:13 2015
+++ chrome.har        Wed May 27 10:25:07 2015
@@ -1,21 +1,21 @@
 ...
-"time": 243.0489062498805,
+"time": 64.56995010375977,
 "request": {
     "method": "GET",
     "url": "[…]",
-    "httpVersion": "HTTP/1.1",
+    "httpVersion": "unknown",
 ...
 "response": {
     "status": 200,
     "statusText": "OK",
-    "httpVersion": "HTTP/1.1",
+    "httpVersion": "unknown",
 ...
 "timings": {
-    "blocked": -1,
-    "dns": 0.654000000054114,
-    "connect": 53.30499999990936,
-    "send": 1.0790000000042994,
-    "wait": 112.32899999993191,
-    "receive": 36.00390625,
-    "ssl": 39.677999999980806
+    "blocked": 0.974000000041997,
+    "dns": -1,
+    "connect": -1,
+    "send": 0.570000000038813,
+    "wait": 30.76099999998409,
+    "receive": 32.264950103694865,
+    "ssl": -1
 ...
```

Listing 14: Unified diff between a HAR entry exported with chrome-har-capturer and Chrome (diff input was reduced to the relevant parts for brevity).

the test. Another issue that was found during the analysis is that the exported HARs are missing the *startedDateTime* property, which is a prerequisite to recreate the exact start time of each request at a later point; but the values are actually not required for the calculations, and their absence is, therefore, not an issue.

Final tests with the new extension were executed with three test cases:

1. The Facebook login page because it is one of the most visited websites of the world according to the Alexa Internet top sites.
2. A synthetic test of a single HTML document referencing all 256 famfamfam flag icons plus a sprited version of them.[72]
3. The most visited Wikipedia article of 2013, which was, like coincidence wants, Facebook.[73]

Connection	Upload kbit/s	Download kbit/s	Delay ms
Fast 3G	1,638.4	768	150
DSL	1,536	384	50
Cable	5,120	1,024	28
FTTH	20,480	5,120	4

Table 8: Reference of used connection technologies and their attributes.

A HAR with payloads was exported via Chrome for Facebook and Wikipedia, and the synthetic test is generated via a script. The tests were conducted as outlined in listing 15. Connections is an array, where each inner array consists of download and upload bandwidth as well as the delay. Each connection represents a common network technology, which are offered by WebPagetest as well, which is the source of the values. An explanation of the connection values can be found in table 8; the last infinity value means that no traffic control was applied and the full connection speed of the virtual network was used. The constraints are applied with the Linux program *tc* from the *iproute2* package.

Each page was loaded twice during a single test run, see line thirty-one to thirty-five in listing 15. The exact flow for a single test run is very simple. Chrome is started with a new profile, which means in effect that all caches are empty, connects to the server and loads the page, the HAR is exported. Opened TCP connections are closed, see listing 16, and the page is loaded again; no other caches are purged. TFO is only active during the repeated view, because the web browser must first connect to the server to create the

72. https://github.com/tkrotoff/famfamfam_flags
73. https://tools.wmflabs.org/wikitrends/2013.html

```
func benchmark {
    for each i in [1..2] {
        for each tfo in (true,false) {
            for each domain in ("facebook","flags","wikipedia") {
                for each protocol in ("http","https","spdy") {
                    if protocol is "http" {
                        benchmark-advance()
                    } else {
                        for each algorithm in ("ecdsa","rsa") {
                            benchmark-advance()
                        }
                    }
                }
            }
        }
    }
}

func benchmark-advance {
    connections := (
        (1638.4,768,150),
        (1536,384,50),
        (5120,1024,28),
        (20480,5120,4),
        INFINITY
    )

    for each sharded in (true,false) {
        for each connection in connections {
            for each i in [1..200] {
                start-chrome()
                request-url-and-capture-http-archive() // first view
                close-connections()
                request-url-and-capture-http-archive() // repeated view
                stop-chrome()
            }
        }
    }
}
```

Listing 15: Pseudocode illustrating the test flow with actually used settings.

cryptographic cookie that is required for future authentication (Cheng et al. 2014, 7–10).

```
114                     cache = true;
115                     console.info('First run complete, reloading tab for second
                         ↪  run.');
116                     // http://gent.ilcore.com/2011/02/chromes-10-caches.html
117                     chrome.benchmarking.closeConnections();
118                     console.info('Closed TCP connections for second run.');
```

Listing 16: Relevant code from the Chrome extension that closes TCP connections between two page loads.

Two hundred iterations for each test case are necessary to ensure statistical significance and to compensate errors in any subsystem (Meier et al. 2007, 180). The outermost loop of two iterations was added to generate a control data set for each benchmark to validate the collected data from the initial run. Many additional, shorter tests were executed to find bugs and to try various configurations. All-in-all, almost one hundred gigabytes of uncompressed data were generated in a period of more than two weeks of pure runtime.

3.1.3 HTTP Archives

HTTP Archive (HAR) is a file format based on JavaScript Object Notation (JSON) that defines a data structure for logging of web browser request and response tracing information. The format's specification is maintained by the Web Performance Work Group of the W3C, and currently only available as a draft, thus, it is still evolving (Odvarko, Jain, and Davies 2012).[74] Native support is offered only by Chrome, but extensions for other web browsers exist as well as several third-party tools. Most web developers can probably imagine what a HAR file contains because it is a representation of the network timeline that is available via the developer tools of every modern web browser (Dyke 2015).

Listing 17 contains a brief overview of the properties that are generally most relevant for performance evaluations. The *pages* property contains all captured web pages, and each page a *pageTimings* property that contains two total times. Both *onContentLoad* and *onLoad* are representations of their JS event equivalents, hence, both values are arbitrary web browser specific numbers that cannot be compared or used for calculations because of the nature of these events (Welsh 2011).

The *entries* property is an array of all URLs that had to be requested in order to render the web page, which includes data URIs as well (inline resources: *base64* encoded

74. Also available at http://www.softwareishard.com/blog/har-12-spec/ with better formatting and navigation.

```
{
    "log": {
        "version": "1.2",
        "creater": {...},
        "browser": {...},
        "pages": [
            {
                ...
                "title": "http://example.com/",
                "pageTimings": {
                    "onContentLoad": 1720,
                    "onLoad": 2500
                }
            }
        ],
        "entries": [
            {
                "startedDateTime": "2015-05-27T07:38:41.025Z",
                ...
                "time": 50,
                "request": {...},
                "response": {...},
                "cache": {...},
                "timings": {
                    "blocked": 0,
                    "dns": -1,
                    "connect": 15,
                    "send": 20,
                    "wait": 38,
                    "receive": 12,
                    "ssl": -1,
                },
                ...
            }
        ]
    }
}
```

Listing 17: HAR format (several properties omitted for brevity).

binary data that is referenced within another resource, like in a HTML document). The provided *timings* object is the most relevant for this thesis; the following list is taken from the specification and explains the intend of each of its properties:

- *blocked – Time spent in a queue waiting for a network connection.*
- *dns – DNS resolution time. The time required to resolve a host name.*
- *connect – Time required to create TCP connection.*
- *send – Time required to send HTTP request to the server.*
- *wait – Waiting for a response from the server.*
- *receive – Time required to read entire response from the server (or cache).*
- *ssl – Time required for SSL/TLS negotiation. If this field is defined then the time is also included in the connect field (to ensure backward compatibility with HAR 1.1).*

— (Odvarko, Jain, and Davies 2012)

Each entry in the *entries* array additionally contains a *time* property that equals the sum of all of these timings. It is important to emphasize that the *connect* time already contains the *ssl* time (if applicable) and must be taken care of during all calculations. The last property to understand is *startedDateTime* that contains the exact time a request for an URL started, and is required for the exact recreation of a complete page load; more about this later. This brief overview of the format is sufficient to understand the upcoming sections, where reference to the terminology of entries and timings is used. For the sake of completeness, a HAR contains detailed HTTP headers both in raw and parsed formats.

3.2 Analysis

The collected HARs are aggregated and exported to comma-separated values (CSV) files because one test run—protocol + key exchange algorithm + sharding + bandwidth-delay + first or repeated view—results in two hundred single HARs. This gives two thousand HARs for a bandwidth-delay group and twenty thousand for a complete benchmark. Additionally, there is no statistics software available that supports the HAR format, but conversion is very simple. Each entry corresponds to one line within the CSV; see listing 18 for an extract.[75] The analysis of the aggregated CSV files is done with a combination of IBM SPSS Statistics (SPSS)[76] and Highcharts[77], supplementing the provisioning and benchmark program.

75. All CSV files are included on the enclosed volume.
76. http://www-01.ibm.com/software/analytics/spss/products/statistics/
77. http://www.highcharts.com/

```
1 group,id,protocol,algorithm,sharding,iteration,on_content_load,on_load,url,time,
   ↪ blocked,open,dns,dns_reused,connect,connection_reused,send,wait,receive,
   ↪ ssl,ssl_reused
2 'no-cache',1802,'http','','not-sharded',1,127.14004516602,1104.2101383209,
   ↪ 'http://flags.thesis/',6.3900947570801,0.55599999996048,0,0.5089999995107,0,
   ↪ 0.29500000073313,0,0.07000000005064,3.5849999994753,1.3750947573499,0,0
3 'no-cache',1802,'http','','not-sharded',1,127.14004516602,1104.2101383209,
   ↪ 'http://flags.thesis/cx.png',3.4000873565674,0.74099999983446,0,0,1,0,1,
   ↪ 0.090999999883934,1.6169999998965,0.95108735695248,0,0
4 'no-cache',1802,'http','','not-sharded',1,127.14004516602,1104.2101383209,
   ↪ 'http://flags.thesis/je.png',3.3500194549561,0.74899999981426,0,0.12700000024779,
   ↪ 0,0.64500000007683,0,0.06899999971211,1.1789999998655,0.58101945523958,0,0
```

Listing 18: Aggregated CSV extract.

3.2.1 Convergence

The first analysis consists of convergence plots of the average and median to determine if the number of iterations/samples is actually enough (Greenberg 2015, time index 17:45). One of the aggregated time properties is predestined for this plot, but it was already mentioned that both *pageTimings* are not to be trusted. It is not clear what time they actually represent, and it would make comparison with other browsers complicated or even impossible. The *time* property of each entry was, therefore, the best available metric for an illustration of the convergence. Of course, the value has to be aggregated for all requests of each entry. Using the median of the aggregated *time* instead of the average would not account for the actual time of a complete test run because resources have different sizes.

The *receive* property's time is, therefore, always greater for bigger files, thus, the median would not correctly represent those files. See listing A.1 and listing A.2 for the relevant SPSS Syntax that was used to load the aggregated CSV files and calculate the average time of the split data set; see line sixty for the self-explanatory split variables. Further, a PHP script was used to calculate the average and median of already seen samples while iteratively going through the file, which created a new CSV file; see listing A.3.

However, anomalies in the data were spotted before the first plot was created. The *time* of some entries was negative, even though the specification states that the property is the "sum of all timings available in the timings object (i.e. not including -1 values)" (Odvarko, Jain, and Davies 2012, entries), and "*send*, *wait* and *receive* timings are not optional and must have non-negative values" (timings). In other words, it is impossible that *time* is negative. It is unclear why Chrome is logging negative values for these resources despite the fact that *send*, *wait* and *receive* contain positive numbers. A new sum has to be calculated based on the other properties; see first line in listing A.1.

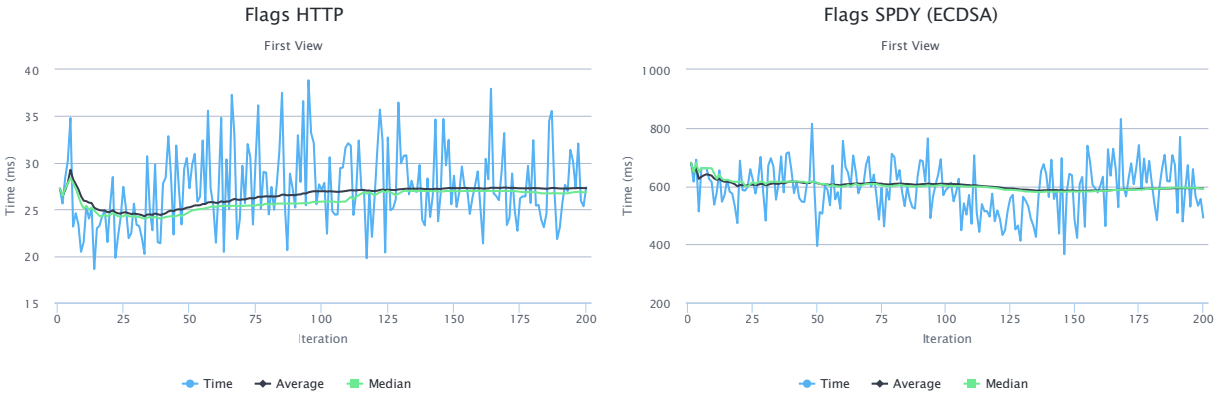


Figure 16: Convergence plots from the first view of the synthetic flags data set.

An investigation of the other properties for illegal values with SPSS revealed that the *onLoad* time is sporadically missing from some samples. No more illegal values were encountered, and the first series of plots could be created after applying the patch. Figure 16 shows two plots from the synthetic flags data set: the left side shows the convergence plot for the HTTP tests and the one on the right is SPDY with ECDSA key exchange, both not sharded. The blue line represents the average time (y-axis) of the sample from a specific iteration (x-axis), while the black line is the total average, and the green line is the median of all average times that were plotted so far.

Protocol	Algorithm	Sharded	Minimum (ms)	Maximum (ms)	Average (ms)
HTTP	—	No	19.35	51.00	30.01
		Yes	30.09	125.99	43.55
HTTPS	ECDSA	No	25.60	47.69	34.59
		Yes	41.11	72.79	56.88
	RSA	No	21.09	42.80	30.41
		Yes	40.48	73.84	54.62
SPDY	ECDSA	No	457.07	853.17	645.91
		Yes	366.62	653.33	472.65
	RSA	No	420.67	700.25	543.59
		Yes	306.86	685.01	466.81

Table 9: Detailed timings of the flag data set.

However, the actual convergence is not really interesting in these plots from the first flags benchmark, it is the y-axis scale from the SPDY result, which is interesting. The HTTP and HTTPS samples range from 19.35 ms to 125.99 ms, while the SPDY range

is 420.67 ms to 853.17 ms; see table 9 for detailed report. A circumstance that was only seen in the synthetic flags test, where over two hundred images were simply embedded in a document. A closer look at the logged times revealed that the extremely high values, which are not comparable to any other data set, are the result from a *block* and *send* combination of the HAR's entries. The problem is that Chrome requests all images at the same time.

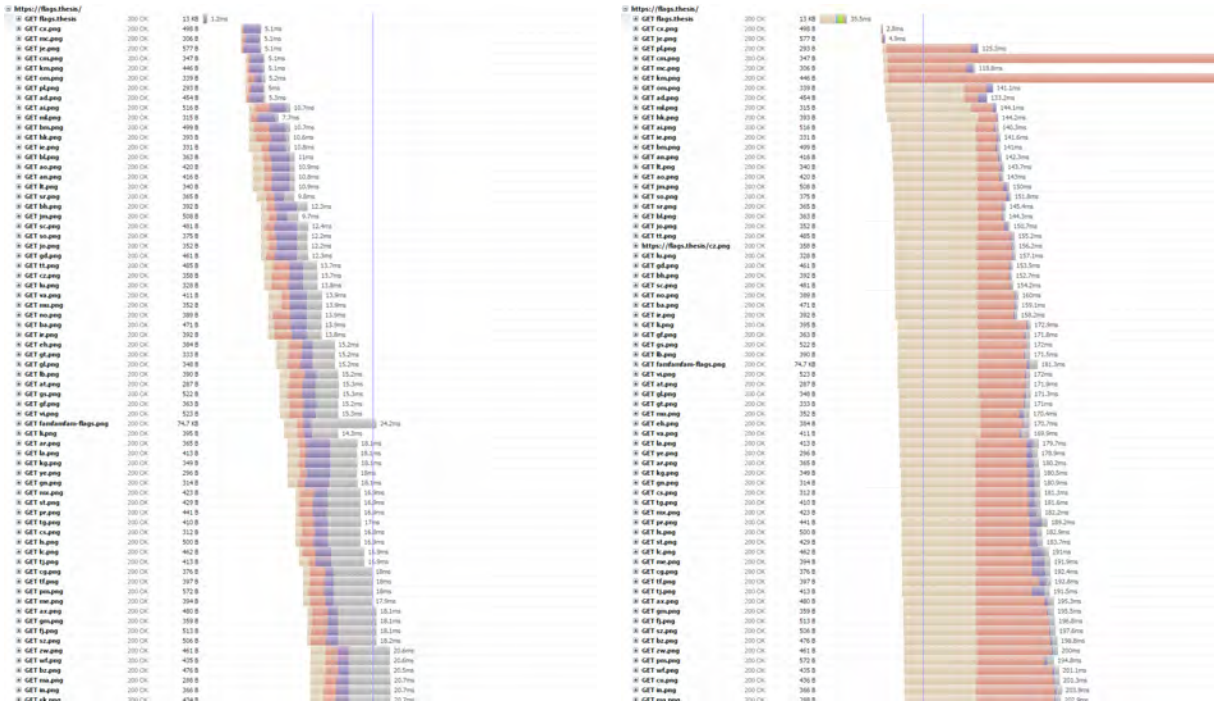
This is best visualized with a comparison: see figure 17, where the left side shows the timeline of a Window's Chrome (43.0.2357.81 m 64-bit) requesting the same page from the same server as before, which is running in the VM. The right side is from the benchmark VM's Linux Chrome with a manually exported HAR that contains the *startedDateTime*. The HARs that were exported during the benchmarks from the Linux Chrome miss this property, and it would not be possible to visualize the exact time that a request was started.

The Windows PLT is 131.738 ms, while the Linux PLT is again extremely high with 655.17 ms; no traffic control is in effect. The problem is that Chrome is requesting more or less all images at the same time, which is simply too much parallelism for the network. Whereas the Windows pendant is requesting the images in batches, which is even more apparent in the bottom of the visualized traces in figure 18 (Chan 2013b, 2013c, 2012). The flags dataset will, therefore, be omitted from the rest of the analyses.

Plots from the Facebook and Wikipedia data set show that the convergence of both the average and median is good. The indicator of a good convergence is the stabilization of their respective lines in the plots. These plots already reveal a lot of other information about the benchmarks. For instance, a high fluctuation in the timings (blue lines); interestingly, the RSA plots show less variance than the ECDSA plots. It is also apparent that encryption has a negative effect on the timings. See figure 19 for all convergence plots of the first Wikipedia benchmark without bandwidth or delay restrictions.[79]

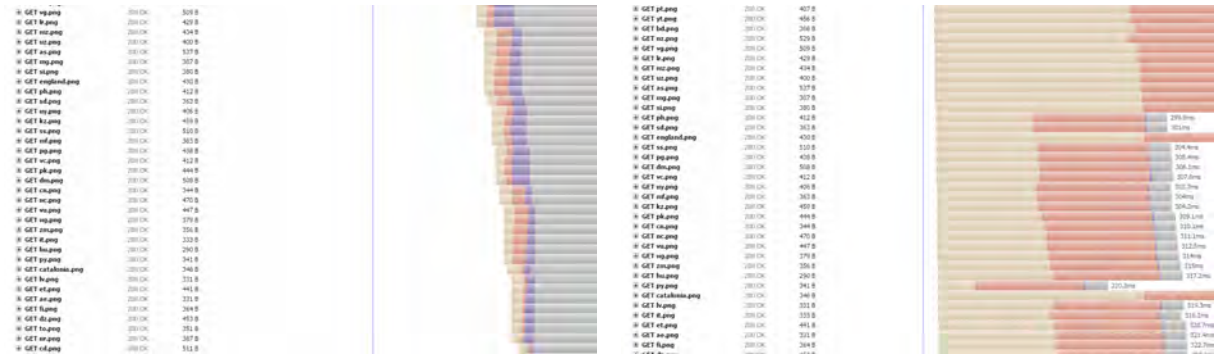
78. Created with: http://www.softwareishard.com/har/viewer/

79. All plots are available on the enclosed volume and will be published after the final exam, most likely as well on GitHub.



(a) Windows (b) Linux

Figure 17: Comparison of Windows and Linux loading the synthetic flags test page.[78]



(a) Windows (b) Linux

Figure 18: Lower end of the requests for the flags test page.[78]

Figure 19: Convergence plots from first view.

3.2.2 Outliers

The timings, as seen in the previous plots, contain various peaks, which are most likely outliers. Usually, they are removed from the collected samples because they are atypical and infrequent observations that might be due to measuring errors or other anomalies. They could be easily factored out if a normal distribution is present, which is often the case in response times (Meier et al. 2007, 179). Another prerequisite is that the number of samples is statistically significant, which it is in this case with two hundred samples per benchmark. The last criterion is that the same anomalies are not found in other tests.

Some samples meet all of the prerequisite, while some do not. The collected HTTP times are mainly normally distributed, while the HTTPS times are slightly inferior but the SPDY timings are not. The reason could be due to previously observed parallelism and contention, or bugs in the experimental nginx module.[80] See figure 20 for two random SPDY samples. For the mentioned reasons, no outliers were removed for the analysis since it is assumed that they correspond to real world problems of available software, even if they could have been easily removed via z scores (Luko 2011).

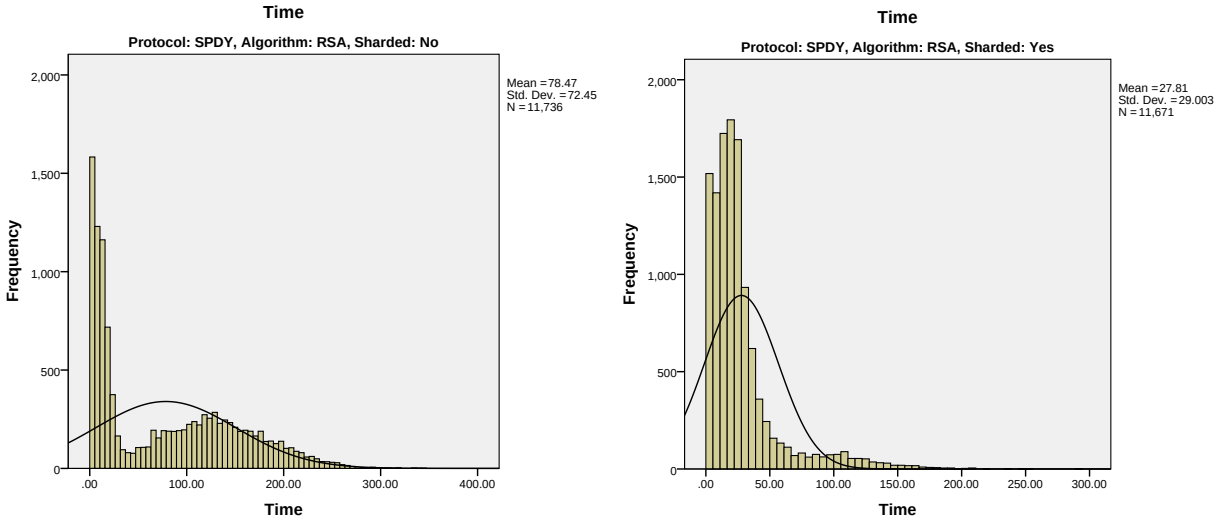


Figure 20: Random sample histograms showing the normal distribution of the timings.

3.2.3 TLS Impact

The data sets have to be aggregated further in order to determine the actual impact of encryption. Various other analyses were performed on the data sets, for instance descriptives and histogram plots, but discussing everything is beyond the scope of this thesis.

80. For instance: http://trac.nginx.org/nginx/ticket/714

Additionally, this information is not essential because web developers and administrators usually do not have a strong background in statistics. However, further aggregation of the data sets is performed through determination of the median from the *time* and all its components (*block*, *dns*, *connect*, *send*, *wait*, *receive*, and *ssl*) from all samples.

View	Protocol	Not Sharded (%)	Sharded (%)
First	HTTP	90	79
	HTTPS	91	80
	SPDY	98	97
Repeated TFO	HTTP	90	82
	HTTPS	90	83
	SPDY	98	96

Table 10: Reuse of DNS information and connections per protocol.

In other words this is the median of the black average line from the convergence plots. Using the green median line would result in missing out information since the combination of *dns* and *connect* (plus its subcomponent *ssl*) are reused for many requests; the median is, therefore, always zero. The actual reuse count depends on the protocol and the question whether the web page's resources are sharded or not; see table 10 for the exact percentages and listing A.4 for the relevant SPSS Syntax. SPDY with its single TCP connection to each domain exhibits the greatest reuse, as expected. As a consequence, only a single TLS negotiation phase has to be completed between both peers with SPDY even if no session caching is active.

The reuse percentages are the same for both key exchange algorithms, and only change slightly with varying bandwidths and delays. This analysis revealed an error that was somehow overlooked in all previous tests that were executed for this thesis. The Facebook data set's sharded web page loaded substantially less resources than its un-sharded counterpart. This means that the results from this data set are unusable because it is not possible to compare the two with each other. Inclusion of the data would most certainly yield confusion, which is why all plots and tables contain only data from the Wikipedia data set.

Nonetheless, figure 21 shows a breakdown of the various aggregated timings of the first view, and figure 22 shows the repeated TFO view of the Wikipedia benchmarks; relevant SPSS syntax can be found at listing A.5. The plots show stacked charts of all time components. Most apparent is the low performance of SPDY if sharding is not in use. This is in strong contrast to the expectation based on its mode of operation. The

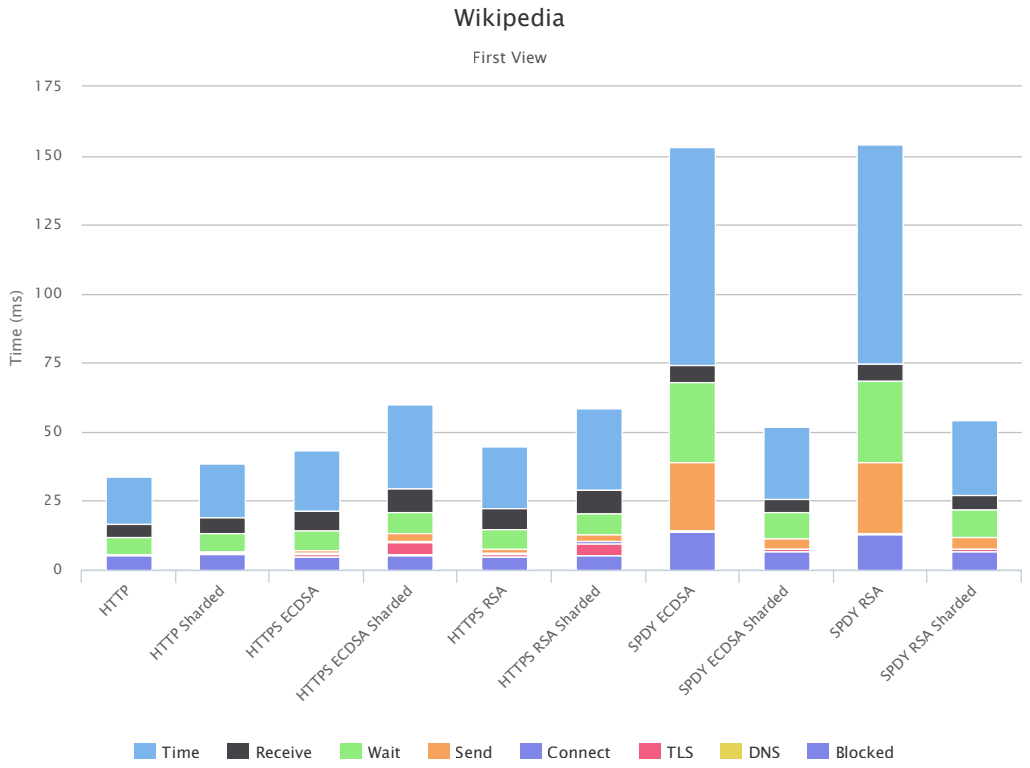


Figure 21: Breakdown of the first view timings.

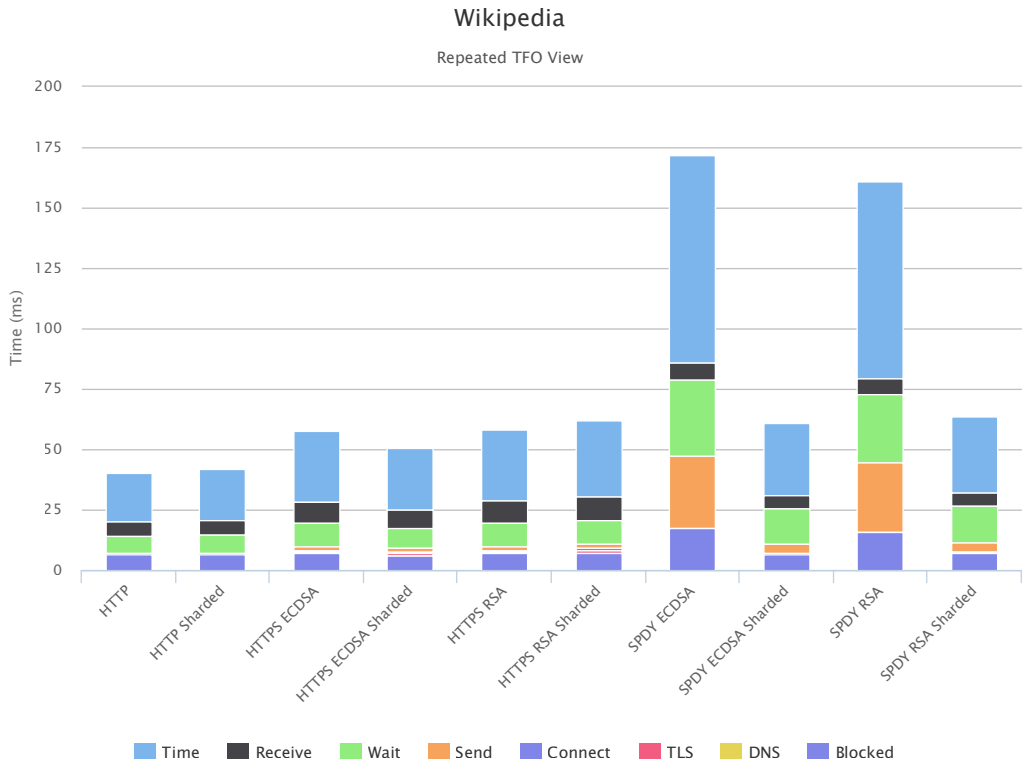


Figure 22: Breakdown of the repeated TFO view timings.

increased times are likely to occur again because the browser requests too many resources in parallel and induces contention; like it was the case in the flags benchmark. Evidence supporting this is the substantial surge in the *send* time (orange areas). Additionally, all four SPDY areas show more *blocked* time than the other protocols.

Something that is also apparent in the plots is that the negotiation phase of TLS occupies a substantial amount of time in the HTTPS samples, is less apparent in SPDY, and, of course, not part of HTTP at all. The overall percentage that the TLS negotiation phase requires is easily computed. Of course, the impact of the handshake depends on the total amount of resources on the page. Only a few, small resources result in a greater impact because the receive times are very low in contrast to long transactions. Table 11 contains the calculated percentages for the aggregated plots, where SPDY is able to reduce the impact again by utilizing only a single TCP connection.

Protocol	Algorithm	Sharded	TLS (%)
HTTPS	ECDSA	No	3.78
		Yes	14.24
	RSA	No	3.37
		Yes	14.17
SPDY	ECDSA	No	0.62
		Yes	3.29
	RSA	No	0.42
		Yes	3.08

(a) First view.

Protocol	Algorithm	Sharded	TLS (%)
HTTPS	ECDSA	No	1.96
		Yes	3.48
	RSA	No	1.85
		Yes	3.37
SPDY	ECDSA	No	0.03
		Yes	0.35
	RSA	No	0.03
		Yes	0.34

(b) Repeated view.

Protocol	Algorithm	Sharded	TLS (%)
HTTPS	ECDSA	No	1.67
		Yes	3.59
	RSA	No	1.86
		Yes	3.32
SPDY	ECDSA	No	0.03
		Yes	0.39
	RSA	No	0.03
		Yes	0.38

(c) Repeated TFO view.

Table 11: TLS negotiation phase impact of total *time*.

These numbers also confirm that the performance of ECDSA cannot reach that of RSA, but provides better security. Additionally, the smaller certificates should offer more room

for payload, which is interesting with newer protocols like TLS 1.3 or the TLS False Start extension. It does not look as if TFO would exert an influence on the negotiation phase, no matter what the bandwidths and delays were. The aggregated plots provide a good overview of the overall performance of each protocol and the impact of the TLS negotiation phase, but this only accounts for the key exchange and asymmetric cryptography. The total time determines the actual impact of encryption since everything has to be encrypted before it can be sent.

Consequently, the *receive*, *send*, and *ssl* times need to be compared to the total average *time* of all samples, and subsequently subtracted from the same values that were calculated for HTTP; see table 12 for the results. The un-sharded SPDY results are again complicated to classify because of the same problems as previously mentioned, but the other numbers are stable and show a clear result of ∼11.5 % for the actual impact of TLS on the total *time*. SPDY is definitely going to lower that value, as can be seen from the sharded percentages in the table, but this requires that the parallelism and contention, in other words, its prioritization feature, actually works.

The results of the tests with different bandwidths and delays result in a comparable percentage for the overhead of TLS, ranging from 5 to 15 % and SPDY always shows the best results. This concludes the research question of this thesis that was stated as follows: *Does it hold true that new, optimized algorithms and their implementations as well as faster computers reduce the overhead of encryption to such a great extent that there is hardly a difference to unencrypted communication?* The author comes to the conclusion that this is actually the case, although implementations have their pitfalls and proper optimization for best performance is key. The findings and conclusion are further discussed in section 4.

Of course, these numbers only apply to the actual low-level networking operations and do not represent the time that the web page actually requires to render. It was mentioned earlier that the *onContentLoad* (DOM is ready) and *onLoad* (page is loaded) properties of the HAR format are mapped to their JS equivalents, and that their times are somewhat arbitrary. However, they are the only measure that is available from the format to determine the PLT. Figure 23 shows a comparison of the *ssl*, *send*, and *receive* sum lined-up against the aforementioned time properties.

Protocol	Algorithm	Sharded	TLS (%)
HTTP	—	No	31.24
		Yes	30.46
HTTPS	ECDSA	No	Δ12.06
		Yes	Δ21.25
	RSA	No	Δ12.37
		Yes	Δ20.85
SPDY	ECDSA	No	Δ9.53
		Yes	Δ6.69
	RSA	No	Δ10.73
		Yes	Δ9.49

(a) First view.

Protocol	Algorithm	Sharded	TLS (%)
HTTP	—	No	28.48
		Yes	29.24
HTTPS	ECDSA	No	Δ11.76
		Yes	Δ11.73
	RSA	No	Δ12.25
		Yes	Δ11.58
SPDY	ECDSA	No	Δ13.44
		Yes	Δ4.37
	RSA	No	Δ14.50
		Yes	Δ3.17

(b) Repeated view.

Protocol	Algorithm	Sharded	TLS (%)
HTTP	—	No	29.41
		Yes	29.42
HTTPS	ECDSA	No	Δ11.23
		Yes	Δ11.59
	RSA	No	Δ11.12
		Yes	Δ11.68
SPDY	ECDSA	No	Δ12.53
		Yes	Δ5.29
	RSA	No	Δ12.68
		Yes	Δ5.4

(c) Repeated TFO view.

Table 12: Actual TLS impact of total *time*. The Δ denotes that the percentage is the difference between the real sum of all times subtracted by its HTTP counterpart.

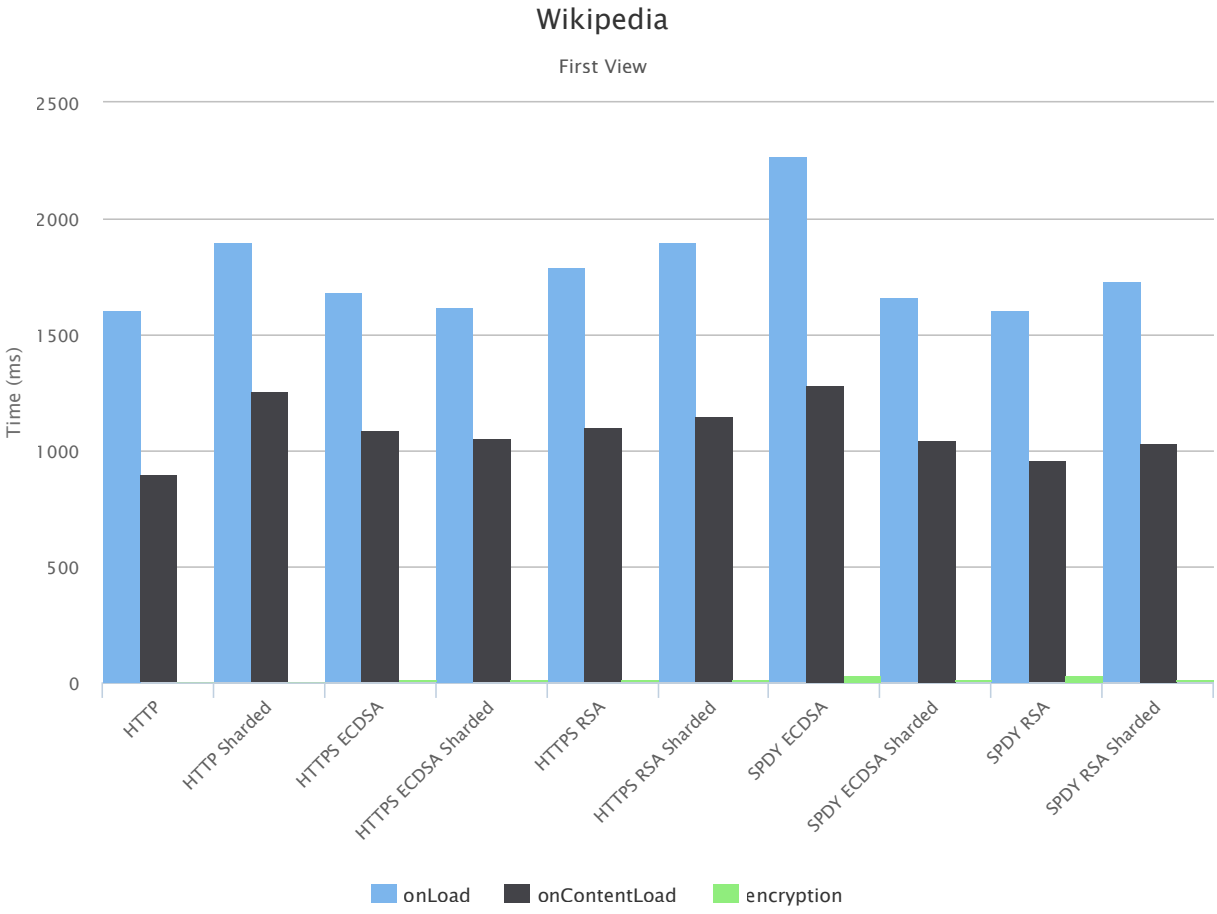


Figure 23: Total TLS time compared with *onContentLoad* and *onLoad*.

3.3 Session Ticket Key Rotation

Nginx supports session tickets (see section 2.6.1 for mode of operation) that are switched on by default, but there are several drawbacks with the current implementation that are not obvious. A new key for encryption is generated when nginx is started, restarted, or the configuration is reloaded (Dounin and Fussenegger 2014). Hence, the key is never rotated if none of these events occurs. Although long lived keys may improve performance, if paired with very long session timeouts, they effectively destroy PFS (Langley 2013). This is the reason why some major projects do not recommend using session tickets with nginx altogether (Dawson 2014).

Nginx developers state that the "current behaviour is believed to be good enough" (Dounin and Fussenegger 2014), and that frequent rotation should be implemented in OpenSSL rather than in their code base (Dounin and Gravion 2012). It is important to mention at this point that the behavior for session tickets is the same in Apache and many other software. Google's Go language has built in support for session ticket key rotation since April 2015.[81] RFC 5077 is not describing the management of keys, only recommendations are provided (Salowey et al. 2008, 10), and the OpenSSL developers decided to provide a callback[82] only and leave the implementation of the actual key management to the software.

Nginx's callback implementation is using a hard coded AES-256-CBC cipher with either SHA1 or SHA256 for hashing; see listing 19 for the relevant code from the nginx 1.9.0 source.[83] The result of this is that a ticket's data might be encrypted with a weaker cipher than the one used for the TLS channel (Ristić 2014, 58). The common consensus that 128 bit of security are strong enough for symmetric encryption for the next ten to twenty years, and that a generic web application should favor shorter lengths for performance reasons, allows the conclusion that the implementation restriction is not a tremendous problem. Especially, if the consideration that AES-192 and AES-256 might be reducible to AES-128 or worse is taken into account as well (Biryukov et al. 2010; Biryukov and Khovratovich 2009; Biryukov, Khovratovich, and Nikolić 2009; Dunkelman, Keller, and Shamir 2010).

Real attacks are not feasible today, but they will most likely reduce all versions of AES if they become usable (Bogdanov, Khovratovich, and Rechberger 2011). It is, therefore, still desirable that web servers provide the ability to swap the used cipher or determine

81. https://go-review.googlesource.com/#/c/9072/
82. https://www.openssl.org/docs/ssl/SSL_CTX_set_tlsext_ticket_key_cb.html
83. `src/event/ngx_event_openssl.c#2822`

```
2821 ...
2822 #ifdef OPENSSL_NO_SHA256
2823 #define ngx_ssl_session_ticket_md  EVP_sha1
2824 #else
2825 #define ngx_ssl_session_ticket_md  EVP_sha256
2826 #endif
2827
2828 static int
2829 ngx_ssl_session_ticket_key_callback(ngx_ssl_conn_t *ssl_conn,
2830     unsigned char *name, unsigned char *iv, EVP_CIPHER_CTX *ectx,
2831     HMAC_CTX *hctx, int enc)
2832 ...

2863 ...
2864         RAND_pseudo_bytes(iv, 16);
2865         EVP_EncryptInit_ex(ectx, EVP_aes_128_cbc(), NULL, key[0].aes_key, iv);
2866         HMAC_Init_ex(hctx, key[0].hmac_key, 16,
2867                      ngx_ssl_session_ticket_md(), NULL);
2868 ...
```

Listing 19: Nginx 1.9.0 session ticket key callback with highlighted encryption functions (lines omitted for brevity).

the best suited based on the server's strongest configured cipher for TLS channels. This would also allow applications with long term security requirements to utilize this feature. Still, constant ticket key rotation is far more important because a short lived key allows to decrypt only a small portion of previous messages if it gets compromised. A key that is never rotated and then compromised would enable an attacker to decrypt all previous messages, no matter how strong the key was.

3.3.1 Naive Approach

A simple *cron* command to reload or restart the web server on predefined intervals for key regeneration is the minimum that has to be implemented if tickets shall be used together with PFS, but this naive approach has several drawbacks. Most apparent is that the key is not sharable in clusters among multiple server instances because it is hidden in a software without the capability to support such an environment. Of course, this is not a problem for most websites, which are only hosted on a single server. However, another problem is that the currently in use key will be lost or overwritten by the newly generated one.

The web server consequently loses the ability to decrypt previously issued tickets because the server transmits a ticket that was encrypted with key A to the client. The cron job reloads the server after this request was handled, and a new key B is generated.

The client requests a new resource from the server—providing the previously received ticket that was encrypted using A—and the server is unable to decrypt the ticket with B. Session resumption has to be aborted, and a full TLS handshake is required to establish a new session.

A security vulnerability that affected older versions of nginx—as well as most other web servers and many websites like Akamai Technologies' CDN—was related to shared session states among multiple virtual servers, affecting both server side state and client side tickets. The range of possible attacks with the so-called "Virtual Host Confusion" is large, ranging from impersonating websites to session hijacking (Delignat-Lavaud and Bhargavan 2014). This issue was addressed with the immediate release of versions 1.6.2 and 1.7.5 (Dounin 2014). Nonetheless, package databases of some widely deployed server OSs are still offering old versions; Ubuntu, for instance, distributes nginx 1.4.6 for the long term support (LTS) version of their server branch.

3.3.2 Improved Approach

To overcome these drawbacks a shell script based program was developed that makes use of the *ssl_session_ticket_key* directive[84] to provide the keys for encryption and decryption of the session tickets. The decision to create a shell script and not, for instance, a C program is simply based on the fact that a Portable Operating System Interface (POSIX) compliant shell script is supported by almost any Unix OS without additional dependencies. The requirements for the program, based on the preceding considerations, are summarized in the following list:

1. Generated keys must be:
    - Cryptographically secure random.
    - Rotated regularly with the time being configurable.
    - Stored on volatile memory (Langley 2013).
2. Script must be executed during server boot before nginx starts.
3. Multiple TLS session ticket keys:
    - One encryption key.
    - Two decryption keys.
4. One key per domain to address "Virtual Host Confusion" for older releases.
5. Possibility of extension for server cluster sharing of the keys, either via a push or

84. http://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_session_ticket_key

pull scheme.

Multiple keys are not absolutely necessary as one would also suffice, however, they allow to advertise longer lifetimes on the tickets. Clients with tickets that were encrypted with an old key, which the server still possesses, do not have to undergo the full negotiation phase again. The server can simply issue a newly encrypted ticket with the current key if the session resumption was successful with the old key. More than one key is also helpful in cluster environments because web servers can wait before using the new key for encryption to ensure that every member of the cluster has it. Of course, the servers can already use it for decryption in case any of the other servers has a differing time and starts using the new key earlier. (Hoffman-Andrews 2013).

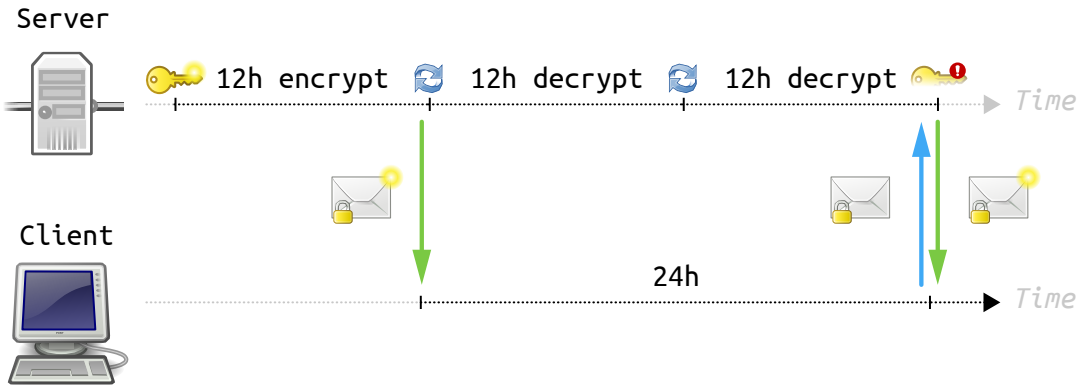


Figure 24: TLS session ticket key lifetime. (Fussenegger 2015i).

Figure 24 illustrates the usage of three keys with a twelve hour key rotation scheme and a thirty-six hour ticket lifetime. A new ticket is issued to the client right before the first rotation happens. The receiver of the ticket visits the server again after twenty-four hours and submits its ticket, which was encrypted with a key that was already rotated twice. The server, which is still in possession of that key, decrypts the ticket with the old key and issues a new ticket that was encrypted with the current encryption key. Of course, the server only does that if the resumption of the session was successful. A full handshake would be necessary if resumption fails, and the submitted ticket would be discarded.

The program that was developed consists of four shell scripts and a *make* file for automated de- and installation as well as running the tests. Settings are configured in *config.sh*, where the most important variables are the *cron* masks that define the rotation scheme. Listing 20 is an excerpt from the default settings that are included in the program. Keys are rotated every twelve hours, lifetimes of the tickets should be issued with twice this amount for maximum gain. Nginx is reloaded thirty minutes after the rotation has

happened, ensuring that the tickets are actually present and were correctly rotated. The server keeps the keys in its memory until the next server rotation/reload is performed, and, thus, the rotation of the keys does not affect the running process.

```
45 # The session ticket keys rotation interval as cron mask.
46 #
47 # Default is 12 hours which means that a key will reside in memory for 36 hours
48 # before it's deleted (three keys are used). You shouldn't go for much more
49 # than 24 hours for the encrypt key.
50 #
51 # The ticket's lifetime should be set to twice the lifetime of the encryption
52 # key.
53 readonly KEY_ROTATION='0 0,12 * * *'
54
55 # The nginx rotation interval as cron mask.
56 #
57 # This should be after the keys have been rotated (see $KEY_ROTATION). Note
58 # that keys are only in-use after nginx has read them. This is very important if
59 # you're syncing the keys within a cluster.
60 readonly SERVER_ROTATION='30 0,12 * * *'
61
62 # Absolute path to the web server system startup program.
63 readonly SERVER_INIT_PATH='/etc/init.d/nginx'
64
65 # The minimum version the server has to have for session ticket keys via files.
66 readonly SERVER_MIN_VERSION='1.5.7'
67
68 # The minimum version the OpenSSL library requires for session ticket support.
69 readonly OPENSSL_MIN_VERSION='0.9.8f'
```

Listing 20: Nginx session ticket key rotation configuration excerpt.

The installation script *install.sh* must be invoked with the domains of the virtual hosts for which the keys will be used. It is possible to simply pass an arbitrary name to the script if nginx version 1.6.2 or 1.7.5 and higher is used because the previously discussed vulnerability is not usable anymore in these releases, and a single key set is sufficient for all virtual hosts. However, installation starts with preliminary checks to ensure that the user who invoked the script possesses elevated privileges, the server is actually installed, and meets the minimum version requirement of 1.5.7 because earlier nginx versions do not have the *ssl_session_ticket_key* directive, and taking care of not overwriting an existing installation.

Installation continues, after all checks succeeded and no problems were detected, with mounting of the file system that is used to store the keys, directly followed by the creation of a mount point in *fstab*. This file contains the configuration for file systems that are to be automatically mounted when the server boots up. Memory to store the keys must be

volatile, thus, it must lose the stored data immediately if the system loses power. This is very important to thwart any attempts of key recovery. Linux OSs generally provide two file systems that are main memory mapped, accomplishing the requirement of volatility in form of the *ramfs* and the newer *tmpfs*. Table 13 contains an overview of the differences between both file systems.

Feature	*ramfs*	*tmpfs*
Size Limit	No	Yes
Swap	No	Yes
Volatile	Yes	Yes

Table 13: Differences between *ramfs* and *tmpfs*.

Servers usually have a swap partition that is required for paging. This makes *ramfs* the preferred file system for this program because it will never swap the keys to the hard disk if the main memory is full. The absence of size limitations could be a problem since *ramfs* will continue allocating unfreeable memory if data is written to it, until the system becomes unresponsive and most likely crashes (Landley 2005). A circumstance that could be exploited by a malicious user. However, the mounted file system is restricted to the user and group that is set in the configuration, which defaults to root. The mount process is also the reason why execution of the script is restricted to users with elevated privileges, as some involved commands are not executable by normal system users.

Creation of the *cron* job (listing 21) for key rotation and reloading of the web server is performed next, directly followed by the generation of the keys (listing 22) for all domains that were passed to the script. All three keys are generated; the fact that the second and third key are not usable for decryption does not matter to the web server, but it would throw an error if the keys referenced in its configuration did not exist. The last installation step involves the creation of a SysVinit script (listing 23) that is executed during the boot process to generate new keys because any of the previously generated keys are lost. The keys have to be generated before the web server starts, otherwise, it would again throw an error because of the missing files. This is accomplished through a dependency on the newly created SysVinit script within the web server's SysVinit script's *RequiredStart* attribute.

The installation of the program is finished, and the only task that has to be performed is the configuration of nginx. This is not automated and must be performed by the administrator because web server configurations tend to be comprehensive and highly customized. Listing 24 contains an example configuration for nginx that would work

```
0 0,12 * * * sh -- '/path/to/session-ticket-key-rotation/generator.sh' localhost
30 0,12 * * * service nginx reload
```

Listing 21: *Cron* job example for session ticket key rotation.

```
dd 'if=/dev/random' 'of=/mnt/session_ticket_keys/localhost.X.key' 'bs=1' 'count=48'
```

Listing 22: Command to generate a session ticket key.

```
#!/bin/sh

### BEGIN INIT INFO
# Provides:           session_ticket_keys
# Required-Start:     $local_fs $syslog
# Required-Stop:
# Default-Start:      2 3 4 5
# Default-Stop:
# Short-Description:  Generates random TLS session ticket keys on boot.
# Description:
#  The script will generate random TLS session ticket keys for all servers ...
### END INIT INFO

sh -- '/path/to/session-ticket-key-rotation/generator.sh' localhost
```

Listing 23: SysVinit example script for session ticket key rotation.

```
# ...
http {
    # ...
    server {
        # ...
        ssl_session_timeout         36h;
        ssl_session_ticket_key      /mnt/session_ticket_keys/localhost.1.key;
        ssl_session_ticket_key      /mnt/session_ticket_keys/localhost.2.key;
        ssl_session_ticket_key      /mnt/session_ticket_keys/localhost.3.key;
        # ...
    }
    # ...
}
```

Listing 24: Example configuration for session ticket keys in nginx.

together with the previous code listings and the default mount path for the ticket keys. The program would work with other web servers as well if they support session ticket keys. Apache, for instance, offers the *SSLSessionTicketKeyFile*, but the program was not tested with it and it is also not clear whether a *reload* would be sufficient or not.

Only Debian based Linux distributions were tested, but the code follows the POSIX standard and was tested with the very restrictive *dash* shell. This should ensure maximum compatibility with other Unix-like OSs. Support for sharing of the keys within server clusters is not provided by the program itself, but an implementation of this feature should be trivial. Servers within the cluster can either connect to the master server and pull the keys via Secure Shell (SSH) or the master server connects to the slave servers and pushes the keys via SSH, which is followed by a time-displaced reload. The program itself was released into the public domain (PD) and published on GitHub.[85,86]

85. https://github.com/Fleshgrinder/nginx-session-ticket-key-rotation
86. The complete repository is included on the enclosed volume.

4 Discussion & Outlook

The impact of TLS on web communication has changed a lot in the last twelve years since the paper of (Coarfa, Druschel, and Wallach 2002). Results from the tests that were conducted as part of this thesis show that encryption actually is negligible with its small impact of ∼11.5 % on low-level networking, and a not worth mentioning impact on the total PLT. The final numbers are somewhat surprising for SPDY because of its performance in the synthetic test, where one would expect SPDY to show what it was designed for. But many things that look sound in theory pose unexpected side effects in practice; however, this is not an unknown phenomenon but rather not communicated enough. There is a strong need for prioritization, and web servers should support this feature to speed up web applications. After all, nginx with its experimental SPDY module definitely played a role in the measured results.

The most serious problem with the benchmarks were the times that Chrome reported, and the fact that the web browser is working differently under Linux than it would under Windows. All benchmarks should be repeated with real hardware and a Windows system for the client side. This would also allow to use WebPagetest for the benchmarks, and subsequently to use different browsers and retrieve more reliable timing information. The approach to measure on the client side is good, but the software used in this thesis to perform the measurements is not. Developers and the W3C should push the development of the Navigation and Resource Timing APIs, and web browser vendors should implement them with an easily accessible interface, including CLI.[87,88]

Additionally, the benchmarks should be repeated as soon as HTTP/2 is broadly deployed and available in major web servers, like Apache and nginx; the same applies to TLS 1.3. Web browser support for the new protocol has already landed in Firefox, Chrome, and others will follow soon. Together with HTTP/2 comes the de facto mandatory requirement for encryption, and a fully encrypted Internet to counteract the NSA and alike, as well as criminals and malicious users. However, at least as important as these "*slogans*" for a free speech and a neutral Internet is the protection of web services, and while (Naylor et al. 2014) argue that the loss of in-network services is a problem, but in fact the in-network services sometimes are the problem themselves.

Web developers, administrators, as well as users should be able to control and decide what they consume, and what their devices are actually doing. This should not be done

87. http://www.w3.org/TR/resource-timing/
88. http://www.w3.org/TR/navigation-timing/

by third parties that are opaque to the users and perform various actions without their knowledge. Only cryptography can provide this freedom, as exemplified by the story of Edward Snowden. Improving the TLS session ticket rotation program to support more web servers besides nginx would help to comprehensively implement PFS. The changes should be trivial, as well as performing tests on more Linux distributions. However, an implementation in the web server itself would be much better, even if the nginx developers disagree, though, it could be provided by the security library as well. No matter which approach is chosen, both are valid and would definitely improve the future of the TLS's PFS support. Especially, with the advent of TLS 1.3 that might make PFS mandatory.

I would like to conclude this thesis with a quote of a not personally known mentor of mine.

> *TLS has exactly one performance problem: it is not used widely enough.*
> *Everything else can be optimized.*
>
> — (Grigorik 2014, Slide 5)

List of Figures

List of Listings

List of Tables

Acronyms

3WHS . . . . . Three-Way Handshake

AEAD . . . . . Authenticated Encryption with Associated Data

AES . . . . . Advanced Encryption Standard

AIMD . . . . . Additive Increase/Multiplicative Decrease

AJAX . . . . . Asynchronous JavaScript + XML

ALPN . . . . . Application Layer Protocol Negotiation

Amazon . . . . . Amazon.com Incorporation

Apache . . . . . Apache HTTP Server

API . . . . . Application Programming Interface

Apple . . . . . Apple Incorporation

ARPANET . . . . . Advanced Research Projects Agency Network

AtE . . . . . Authenticate-then-Encrypt

ATM . . . . . Asynchronous Transfer Mode

BDP . . . . . Bandwidth-Delay Product

BEAST . . . . . Browser Exploit Against SSL/TLS

BGP . . . . . Border Gateway Protocol

CA . . . . . Certificate Authority

CBC . . . . . Cipher Block Chaining

CDN . . . . . Content Delivery Network

CIA . . . . . Central Intelligence Agency

CIA triad . . . . . Confidentiality, Integrity, and Availability

CLI . . . . . Command-Line Interface

CPU . . . . . Central Processing Unit

CR . . . . . Carriage Return

CRIME . . . . . Compression Ratio Info-leak Made Easy

CRL . . . . . Certificate Revocation List

CSPRNG . . . . . Cryptographically Secure PRNG

CSS . . . . . Cascading Style Sheets

CSSOM . . . . . CSS Object Model

CSV . . . . . Command-Separated Values

cwnd . . . . . Congestion Window

DES . . . . . Data Encryption Standard

DH . . . . . Diffie–Hellman Key Exchange

DHE . . . . . Ephemeral Diffie–Hellman Key Exchange

DNS . . . . . Domain Name System

DOM . . . . . Document Object Model

DPI . . . . . Deep Packet Inspection

DRBG . . . . . Deterministic Random Bit Generator

DTLS . . . . . Datagram Transport Layer Security

E2EE . . . . . End-to-End Encryption

E&A . . . . . Encrypt-and-Authenticate

E&M . . . . . Encrypt-and-MAC

EC . . . . . Elliptic Curve

ECC . . . . . Elliptic Curve Cryptography

ECDH . . . . . Elliptic Curve Diffie–Hellman

ECDHE . . . . . Elliptic Curve Diffie–Hellman Key Exchange

ECDSA . . . . . Elliptic Curve Digital Signature Algorithm

EtA . . . . . Encrypt-then-Authenticate

EtM . . . . . Encrypt-then-MAC

FIFO . . . . . First-In-First-Out

FOC . . . . . Fiber Optic Cable

FTTH . . . . . Fiber to the Home

GCM . . . . . Galois/Counter Mode

Google . . . . . Google Incorporation

HAR . . . . . HTTP Archive

HMAC . . . . . Hash Message Authentication Code

HOL blocking . . . . . Head-of-Line Blocking

HSTS . . . . . HTTP Strict Transport Security

HTML . . . . . Hypertext Markup Language

HTTP . . . . . Hypertext Transfer Protocol

HTTP-NG . . . . . HTTP Next Generation

HTTPS . . . . . HTTP Secure

IDEA . . . . . International Data Encryption Algorithm

IE . . . . . Internet Explorer

IETF . . . . . Internet Engineering Task Force

IHS . . . . . Internet Hosting Service

InfoSec . . . . . Information Security

init_cwnd . . . . . Initial Congestion Window

init_rwnd . . . . . Initial Receive Window

IP . . . . . Internet Protocol

IPS . . . . . Internet Protocol Suite

IPSec . . . . . Internet Protocol Security

IPv4 . . . . . Internet Protocol Version 4

IPv6 . . . . . Internet Protocol Version 6

ISoc . . . . . Internet Society

ISP . . . . . Internet Service Provider

IV . . . . . Initialization Vector

JS . . . . . JavaScript

JSON . . . . . JavaScript Object Notation

JSSE . . . . . Java Secure Socket Extension

LF . . . . . Line Feed

LFN . . . . . Long Fat Network (*elephan*)

Lucky 13 . . . . . Lucky Thirteen attack

MAC . . . . . Message Authentication Code

MAC . . . . . Media Access Control

MD . . . . . Message-Digest Algorithm

Microsoft . . . . . Microsoft Corporation

Mozilla . . . . . Mozilla Foundation

MSS . . . . . Maximum Segment Size

MtE . . . . . MAC-then-Encrypt

MTU . . . . . Maximum Transmission Unit

NAT . . . . . Network Address Translation

Netscape . . . . . Netscape Communications Incorporation

NIST . . . . . National Institute of Standards and Technology

NOP . . . . . No Operation

NPN . . . . . Next Protocol Negotiation

NSA . . . . . National Security Agency

NSS . . . . . Network Security Services

NTP . . . . . Network Time Protocol

OCSP . . . . . Online Certificate Status Protocol

OS . . . . . Operating System

OSI model . . . . . Open Systems Interconnection Model

OTR . . . . . Off-the-Record

P2P . . . . . Peer-to-Peer

P2PE . . . . . Point-to-Point Encryption

PCT . . . . . Private Communications Technology

PD . . . . . Public Domain

PFS . . . . . Perfect Forward Secrecy

PKI . . . . . Public Key Infrastructure

PLT . . . . . Page Load Time

POSIX . . . . . Portable Operating System Interface

PRF . . . . . Pseudo Random Function

PRNG . . . . . Pseudo Random Number Generator

PRR . . . . . Proportional Rate Reduction

QoS . . . . . Quality of Service

QUIC . . . . . Quick UDP Internet Connection

RC4 . . . . . Rivest Cipher 4

RC5 . . . . . Rivest Cipher 5

RFC . . . . . Request for Comments

RIPEMD . . . . . RACE Integrity Primitives Evaluation Message Digest

RNG . . . . . Random Number Generator

RSA . . . . . Rivest Shamir Adleman Cryptosystem

RTT . . . . . Round-Trip Time

rwnd . . . . . Receive Window

SHA . . . . . Secure Hash Algorithm

SNI . . . . . Server Name Indication

SPSS . . . . . IBM SPSS Statistics

SSH . . . . . Secure Shell

SSL . . . . . Secure Socket Layer

SSR . . . . . Slow-Start Restart

ssthresh . . . . . Slow-Start Threshold

TCP . . . . . Transmission Control Protocol

TFO . . . . . TCP Fast Open

TLD . . . . . Top-Level Domain

TLS . . . . . Transport Layer Security

ToS . . . . . Terms of Service

TRNG . . . . . True Random Number Generator

TTFB . . . . . Time to First Byte

UA . . . . . User Agent

UDP . . . . . User Datagram Protocol

URI . . . . . Uniform Resource Identifier

URL . . . . . Uniform Resource Locator

VM . . . . . Virtual Machine

VPN . . . . . Virtual Private Network

W3C . . . . . World Wide Web Consortium

WebRTC . . . . . Web Real-Time Communication

WWW . . . . . World Wide Web

XML . . . . . Extensible Markup Language

References

Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI). 2014. *Le Référentiel Général de Sécurité: Annexe B1 – Mécanismes Cryptographiques* [in French]. RGS 2.03. Paris, France, February 21. Accessed May 20, 2015. http://www.ssi.gouv.fr/uploads/2015/01/RGS_v-2-0_B1.pdf.

Allman, Mark, Vern Paxson, and Ethan Blanton. 2009. *TCP Congestion Control.* RFC, Internet Requests for Comments 5681. Status: Draft Standard; Obsoletes: RFC 2581; Stream: IETF. RFC Editor, September. Accessed May 20, 2015. doi:10.17487/RFC5681. http://www.rfc-editor.org/rfc/pdfrfc/rfc5681.txt.pdf.

Allman, Mark, Vern Paxson, and W. Richard Stevens. 1999. *TCP Congestion Control.* RFC, Internet Requests for Comments 2581. Status: Proposed Standard; Obsoletes: RFC 2001; Obsoleted by: RFC 5681; Updated by: RFC 3390; Stream: IETF. RFC Editor, April. Accessed May 20, 2015. doi:10.17487/RFC2581. http://www.rfc-editor.org/rfc/pdfrfc/rfc2581.txt.pdf.

Andy. 2014. "BBC: ISPs Should Assume Heavy VPN Users are Pirates." *TorrentFreak* (blog), September 8. Accessed May 20, 2015. https://torrentfreak.com/?p=93684.

Antony, Antony, Daniel Karrenberg, Robert Kisteleki, Tiziana Refice, and Rene Wilhelm. 2008. *YouTube Hijacking (February 24th 2008) Analysis of (BGP) Routing Dynamics.* Technical report RT-DIA-123-2008. Rome, Italy: Dept. of Computer Science and Automation, University of Roma Tre, February. Accessed May 20, 2015. http://www.dia.uniroma3.it/~compunet/www/docs/2008-123.pdf.

Apostolopoulos, George, Vinod Peris, and Debanjan Saha. 1999. "Transport Layer Security: How Much Does it Really Cost?" In *Proceedings IEEE INFOCOM '99: The Conference on Computer Communications,* Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies, 3:717–25. New York, NY, USA: Institute of Electrical and Electronics Engineers, Inc. (IEEE), March. ISBN: 0-7803-5417-6, accessed May 20, 2015. doi:10.1109/INFCOM.1999.751458. http://user.informatik.uni-goettingen.de/~fu/ptls/apostolopoulos99transport.pdf.

Badra, Mohamad, Omar Cherkaoui, Ibrahim Hajjeh, and Ahmed Serhrouchni. 2005. *Pre-Shared Key Ciphersuites for Transport Layer Security (TLS).* RFC, Internet Requests for Comments 4279. Status: Proposed Standard; Stream: IETF. RFC Editor, December. Accessed May 20, 2015. doi:10.17487/RFC4279. http://www.rfc-editor.org/rfc/pdfrfc/rfc4279.txt.pdf.

Barnes, Richard. 2015. "Deprecating Non-Secure HTTP." *Mozilla Security Blog* (blog) (April 30). Accessed May 20, 2015. https://blog.mozilla.org/security/2015/04/30/deprecating-non-secure-http/.

Barnes, Richard, Martin Thomson, Alfredo Pironti, and Adam Langley. 2015. *Deprecating Secure Sockets Layer Version 3.0.* Work in Progress, Internet-Draft draft-ietf-tls-sslv3-diediedie-03. April 10. Accessed May 20, 2015. https://tools.ietf.org/pdf/draft-ietf-tls-sslv3-diediedie-03.pdf.

Belshe, Mike. 2010. *More Bandwidth Doesn't Matter (Much).* Google Inc., April 8. Accessed May 20, 2015. https://docs.google.com/a/chromium.org/viewer?a=v&pid=sites&srcid=Y2hyb21pdW0ub3JnfGRldnxneDoxMzcyOWI1N2I4YzI3NzE2.

———. 2011. "SSL FalseStart Performance Results." *Chromium Blog: News and developments from the open source browser project* (blog) (May 18). Accessed May 27, 2015. http://blog.chromium.org/2011/05/ssl-falsestart-performance-results.html.

Belshe, Mike, and Roberto Peon. 2009. "A 2x Faster Web." *Google Research Blog* (blog) (November 12). Accessed May 20, 2015. http://googleresearch.blogspot.com/2009/11/2x-faster-web.html.

———. 2015. *Hypertext Transfer Protocol Version 2 (HTTP/2).* RFC, Internet Requests for Comments 7540. Status: Proposed Standard; Stream: IETF. RFC Editor, May. Accessed May 20, 2015. doi:10.17487/RFC7540. http://www.rfc-editor.org/rfc/pdfrfc/rfc7540.txt.pdf.

Belson, David, ed. 2014. *Akamai's State of the Internet: Q4 [2014 Report].* The Akamai State of the Internet Report Volume 7 / Number 4. Cambridge, MA, USA. Accessed May 20, 2015. https://www.akamai.com/stateoftheinternet/.

Benaloh, Josh, Butler Lampson, Daniel R. Simon, Terence Spies, and Bennet Yee. 1996. *The Private Communication Technology Protocol.* Working Draft, Internet-Draft draft-benaloh-pct-00. State: Expired. April 5. "This document specifies Version 2 of the Private Communication Technology (PCT) protocol […]" but the actual available document specifies version 1, accessed May 20, 2015, https://tools.ietf.org/pdf/draft-benaloh-pct-00.pdf.

Benbennick, David. 2006. *Complete graph K3,* January 14. SVG. Wikimedia Commons. Public Domain, accessed May 21, 2015, https://commons.wikimedia.org/wiki/File:Complete_graph_K3.svg.

Bentzel, Chris, and Bence Béky. 2015. "Hello HTTP/2, Goodbye SPDY." *Chromium Blog: News and developments from the open source browser project* (blog) (February 9). Accessed May 21, 2015. http://blog.chromium.org/2015/02/hello-http2-goodbye-spdy-http-is_9.html.

Bernat, Vincent. 2011. "SSL/TLS & Perfect Forward Secrecy." *Disruptive Ninja* (blog), November 28. Accessed May 20, 2015. http://vincent.bernat.im/en/blog/2011-ssl-perfect-forward-secrecy.html.

Berners-Lee, Tim. 1991. "The Original HTTP as defined in 1991." Accessed May 20, 2015. http://www.w3.org/Protocols/HTTP/AsImplemented.html.

Berners-Lee, Tim, Roy T. Fielding, and Henrik Frystyk Nielsen. 1996. *Hypertext Transfer Protocol -- HTTP/1.0.* RFC, Internet Requests for Comments 1945. Status: Informational; Stream: IETF. RFC Editor, May. Accessed May 20, 2015. doi:10.17487/RFC1945. http://www.rfc-editor.org/rfc/pdfrfc/rfc1945.txt.pdf.

Berners-Lee, Tim. 1992a. "Basic HTTP as defined in 1992." Accessed May 20, 2015. http://www.w3.org/Protocols/HTTP/HTTP2.html.

———. 1992b. "HTTP 0.9 Summary." Accessed May 20, 2015. http://www.w3.org/DesignIssues/HTTP0.9Summary.html.

Bernstein, Daniel J. 2006. "Curve25519: New Diffie-Hellman Speed Records: Third International Workshop on Practice and Theory in Public Key Cryptosystems." New York, NY, USA, April 24–26, 2006. Proceedings, edited by David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, et al., 3958:207–28. Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg, Germany: Springer-Verlag Berlin Heidelberg, April. ISBN: 978-3-540-33851-2, accessed May 20, 2015. doi:10.1007/11745853_14. http://cr.yp.to/ecdh/curve25519-20060209.pdf.

Bernstein, Daniel J., Erik Dahmen, and Johannes Buchmann, eds. 2009. *Post-Quantum Cryptography.* 1st ed. Berlin/Heidelberg, Germany: Springer-Verlag Berlin Heidelberg. ISBN: 978-3-540-88702-7, accessed May 20, 2015. doi:10.1007/978-3-540-88702-7. http://pqcrypto.org/.

Bernstein, Daniel J., and Tanja Lange. 2013. "Rigidity." *SafeCurves: choosing safe curves for elliptic-curve cryptography,* October 25. Accessed May 20, 2015. http://safecurves.cr.yp.to/rigid.html.

Biryukov, Alex, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, and Adi Shamir. 2010. "Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds." In *Advances in Cryptology – EUROCRYPT 2010: 29th Annual Internatinal Conference on the Theory and Applications of Crytographic Techniques,* French Riviera, May 30–June 3, 2010. Proceedings, edited by Henri Gilbert, 6110:299–319. Lecture Notes in Computer Science (LNCS). Cryptology ePrint Archive 374: https://eprint.iacr.org/2009/374.pdf, Slides: http://msrvideo.vo.msecnd.net/rmcvideos/115866/dl/115866.pdf. French Riviera, France and Monaco: Springer-Verlag Berlin Heidelberg, June. ISBN: 978-3-642-13189-9, accessed May 21, 2015. doi:10.1007/978-3-642-13190-5_15. http://www.springer.com/de/book/9783642131899.

Biryukov, Alex, and Dmitry Khovratovich. 2009. "Related-Key Cryptanalysis of the Full AES-192 and AES-256." In *Advances in Cryptology – ASIACRYPT 2009: 15th International Conference on the Theory and Application of Cryptology and Information Security,* Tokyo, Japan, December 6–10, 2009. Proceedings, edited by Mitsuru Matsui, 5912:1–18. Lecture Notes in Computer Science (LNCS). Cryptology ePrint Archive 317: https://eprint.iacr.org/2009/317.pdf. Tokyo, Japan: Springer-Verlag Berlin Heidelberg, December. ISBN: 978-3-642-10365-0, accessed May 21, 2015. doi:10.1007/978-3-642-10366-7_1. http://www.springer.com/de/book/9783642103650.

Biryukov, Alex, Dmitry Khovratovich, and Ivica Nikolić. 2009. "Distinguisher and Related-Key Attack on the Full AES-256." In *Advances in Cryptology – CRYPTO 2009: 29th Annual International Cryptology Conference,* Santa Barbara, CA, USA, August 16–20, 2009. Proceedings, edited by Shai Halevi, 5677:231–49. Lecture Notes in Computer Science (LNCS). Cryptology ePrint Archive 241: https://eprint.iacr.org/2009/241.pdf. Santa Barbara, CA, USA: Springer-Verlag Berlin Heidelberg, August. ISBN: 978-3-642-03355-1, accessed May 21, 2015. doi:10.1007/978-3-642-03356-8_14. http://www.springer.com/de/book/9783642033551.

Blake-Wilson, Simon, Nelson Bolyard, Vipul Gupta, Chris Hawk, and Bodo Moeller. 2006. *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS).* RFC, Internet Requests for Comments 4492. Status: Informational; Updated by: RFC 5246, RFC 7027; Stream: IETF. RFC Editor, May. Accessed May 20, 2015. doi:10.17487/RFC4492. http://www.rfc-editor.org/rfc/pdfrfc/rfc4492.txt.pdf.

Blake-Wilson, Simon, Magnus Nystrom, David Hopwood, Jan Mikkelsen, and Tim Wright. 2003. *Transport Layer Security (TLS) Extensions.* RFC, Internet Requests for Comments 3546. Status: Proposed Standard; Obsoleted by: RFC 4366; Updates: RFC 2246; Stream: IETF. RFC Editor, June. Accessed May 20, 2015. doi:10.17487/RFC3546. http://www.rfc-editor.org/rfc/pdfrfc/rfc3546.txt.pdf.

Bleichenbacher, Daniel. 1998. "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1." In *Advances in Cryptology — CRYPTO 1998: 18th Annual International Cryptology Conference,* Santa Barbara, CA, USA, August 23–27, 1998. Proceedings, edited by Hugo Krawczyk, 1462:1–12. Lecture Notes in Computer Science (LNCS). Santa Barbara, CA, USA: Springer-Verlag Berlin Heidelberg, August. ISBN: 978-3-540-64892-5, accessed May 20, 2015. doi:10.1007/BFb0055715. http://archiv.infsec.ethz.ch/education/fs08/secsem/Bleichenbacher98.pdf.

Bogdanov, Andrey, Dmitry Khovratovich, and Christian Rechberger. 2011. "Biclique Cryptanalysis of the Full AES." In *Advances in Cryptology – ASIACRYPT 2011: 17th International Conference on the Theory and Application of Cryptology and Information Security,* Seoul, South Korea, December 4–8, 2011. Proceedings, edited by Xiaoyun Wang, 7073:344–71. Lecture Notes in Computer Science (LNCS). Seoul, South Korea: Springer-Verlag Berlin Heidelberg, December. ISBN: 978-3-642-25384-3, accessed May 20, 2015. doi:10.1007/978-3-642-25385-0_19. https://research.microsoft.com/en-us/projects/cryptanalysis/aesbc.pdf.

Borisov, Nikita, Ian Goldberg, and Eric Brewer. 2004. "Off-the-Record Communication, or, Why Not To Use PGP." In *Proceedings of WPES '04: ACM Workshop on Privacy in the Electronic Society,* 2004:77–84. WPES. Washington D.C., USA: Association for Computing Machinery (ACM). ISBN: 1-58113-968-3, accessed May 20, 2015. doi:10.1145/1029179.1029200. https://otr.cypherpunks.ca/otr-wpes.pdf.

Bos, Joppe W., Craig Costello, Michael Naehrig, and Douglas Stebila. 2015. "Post-Quantum Key Exchange for the TLS Protocol from the Ring Learning with Errors Problem." In *Proceedings of SP '15: 36th IEEE Symposium on Security and Privacy,* vol. 36. IEEE SP. IEEE. Accessed May 20, 2015. http://www.douglas.stebila.ca/research/papers/bcns15.

Braden, Robert T., ed. 1989a. *Requirements for Internet Hosts - Application and Support.* RFC, Internet Requests for Comments 1123. Status: Internet Standard; Updates: RFC 822, RFC 952; Updated by: RFC 1349, RFC 2181, RFC 5321, RFC 5966; Stream: Legacy. RFC Editor, October. Accessed May 20, 2015. doi:10.17487/RFC1123. http://www.rfc-editor.org/rfc/pdfrfc/rfc1123.txt.pdf.

———, ed. 1989b. *Requirements for Internet Hosts - Communication Layers.* RFC, Internet Requests for Comments 1122. Status: Internet Standard; Updates: RFC 793; Updated by: RFC 1349, RFC 4379, RFC 5884, RFC 6093, RFC 6298, RFC 6633, RFC 6864; Stream: Legacy. RFC Editor, October. Accessed May 20, 2015. doi:10.17487/RFC1122. http://www.rfc-editor.org/rfc/pdfrfc/rfc1122.txt.pdf.

———. 1994. *T/TCP -- TCP Extensions for Transactions Functional Specification.* RFC, Internet Requests for Comments 1644. Status: Historic; Obsoleted by: RFC 6247; Updates: RFC 1379; Stream: Legacy. RFC Editor, July. Accessed May 20, 2015. doi:10.17487/RFC1644. http://www.rfc-editor.org/rfc/pdfrfc/rfc1644.txt.pdf.

Brandt, Mathias. 2014. "Cybercrime boomt in Deutschland" [in German]. *Statista,* June 4. Accessed May 20, 2015. http://de.statista.com/infografik/1614/.

Brutlag, Jake. 2009a. "Speed Matters." *Google Research Blog* (blog) (June 23). Accessed May 20, 2015. http://googleresearch.blogspot.co.at/2009/06/speed-matters.html.

———. 2009b. "Speed Matters for Google Web Search." *Google Research Blog* (blog) (June 22). Accessed May 20, 2015. http://services.google.com/fh/files/blogs/google_delayexp.pdf.

Bundesamt für Sicherheit in der Informationstechnik (BSI). 2015. *Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung: Übersicht über geeignete Algorithmen* [in German]. BAnz AT 30.01.2015 B3. Berlin, Germany: Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen (BNetzA), January 30.

Canadi, Igor, Paul Barford, and Joel Sommers. 2012. "Revisiting Broadband Performance." In *IMC '12: Proceedings of the ACM Internet Measruement Conference,* edited by John W. Byers, Jim Kurose, Ratul Mahajan, and Alex C. Snoeren, 273–86. Boston, MA, USA: Association for Computing Machinery, Inc. (ACM), November. ISBN: 978-1-4503-1705-4, accessed May 20, 2015. doi:10.1145/2398776.2398805. http://www-net.cs.umass.edu/imc2012/papers/p273.pdf.

Cerf, Vinton G., and Robert E. Kahn. 1974. "A Protocol for Packet Network Intercommunication." *IEEE Transactions on Communications,* no. 5, 637–48. ISSN: 0096-2244, accessed May 20, 2015. doi:10.1109/TCOM.1974.1092259. http://www.cs.princeton.edu/courses/archive/fall06/cos561/papers/cerf74.pdf.

Cermak, Gregory W. 2005. "Multimedia Quality as a Function of Bandwidth, Packet Loss, and Latency." *International Journal of Speech Technology* 8 (3): 259–70. ISSN: 1381-2416, accessed May 20, 2015. doi:10.1007/s10772-006-6368-3. http://enpub.fulton.asu.edu/resp/vpqm/vpqm2005/papers/221.pdf.

Chan, William. 2012. "Prioritization Is Critical to SPDY." *Insouciant: William Chan's blag* (blog) (December 24). Accessed May 31, 2015. https://insouciant.org/tech/prioritization-is-critical-to-spdy/.

———. 2013a. "Some Quick Thoughts on TCP Fast Open and Chromium." *Insouciant: William Chan's blag* (blog) (April 12). Accessed May 20, 2015. https://insouciant.org/tech/some-quick-thoughts-on-tcp-fast-open-and-chromium/.

———. 2013b. "Some Quick Thoughts on TCP Fast Open and Chromium." *Insouciant: William Chan's blag* (blog) (May 20). Accessed May 20, 2015. https://insouciant.org/tech/network-congestion-and-web-browsing/.

———. 2013c. "Some Quick Thoughts on TCP Fast Open and Chromium." *Insouciant: William Chan's blag* (blog) (February 28). Accessed May 20, 2015. https://insouciant.org/tech/connection-management-in-chromium/.

———. 2014. "HTTP/2 Considerations and Tradeoffs." *Insouciant: William Chan's blag* (blog) (January 10). Accessed May 20, 2015. https://insouciant.org/tech/http-slash-2-considerations-and-tradeoffs/.

Chan, William, Jose E. Roman, Chris Bentzel, and Mike Pinkerton. 2010. "Issue 44501: Consider changing connections per proxy server limit." May. Accessed May 24, 2015. https://code.google.com/p/chromium/issues/detail?id=44501.

Cheng, Yuchung, Jerry Chu, Sivasankar Radhakrishnan, and Arvind Jain. 2014. *TCP Fast Open.* RFC, Internet Requests for Comments 7413. Status: Experimental; Stream: IETF. RFC Editor, December. Accessed May 20, 2015. doi:10.17487/RFC7413. http://www.rfc-editor.org/rfc/pdfrfc/rfc7413.txt.pdf.

Chu, Jerry, Nandita Dukkipati, Yuchung Cheng, and Matt Mathis. 2013. *Increasing TCP's Initial Window.* RFC, Internet Requests for Comments 6928. Status: Experimental; Stream: IETF. RFC Editor, April. Accessed May 20, 2015. doi:10.17487/RFC6928. http://www.rfc-editor.org/rfc/pdfrfc/rfc6928.txt.pdf.

Coarfa, Cristian, Peter Druschel, and Dan S. Wallach. 2002. "Performance Analysis of TLS Web Servers." In *Proceedings ISOC NDSS '02: Network and Distributed System Security Symposium,* 1–12. San Diego, CA, USA: The Internet Society (ISoc), February. Revised: http://www.cs.rice.edu/~dwallach/pub/tls-tocs.pdf, ISBN: 1-891562-14-2, accessed May 20, 2015, https://www.isoc.org/isoc/conferences/ndss/02/papers/coarfa.pdf.

Comer, Douglas E. 2014. *Internetworking with TCP/IP Vol I: Principles, Protocol, and Architecture.* 6th ed. Upper Saddle Creek, NJ, USA: Pearson Education Inc. ISBN: 978-0-13-344975-4, accessed May 20, 2015. https://www.cs.purdue.edu/homes/comer/netbooks.html.

Cooper, David, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and Tim Polk. 2008. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* RFC, Internet Requests for Comments 5280. Status: Proposed Standard; Obsoletes: RFC 3280, RFC 4325, RFC 4630; Updated by: RFC 6818; Stream: IETF. RFC Editor, May. Accessed May 20, 2015. doi:10.17487/RFC5280. http://www.rfc-editor.org/rfc/pdfrfc/rfc5280.txt.pdf.

Dawson, Andy. 2014. "ssl.conf." H5bp/server-configs-nginx. July 28. Accessed May 20, 2015. https://github.com/h5bp/server-configs-nginx/blob/master/h5bp/directive-only/ssl.conf#L19-L28.

Delignat-Lavaud, Antoine, and Karthikeyan Bhargavan. 2014. "Virtual Host Confusion: Weaknesses and Exploits: Black Hat 2014 Report." In *Black Hat USA 2014.* San Francisco, CA, USA: UBM Tech, August. Accessed May 20, 2015. http://bh.ht.vc/vhost_confusion.pdf.

Dent, Alexander W. 2010. "Choosing Key Sizes for Cryptography." *Information Security Technical Report* 15 (1): 21–27. ISSN: 13634127, accessed May 20, 2015. doi:10.1016/j.istr.2010.10.006. http://www.csb.uncw.edu/people/cummingsj/classes/MIS534/Articles/Ch10bCryptographyKeys.pdf.

Dierks, Tim. 2014. "Security Standards and Name Changes in the Browser Wars." *Tim Dierks* (blog), May 23. Accessed May 20, 2015. http://tim.dierks.org/2014/05/security-standards-and-name-changes-in.html.

Dierks, Tim, and Christopher Allen. 1999. *The TLS Protocol Version 1.0.* RFC, Internet Requests for Comments 2246. Status: Proposed Standard; Obsoleted by: RFC 4346; Updated by: RFC 3546, RFC 5746, RFC 6176, RFC 7465, RFC 7507; Stream: IETF. RFC Editor, January. Accessed May 20, 2015. doi:10.17487/RFC2246. http://www.rfc-editor.org/rfc/pdfrfc/rfc2246.txt.pdf.

Dierks, Tim, and Eric Rescorla. 2006. *The Transport Layer Security (TLS) Protocol Version 1.1.* RFC, Internet Requests for Comments 4346. Status: Proposed Standard; Obsoletes: RFC 2246; Obsoleted by: RFC 5246; Updated by: RFC 4366, RFC 4680, RFC 4681, RFC 5746, RFC 6176, RFC 7465, RFC 7507; Stream: IETF. RFC Editor, April. Accessed May 20, 2015. doi:10.17487/RFC4346. http://www.rfc-editor.org/rfc/pdfrfc/rfc4346.txt.pdf.

———. 2008. *The Transport Layer Security (TLS) Protocol Version 1.2.* RFC, Internet Requests for Comments 5246. Status: Proposed Standard; Obsoletes: RFC 3268, RFC 4346, RFC 4366; Updates: RFC 4492; Updated by: RFC 5746, RFC 5878, RFC 6176, RFC 7465, RFC 7507; Stream: IETF. RFC Editor, August. Accessed May 20, 2015. doi:10.17487/RFC5246. http://www.rfc-editor.org/rfc/pdfrfc/rfc5246.txt.pdf.

Diffie, Bailey W., Martin E. Hellman, and Ralph C. Merkle (Stanford University). 1980. Cryptographic Apparatus and Method. http://www.lens.org/lens/patent/US_4200770_A US 4200770 A (Stanford, CA, USA), filed September 6, 1977, and issued April 29, 1980. Accessed May 20, 2015. https://www.google.com/patents/US4200770.

Diffie, Whitfield, and Martin E. Hellman. 1976. "New Directions in Cryptography." *IEEE Transactions on Information Theory* 22 (6): 644–54. ISSN: 0018-9448, accessed May 20, 2015. doi:10.1109/TIT.1976.1055638. https://ee.stanford.edu/~hellman/publications/24.pdf.

Dounin, Maxim. 2014. "[nginx-announce] nginx security advisory (CVE-2014-3616): Nginx Development Mailing List." September 16. Accessed May 29, 2015. http://mailman.nginx.org/pipermail/nginx-announce/2014/000147.html.

Dounin, Maxim, and Richard Fussenegger. 2014. "Session Ticket Rotation: Nginx Development Mailing List." September 22. Accessed May 20, 2015. http://mailman.nginx.org/pipermail/nginx-devel/2014-September/005979.html.

Dounin, Maxim, and Eiji Gravion. 2012. "Re: proper setup for forward secrecy: Nginx Development Mailing List." September 24. Accessed May 20, 2015. http://mailman.nginx.org/pipermail/nginx/2012-September/035568.html.

Duell, Jason, Patrick McManus, Ryan Van der Meulen, and Baron David. 2012. "Bug 648603 - Bump max HTTP proxy connection limits." September. Accessed May 24, 2015. https://bugzilla.mozilla.org/show_bug.cgi?id=648603.

Dukkipati, Nandita, Matt Mathis, Yuchung Cheng, and Monia Ghobadi. 2011. "Proportional rate reduction for TCP." In *IMC '11: Proceedings of the 2011 ACM SIGCOMM Internet Measurement Conference,* edited by Patrick Thiran and Walter Willinger, 155–69. Berlin, Germany: Association for Computing Machinery (ACM), November. ISBN: 978-1-4503-1013-0, accessed May 20, 2015. doi:10.1145/2068816.2068832. http://conferences.sigcomm.org/imc/2011/docs/p155.pdf.

Dukkipati, Nandita, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, and Natalia Sutin. 2010. "An Argument for Increasing TCP's Initial Congestion Window." ACM SIGCOMM. *Computer Communication Review* (New York, NY, USA) 40, no. 3 (July): 26–33. ISSN: 0146-4833, accessed May 20, 2015. doi:10.1145/1823844.1823848. http://research.google.com/pubs/pub36640.html.

Dunkelman, Orr, Nathan Keller, and Adi Shamir. 2010. "Improved Single-Key Attacks on 8-Round AES-192 and AES-256." In *Advances in Cryptology – ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security,* Singapore, December 5–9, 2010. Proceedings, edited by Masayuki Abe, 6477:158–76. Lecture Notes in Computer Science (LNCS). Singapore: Springer-Verlag Berlin Heidelberg, December. ISBN: 978-3-642-17372-1, accessed May 20, 2015. doi:10.1007/978-3-642-17373-8_10. http://www.math.huji.ac.il/~nkeller/Crypt-conf1.pdf.

Duong, Thai, and Juliano Rizzo. 2011. "Here Come The XOR Ninjas." Known as BEAST, accessed May 20, 2015, http://www.hpcc.ecs.soton.ac.uk/~dan/talks/bullrun/Beast.pdf.

Durumeric, Zakir, James Kasten, Michael Bailey, and J. Alex Halderman. 2013. "Analysis of the HTTPS Certificate Ecosystem." In *IMC '13: Proceedings of the 13th ACM Internet Measurement Conference,* edited by Konstantina Papagiannaki, Krishna P. Gummadi, and Craig Partridge, 291–304. Barcelona, Spain: Association for Computing Machinery, Inc. (ACM), October. ISBN: 978-1-4503-1953-9, accessed May 20, 2015. doi:10.1145/2504730.2504755. http://conferences.sigcomm.org/imc/2013/papers/imc257-durumericAemb.pdf.

Dyke, Alan. 2015. "HAR File: Visual Glossary." *MaxCDN One* (blog), March 9. Accessed May 28, 2015. https://www.maxcdn.com/one/visual-glossary/har-file/.

Electronic Frontier Foundation (EFF). 2011. 'HTTPS Now' Campaign Urges Users to Take an Active Role in Protecting Internet Security, April 20. San Francisco, CA, USA. Accessed May 20, 2015. https://www.eff.org/press/archives/2011/04/19-0.

Erman, Jeffrey, Vijay Gopalakrishnan, Rittwik Jana, and Kadangode K. Ramakrishnan. 2013. "Towards a SPDY'ier Mobile Web?" In *CoNEXT '13: Proceedings of the 9th ACM International on Conference on Emerging Networking Experiments and Technologies,* 303–14. Santa Barbara, CA, USA: Association for Computing Machinery (ACM), December. ISBN: 978-1-4503-2101-3, accessed May 22, 2015. doi:10.1145/2535372.2535399. http://conferences.sigcomm.org/co-next/2013/program/p303.pdf.

Everspaugh, Adam, Yan Zhai, Robert Jellinek, Thomas Ristenpart, and Michael Swift. 2014. "Not-So-Random Numbers in Virtualized Linux and the Whirlwind RNG." In *Proceedings of SP '14: 35th IEEE Symposium on Security and Privacy,* 35:559–74. IEEE SP. San Jose, CA, USA: IEEE, May. Slides: http://wisdom.cs.wisc.edu/workshops/spring-14/talks/Adam.pdf, ISBN: 978-1-4799-4686-0, accessed May 20, 2015, doi:10.1109/SP.2014.42. http://www.ieee-security.org/TC/SP2014/papers/Not-So-RandomNumbersinVirtualizedLinuxandtheWhirlwindRNG.pdf.

Everts, Tammy. 2014. "How Does Web Page Speed Affect Conversions? Infographic." *Web Performance Today* (blog), April 9. Accessed May 20, 2015. http://www.webperformancetoday.com/2014/04/09/web-page-speed-affect-conversions-infographic/.

Fielding, Roy T., Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, and Tim Berners-Lee. 1997. *Hypertext Transfer Protocol -- HTTP/1.1.* RFC, Internet Requests for Comments 2068. Status: Proposed Standard; Obsoleted by: RFC 2616; Stream: IETF. RFC Editor, January. Accessed May 20, 2015. doi:10.17487/RFC2068. http://www.rfc-editor.org/rfc/pdfrfc/rfc2068.txt.pdf.

Fielding, Roy T., Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. 1999. *Hypertext Transfer Protocol -- HTTP/1.1.* RFC, Internet Requests for Comments 2616. Status: Draft Standard; Obsoletes: RFC 2068; Obsoleted by: RFC 7230, RFC 7231, RFC 7232, RFC 7233, RFC 7234, RFC 7235; Updated by: RFC 2817, RFC 5785, RFC 6266, RFC 6585; Stream: IETF. RFC Editor, June. Accessed May 20, 2015. doi:10.17487/RFC2616. http://www.rfc-editor.org/rfc/pdfrfc/rfc2616.txt.pdf.

Fielding, Roy T., Yves Lafon, and Julian F. Reschke, eds. 2014. *Hypertext Transfer Protocol (HTTP/1.1): Range Requests.* RFC, Internet Requests for Comments 7233. Status: Proposed Standard; Obsoletes: RFC 2616; Stream: IETF. RFC Editor, June. Accessed May 20, 2015. doi:10.17487/RFC7233. http://www.rfc-editor.org/rfc/pdfrfc/rfc7233.txt.pdf.

Fielding, Roy T., Mark Nottingham, and Julian F. Reschke, eds. 2014. *Hypertext Transfer Protocol (HTTP/1.1): Caching.* RFC, Internet Requests for Comments 7234. Status: Proposed Standard; Obsoletes: RFC 2616; Stream: IETF. RFC Editor, June. Accessed May 20, 2015. doi:10.17487/RFC7234. http://www.rfc-editor.org/rfc/pdfrfc/rfc7234.txt.pdf.

Fielding, Roy T., and Julian F. Reschke, eds. 2014a. *Hypertext Transfer Protocol (HTTP/1.1): Authentication.* RFC, Internet Requests for Comments 7235. Status: Proposed Standard; Obsoletes: RFC 2616; Updates: RFC 2617; Stream: IETF. RFC Editor, June. Accessed May 20, 2015. doi:10.17487/RFC7235. http://www.rfc-editor.org/rfc/pdfrfc/rfc7235.txt.pdf.

———, eds. 2014b. *Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests.* RFC, Internet Requests for Comments 7232. Status: Proposed Standard; Obsoletes: RFC 2616; Stream: IETF. RFC Editor, June. Accessed May 20, 2015. doi:10.17487/RFC7232. http://www.rfc-editor.org/rfc/pdfrfc/rfc7232.txt.pdf.

———, eds. 2014c. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing.* RFC, Internet Requests for Comments 7230. Status: Proposed Standard; Obsoletes: RFC 2145, RFC 2616; Updates: RFC 2817, RFC 2818; Stream: IETF. RFC Editor, June. Accessed May 20, 2015. doi:10.17487/RFC7230. http://www.rfc-editor.org/rfc/pdfrfc/rfc7230.txt.pdf.

———, eds. 2014d. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.* RFC, Internet Requests for Comments 7231. Status: Proposed Standard; Obsoletes: RFC 2616; Updates: RFC 2817; Stream: IETF. RFC Editor, June. Accessed May 20, 2015. doi:10.17487/RFC7231. http://www.rfc-editor.org/rfc/pdfrfc/rfc7231.txt.pdf.

Friedl, Stephan, Andrei Popov, Adam Langley, and Emile Stephan. 2014. *Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension.* RFC, Internet Requests for Comments 7301. Status: Proposed Standard; Stream: IETF. RFC Editor, July. Accessed May 20, 2015. doi:10.17487/RFC7301. http://www.rfc-editor.org/rfc/pdfrfc/rfc7301.txt.pdf.

Fussenegger, Richard. 2014a. *TCP Three-Way Handshake,* October 17. SVG. Wikimedia Commons. Public Domain, accessed May 20, 2015, https://commons.wikimedia.org/wiki/File:TCP_Three-Way_Handshake.svg.

———. 2014b. *TCP Slow-Start and Congestion Avoidance,* October 20. SVG. Wikimedia Commons. GNU General Public License version 3, accessed May 20, 2015, https://commons.wikimedia.org/wiki/File:TCP_Slow-Start_and_Congestion_Avoidance.svg.

———. 2014c. *Symmetric Cryptography,* October 27. SVG. Wikimedia Commons. Public Domain, accessed May 20, 2015, https://commons.wikimedia.org/wiki/File:Symmetric_Cryptography.svg.

———. 2014d. *Asymmetric Cryptography,* October 27. SVG. Wikimedia Commons. Public Domain, accessed May 20, 2015, https://commons.wikimedia.org/wiki/File:Asymmetric_Cryptography.svg.

Fussenegger, Richard. 2014e. *Switch with three computers attached,* October 28. SVG. Wikimedia Commons. CC-BY-SA 4.0 International, accessed May 20, 2015, https://commons.wikimedia.org/wiki/File:Switch_with_three_computers_attached.svg.

———. 2015a. *HTTP Connections,* May 21. SVG. Wikimedia Commons. Public Domain, accessed May 21, 2015, https://commons.wikimedia.org/wiki/File:HTTP_Connections.svg.

———. 2015b. *HTTP/2 Header Compression HPACK,* May 21. SVG. Wikimedia Commons. Public Domain, accessed May 21, 2015, https://commons.wikimedia.org/wiki/File:HTTP_2_Header_Compression_HPACK.svg.

———. 2015c. *Full TLS 1.2 Handshake,* May 21. SVG. Wikimedia Commons. Public Domain, accessed May 21, 2015, https://commons.wikimedia.org/wiki/File:Full_TLS_1.2_Handshake.svg.

———. 2015d. *Abbreviated TLS 1.2 Handshake,* May 21. SVG. Wikimedia Commons. Public Domain, accessed May 21, 2015, https://commons.wikimedia.org/wiki/File:Abbreviated_TLS_1.2_Handshake.svg.

———. 2015e. *Full TLS 1.3 Handshake,* May 21. SVG. Wikimedia Commons. Public Domain, accessed May 21, 2015, https://commons.wikimedia.org/wiki/File:Full_TLS_1.3_Handshake.svg.

———. 2015f. *Full TLS 1.3 Handshake with Mismatched Parameters,* May 21. SVG. Wikimedia Commons. Public Domain, accessed May 21, 2015, https://commons.wikimedia.org/wiki/File:Full_TLS_1.3_Handshake_with_Mismatched_Parameters.svg.

———. 2015g. *Abbreviated TLS 1.3 Handshake,* May 21. SVG. Wikimedia Commons. Public Domain, accessed May 21, 2015, https://commons.wikimedia.org/wiki/File:Abbreviated_TLS_1.3_Handshake.svg.

———. 2015h. *TLS False Start,* May 28. SVG. Wikimedia Commons. Public Domain, accessed May 28, 2015, https://commons.wikimedia.org/wiki/File:TLS_False_Start.svg.

———. 2015i. *TLS Session Ticket Key Lifetime,* May 29. SVG. Wikimedia Commons. Public Domain, accessed May 29, 2015, https://commons.wikimedia.org/wiki/File:TLS_Session_Ticket_Key_Lifetime.svg.

Garrett, Owen. 2015. "How NGINX Plans to Support HTTP/2." *NGINX* (blog), February 25. Accessed May 25, 2015. http://nginx.com/blog/how-nginx-plans-to-support-http2/.

Gidda, Mirren. 2013. "Edward Snowden and the NSA files – timeline." *The Guardian: World News,* June 23. Accessed May 20, 2015. http://www.theguardian.com/world/2013/jun/23/edward-snowden-nsa-files-timeline.

Goldberg, Arthur, Robert Buff, and Andrew Schmitt. 1998a. "Secure Web Server Performance Dramatically Improved by Caching SSL Session Keys." In *WISP '98: The First Workshop on Internet Server Performance.* Madison, WI, USA: University of Wisconsin–Madison (UW), June. Accessed May 20, 2015. http://www.cs.nyu.edu/artg/research/ssl/ssl.doc.

———. 1998b. "A Comparison of HTTP and HTTPS performance," December. Presentation: http://www.cs.nyu.edu/artg/publications/Goldberg_Comparison_of_HTTP_and_HTTPS_Performance_1998_presentation.pdf, accessed May 20, 2015, http://www.cs.nyu.edu/artg/publications/Goldberg_Comparison_of_HTTP_and_HTTPS_Performance_1998.pdf.

Google, Inc. 2009. "SPDY: An experimental protocol for a faster web." *The Chromium Projects* (blog), November 13. Accessed May 20, 2015. https://www.chromium.org/spdy/spdy-whitepaper.

———. 2014. "Marking HTTP As Non-Secure." *The Chromium Projects* (blog), December. Accessed May 20, 2015. https://www.chromium.org/Home/chromium-security/marking-http-as-non-secure.

Gourley, David, and Brian Totty. 2002. *HTTP: The Definitive Guide.* 1st ed. Edited by Linda Mui. Sebastopol, CA, USA: O'Reilly Media, Inc., September. ISBN: 978-1-56592-509-0, accessed May 20, 2015. https://www.safaribooksonline.com/library/view/http-the-definitive/1565925092/.

Graessley, Joshua. 2012. "Session 706: Networking Best Practices." In *Apple Worldwide Developers Conference,* vol. 2012. WWDC. San Francisco, CA, USA: Apple, Inc., June. Accessed May 20, 2015. https://developer.apple.com/videos/wwdc/2012/#706.

Greenberg, Aysylu. 2015. *Devnexus 2015 - Benchmarking: You're Doing It Wrong by Aysylu Greenberg.* Length: 51:34. DZone, Inc. MP4 (Stream). Slides: http://www.slideshare.net/aysylu/benchmarking-devnexus-2015, accessed May 21, https://www.youtube.com/watch?v=s3NSTWYNNNo.

Grigorik, Ilya. 2012. "How can I benchmark my website with SPDY support?: Answer 393135." *Server Fault* (qas) (May 27). Accessed May 23, 2015. http://serverfault.com/a/393135.

———. 2013a. *High Performance Browser Networking: What every web developer should know about networking and web performance.* 1st ed. Edited by Courtney Nash. Sebastopol, CA, USA: O'Reilly Media, Inc., September 9. The online version contains several updates which are not included in the print nor e-book versions, ISBN: 978-1-4493-4476-4, accessed May 20, 2015, http://chimera.labs.oreilly.com/books/1230000000545.

Grigorik, Ilya. 2013b. "Optimizing NGINX TLS Time To First Byte (TTTFB)." *igvita.com: a Goal is a Dream with a Deadline* (blog), December 16. Accessed May 27, 2015. https://www.igvita.com/2013/12/16/optimizing-nginx-tls-time-to-first-byte/.

———. 2014. "Is TLS Fast Yet?: It can be. Making TLS fast(er)… the nuts and bolts. TLS all the tubes!" In *O'Reilly Velocity Conference* 2014.

———. 2015. *Figure 12-6. HPACK: Header Compression for HTTP/2,* April 12. PNG, 2,974px × 968px. O'Reilly Media, Inc., Sebastopol, CA, USA. Accessed May 21, 2015. http://orm-chimera-prod.s3.amazonaws.com/1230000000545/images/hpbn_1205.png.

Gutmann, Peter. 2014. *Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS).* RFC, Internet Requests for Comments 7366. Status: Proposed Standard; Stream: IETF. RFC Editor, September. Accessed May 20, 2015. doi:10.17487/RFC7366. http://www.rfc-editor.org/rfc/pdfrfc/rfc7366.txt.pdf.

Hecht, Eugene. 2002. *Optics.* International Edition. 4th ed. Edited by Adam Black. Reading, MA, USA: Addison-Wesley. ISBN: 978-0-321-18878-6.

Hickman, Kipp E. B., and Taher ElGamal (Netscape Communication Corporation). 1997. Secure Socket Layer Application Program Apparatus And Method. http://www.lens.org/lens/patent/US_5657390_A US 5657390 A (Mountain View, CA, USA), filed August 25, 1995, and issued August 12, 1997. Accessed May 20, 2015. https://www.google.com/patents/US5657390A.

Hickman, Kipp E.B., and Taher ElGamal. 1995. *The SSL Protocol.* Working Draft, Internet-Draft draft-hickman-netscape-ssl-00. State: Expired. June 21. Accessed May 20, 2015. https://tools.ietf.org/pdf/draft-hickman-netscape-ssl-00.pdf.

Hodges, Jeff, Collin Jackson, and Adam Barth. 2012. *HTTP Strict Transport Security (HSTS).* RFC, Internet Requests for Comments 6797. Status: Proposed Standard; Stream: IETF. RFC Editor, November. Accessed May 20, 2015. doi:10.17487/RFC6797. http://www.rfc-editor.org/rfc/pdfrfc/rfc6797.txt.pdf.

Hoelzle, Urs, and Bill Coughran. 2009. "Let's make the web faster." *Google Official Blog* (blog), June 23. Accessed May 20, 2015. http://googleblog.blogspot.co.at/2009/06/lets-make-web-faster.html.

Hoffman-Andrews, Jacob. 2013. "Forward Secrecy at Twitter." *The Official Twitter Blog* (blog), November 22. Accessed May 29, 2015. https://blog.twitter.com/2013/forward-secrecy-at-twitter.

Holland, Martin. 2014. "Was bisher geschah: Der NSA-Skandal im Jahr 1 nach Snowden" [in German]. *heise online* (Hannover, Germany) (June 5). Accessed May 20, 2015. http://heise.de/-2214943.

Hornig, Charles. 1984. *A Standard for the Transmission of IP Datagrams over Ethernet Networks.* RFC, Internet Requests for Comments 894. Status: Internet Standard; Stream: Legacy. RFC Editor, April. Accessed May 20, 2015. doi:10.17487/RFC0894. http://www.rfc-editor.org/rfc/pdfrfc/rfc894.txt.pdf.

HTTP Archive. 2015. *Interesting Stats.* Statistic May 15 2015. HTTP Archive, May 15. Accessed May 28, 2015. http://httparchive.org/interesting.php?a=All&l=May%2015%202015#protocol.

Huffman, David. 1952. "A Method for the Construction of Minimum-Redundancy Codes." *Proceedings of the IRE* 40 (9): 1098–101. ISSN: 0096-8390, accessed May 20, 2015. doi:10.1109/JRPROC.1952.273898. http://compression.ru/download/articles/huff/huffman_1952_minimum-redundancy-codes.pdf.

Inacon GmbH. 2010. *Slow Start and Congestion Avoidance in Operation.* GIF, 784px × 518px. Inacon GmbH, Karlsruhe, Germany. License: http://www.inacon.de/ph/data/copying.htm, accessed May 21, 2015, http://www.inacon.de/ph/data/images/Slow-Start-and-Congestion-Avoidance-in-Operation_OS.gif.

Internet Engineering Steering Group (IESG) Secretary. 2007. "WG Review: HyperText Transport Protocol Bis (httpbis)." October 11. Accessed May 20, 2015. http://lists.w3.org/Archives/Public/ietf-http-wg/2007OctDec/0028.html.

Ishigure, Takaaki, Eisuke Nihei, and Yasuhiro Koike. 1996. "Optimum refractive-index profile of the graded-index polymer optical fiber, toward gigabit data links." *Applied Optics,* no. 12, 2048–53. ISSN: 0003-6935, accessed May 20, 2015. doi:10.1364/AO.35.002048. http://www.ishigure.appi.keio.ac.jp/PDFs/AO%20profile%20toward%20Gig.pdf.

Jacobson, Van, and Robert T. Braden. 1988. *TCP extensions for long-delay paths.* RFC, Internet Requests for Comments 1072. Status: Historic; Obsoleted by: RFC 1323, RFC 2018, RFC 6247; Stream: Legacy. RFC Editor, October. Accessed May 20, 2015. doi:10.17487/RFC1072. http://www.rfc-editor.org/rfc/pdfrfc/rfc1072.txt.pdf.

Jacobson, Van, Robert T. Braden, and Dave Borman. 1992. *TCP Extensions for High Performance.* RFC, Internet Requests for Comments 1323. Status: Proposed Standard; Obsoletes: RFC 1072, RFC 1185; Obsoleted by: RFC 7323; Stream: IETF. RFC Editor, May. Accessed May 20, 2015. doi:10.17487/RFC1323. http://www.rfc-editor.org/rfc/pdfrfc/rfc1323.txt.pdf.

Jacobson, Van, and Michael J. Karels. 1988. "Congestion Avoidance and Control." ACM SIGCOMM. Revised, *Computer Communication Review* (New York, NY, USA) 18 (4): 314–29. ISSN: 0146-4833, accessed May 20, 2015. doi:10.1145/52325.52356. http://ee.lbl.gov/papers/congavoid.pdf.

Jain, Raj. 1986. "A Timeout-Based Congestion Control Scheme for Window Flow-Controlled Networks." *IEEE Journal on Selected Areas in Communications* 4, no. 7 (January): 1162–67. ISSN: 0733-8716, accessed May 20, 2015. doi:10.1109/JSAC.1986.1146431. http://www.researchgate.net/profile/Raj_Jain3/publication/220480888_A_Timeout_Based_Congestion_Control_Scheme_for_Window_Flow-_Controlled_Networks/links/0912f505b5089e9901000000.pdf.

Josefsson, Simon, and Manuel Pégourié-Gonnard. 2014. *Curve25519 for ephemeral key exchange in Transport Layer Security (TLS).* Working Draft, Internet-Draft draft-josefsson-tls-curve25519-06. State: Expired. September 11. Accessed May 20, 2015. https://tools.ietf.org/pdf/draft-josefsson-tls-curve25519-06.pdf.

Kamp, Poul-Henning. 2015. "HTTP/2.0: The IETF is Phoning It in." *Communications of the ACM* (New York, NY, USA), February, no. 3, 40–42. ISSN: 0001-0782, accessed May 20, 2015. doi:10.1145/2717515. http://queue.acm.org/detail.cfm?id=2716278.

Kant, Krishna, Ravishankar Iyer, and Prasant Mohapatra. 2000. "Architectural Impact of Secure Socket Layer on Internet Servers." In *ICCD 2000: IEEE International Conference on Computer Design,* edited by A. Denise Williams, 7–14. Austin, TX, USA: IEEE Computer Society, September. ISBN: 0-7695-0801-4, accessed May 21, 2015. doi:10.1109/ICCD.2000.878263. http://spirit.cs.ucdavis.edu/pubs/conf/iccd00.pdf.

Kravets, David. 2014. "Comcast Wi-Fi Serving Self-Promotional Ads via JavaScript Injection: The practice raises security, net neutrality issues as FCC mulls Internet reforms." Edited by Ken Fisher. *Ars Technica* (New York City, NY, USA) (September 8).

Krawczyk, Hugo. 2001. "The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?)" In *Advances in Cryptology — CRYPTO 2001: 21st Annual International Cryptology Conference,* Santa Barbara, CA, USA, August 19–23, 2001. Proceedings, edited by Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, and Joe Kilian, 2139:310–31. Lecture Notes in Computer Science (LNCS). Santa Barbara, CA, USA: Springer-Verlag Berlin Heidelberg, August. ISBN: 978-3-540-42456-7, accessed May 20, 2015. doi:10.1007/3-540-44647-8_19. http://eprint.iacr.org/2001/045.

Krishnamurthy, Balachander, Jeffrey C. Mogul, and David M. Kristol. 1999. "Key differences between HTTP/1.0 and HTTP/1.1." Edited by Philip H., Jr. Enslow. *Computer Networks: The International Journal of Computer and Telecommunications Networking* (New York, NY, USA) 31, nos. 11-16 (May 17): 1737–51. ISSN: 1389-1286, accessed May 20, 2015. doi:10.1016/S1389-1286(99)00008-0. http://www.id.uzh.ch/home/mazzo/reports/www8conf/2136/pdf/pd1.pdf.

Krovetz, Ted, and Wei Dai. 2007. *VMAC: Message Authentication Code using Universal Hashing.* Working Draft, Internet-Draft draft-krovetz-vmac-01. State: Expired. April 23. Accessed May 20, 2015. https://tools.ietf.org/pdf/draft-krovetz-vmac-01.pdf.

Kurose, James F., and Keith W. Ross. 2013. *Computer Networking: A Top-Down Approach.* 6th ed. Upper Saddle River, NJ, USA: Addison-Wesley / Pearson Education Inc. ISBN: 978-0-13-285620-1, accessed May 20, 2015. http://wps.pearsoned.com/ecs_kurose_compnetw_6/.

Al-Kuwari, Saif, James H. Davenport, and Russell J. Bradford. 2010. "Cryptographic Hash Functions: Recent Design Trends and Security Notions." In *Short Paper Proceedings of 6th China International Conference on Information Security and Cryptology,* 133–50. Inscrypt. Shanghai, ZH: Science Press of China. ISBN: 978-3-642-21517-9, accessed May 20, 2015. http://opus.bath.ac.uk/20815/1/HashFunction_Survey_FINAL_221011-1.pdf.

Landley, Rob. 2005. "ramfs, rootfs, and initramfs." October 17. Accessed May 29, 2015. https://www.kernel.org/doc/Documentation/filesystems/ramfs-rootfs-initramfs.txt.

Langley, Adam. 2012. "False Start's Failure." *ImperialViolet* (blog) (April 11). Accessed May 27, 2015. https://www.imperialviolet.org/2012/04/11/falsestart.html.

———. 2013. "How to Botch TLS Forward Secrecy." *ImperialViolet* (blog) (June 27). Accessed May 28, 2015. https://www.imperialviolet.org/2013/06/27/botchingpfs.html.

Langley, Adam, Nagendra Modadugu, and Bodo Moeller. 2015. *Transport Layer Security (TLS) False Start.* Work in Progress, Internet-Draft draft-ietf-tls-falsestart-00. May 10. Accessed May 27, 2015. https://tools.ietf.org/pdf/draft-ietf-tls-falsestart-00.pdf.

Law, Laurie, Alfred Menezes, Minghua Qu, Jerry Solinas, and Scott Vanstone. 1998. *An Efficient Protocol for Authenticated Key Agreement.* CORR 98-05. Centre for Applied Cryptographic Research (CACR), August 28. Accessed May 20, 2015. http://cacr.uwaterloo.ca/techreports/1998/corr98-05.pdf.

Leiba, Barry, and Cindy Morgan. 2012. *Hypertext Transfer Protocol WG Charter.* IETF Secretariat, October 2. Accessed May 20, 2015. http://datatracker.ietf.org/doc/charter-ietf-httpbis/withmilestones-07.txt.

Luko, Stephen N. 2011. "What Are Z-Scores?" *ASTM Standardization News (SN)* (Philadelphia, PA, USA) 39, no. 2 (April). ISSN: 1094-4656, accessed May 31, 2015. http://www.astm.org/SNEWS/MA_2011/datapoints_ma11.html.

McManus, Patrick, and Nick Hurley. 2014. "Networking/http2." *Mozilla Wiki* (wiki), August 5. Accessed May 20, 2015. https://wiki.mozilla.org/index.php?title=Networking/http2&oldid=1003041.

Medvinsky, Ari, and Matthew Hur. 1999. *Addition of Kerberos Cipher Suites to Transport Layer Security (TLS).* RFC, Internet Requests for Comments 2712. Status: Proposed Standard; Stream: IETF. RFC Editor, October. Accessed May 20, 2015. doi:10.17487/RFC2712. http://www.rfc-editor.org/rfc/pdfrfc/rfc2712.txt.pdf.

Meier, J.D., Carlos Farre, Prashant Bansode, Scott Barber, and Dennis Rea. 2007. *Performance Testing Guidance for Web Applications.* Microsoft Patterns & Pracitces I. Redmond, WA, USA: Microsoft Press, August 27. ISBN: 978-0-7356-2570-9, accessed May 23, 2015. http://perftestingguide.codeplex.com/.

Mellia, Marco, Andrea Carpani, and Renato Lo Cigno. 2003. "TStat: TCP STatistic and Analysis Tool." In *Quality of Service in Multiservice IP Networks: Second International Workshop, QoS-IP 2003,* Milano, Italy, February 24–26, 2003. Proceedings, edited by Marco A. Marsan, Giorgio Corazza, Marco Listanti, and Aldo Roveri, 2601:145–57. Lecture Notes in Computer Science (LNCS). Milano, Italy: Springer-Verlag Berlin Heidelberg, February 17. ISBN: 978-3-540-00604-6, accessed May 28, 2015. doi:10.1007/3-540-36480-3_11. http://www.tlc-networks.polito.it/mellia/papers/tstat_cn.ps.

Merkle, Johannes, and Manfred Lochter. 2013. *Elliptic Curve Cryptography (ECC) Brainpool Curves for Transport Layer Security (TLS).* RFC, Internet Requests for Comments 7027. Status: Informational; Updates: RFC 4492; Stream: IETF. RFC Editor, October. Accessed May 20, 2015. doi:10.17487/RFC7027. http://www.rfc-editor.org/rfc/pdfrfc/rfc7027.txt.pdf.

Meyer, Christopher. 2014. "20 Years of SSL/TLS Research An Analysis of the Internet's Security Foundation." PhD diss., Ruhr University. Accessed May 20, 2015. http://www-brs.ub.ruhr-uni-bochum.de/netahtml/HSS/Diss/MeyerChristopher/diss.pdf.

Miller, Robert B. 1968. "Response time in man-computer conversational transactions." In *Proceedings of AFIPS '68: American Federation of Information Processing Societies,* 32:267–77. AFIPS. Washington D.C., USA: Thomson Book Company. Accessed May 20, 2015. doi:10.1145/1476589.1476628. http://theixdlibrary.com/pdf/Miller1968.pdf.

Möller, Bodo, Thai Duong, and Krzysztof Kotowicz. 2014. "This POODLE Bites: Exploiting The SSL 3.0 Fallback" (September). Accessed May 20, 2015. https://www.openssl.org/~bodo/ssl-poodle.pdf.

Nagle, John. 1984. *Congestion Control in IP/TCP Internetworks.* RFC, Internet Requests for Comments 896. Status: Unknown; Stream: Legacy. RFC Editor, January. Accessed May 20, 2015. doi:10.17487/RFC0896. http://www.rfc-editor.org/rfc/pdfrfc/rfc896.txt.pdf.

Nagle, John. 1985. *On Packet Switches With Infinite Storage.* RFC, Internet Requests for Comments 970. Status: Unknown; Stream: Legacy. RFC Editor, December. Accessed May 20, 2015. doi:10.17487/RFC0970. http://www.rfc-editor.org/rfc/pdfrfc/rfc970.txt.pdf.

Nah, Fiona Fui-Hoon. 2004. "A study on tolerable waiting time: how long are Web users willing to wait?" *Behaviour & Information Technology,* no. 3, 153–63. ISSN: 0144-929X, accessed May 20, 2015. doi:10.1080/01449290410001669914. http://cba.unl.edu/research/articles/548/download.pdf.

Naylor, David, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio Munafò, Konstantina Papagiannaki, and Peter Steenkiste. 2014. "The Cost of the 'S' in HTTPS." In *CoNEXT '14: Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies,* 133–40. Sydney, Australia: Association for Computing Machinery (ACM), December. Slides: http://www.cs.cmu.edu/~dnaylor/CostOfTheS_slides.pdf, ISBN: 978-1-4503-3279-8, accessed May 20, 2015, doi:10.1145/2674005.2674991. https://www.cs.cmu.edu/~dnaylor/CostOfTheS.pdf.

Netcraft. 2013. "April 2013 Web Server Survey." *Netcraft News* (blog) (April 2). Accessed May 20, 2015. http://news.netcraft.com/archives/2013/04/02/april-2013-web-server-survey.html.

Nottingham, Mark. 2011a. *Making HTTP Pipelining Usable on the Open Web.* Working Draft, Internet-Draft draft-nottingham-http-pipeline-01. State: Expired. March 13. Accessed May 20, 2015. https://tools.ietf.org/pdf/draft-nottingham-http-pipeline-01.pdf.

———. 2011b. "On HTTP Load Testing." *mnot's blog* (blog), May 18. Accessed May 23, 2015. https://www.mnot.net/blog/2011/05/18/http_benchmark_rules.

Odvarko, Jan, Arvind Jain, and Andy Davies, eds. 2012. *HTTP Archive (HAR) format: Editor's Draft August 14, 2012.* Technical report. W3C, August 14. Accessed May 24, 2015. https://dvcs.w3.org/hg/webperf/raw-file/tip/specs/HAR/Overview.html.

Oppliger, Rolf. 2009. *SSL and TLS: Theory and Practice.* Artech House information security and privacy series. Norwood, MA, USA: Artech House. ISBN: 978-1-59693-447-4.

O'Reilly Velocity Conference. 2014. Vol. 2014. Velocity. Santa Clara, CA, USA: O'Reilly Media, Inc., June. Accessed May 20, 2015. http://velocityconf.com/velocity2014.

Organization for Economic Co-operation and Development (OECD). 2008. *Developments in Fibre Technologies and Investment.* OECD Digital Economy Papers No. 142. OECD Publishing, May 3. doi:10.1787/230526125804.

Paar, Christof, and Jan Pelzl. 2010. *Understanding Cryptography: A Textbook for Students and Practitioners.* Berlin/Heidelberg, Germany: Springer-Verlag Berlin Heidelberg. ISBN: 978-3-642-04100-6.

Padhye, Jitu, and Henrik F. Nielsen. 2012. *A Comparison of SPDY and HTTP Performance.* TechReport MSR-TR-2012-102. Microsoft, Corp., July. Accessed February 22, 2015. http://research.microsoft.com/apps/pubs?id=170059.

Peon, Roberto, and Herve Ruellan. 2015. *HPACK: Header Compression for HTTP/2.* RFC, Internet Requests for Comments 7541. Status: Proposed Standard; Stream: IETF. RFC Editor, May. Accessed May 20, 2015. doi:10.17487/RFC7541. http://www.rfc-editor.org/rfc/pdfrfc/rfc7541.txt.pdf.

Podjarny, Guy. 2010. "17 Statistics to Sell Web Performance Optimization." *Guy's Pod: Quick Thoughts on Performance* (blog) (December 16). Accessed May 20, 2015. http://www.guypo.com/business/17-statistics-to-sell-web-performance-optimization/.

———. 2012. "Not as SPDY as You Thought." *Guy's Pod: Quick Thoughts on Performance* (blog) (June 12). Accessed May 22, 2015. http://www.guypo.com/not-as-spdy-as-you-thought/.

Popov, Andrei. 2015. *Prohibiting RC4 Cipher Suites.* RFC, Internet Requests for Comments 7465. Status: Proposed Standard; Updates: RFC 5246, RFC 4346, RFC 2246; Stream: IETF. RFC Editor, February. Accessed May 20, 2015. doi:10.17487/RFC7413. http://www.rfc-editor.org/rfc/pdfrfc/rfc7465.txt.pdf.

Radhakrishnan, Sivasankar, Yuchung Cheng, Jerry Chu, Arvind Jain, and Barath Raghavan. 2011. "TCP Fast Open." In *Proceedings of CoNEXT '11: The 7th International Conference on emerging Networking EXperiments and Technologies,* 2011:1–12. CoNEXT. New York, NY, USA: ACM. ISBN: 978-1-4503-1041-3, accessed May 20, 2015. doi:10.1145/2079296.2079317. http://conferences.sigcomm.org/co-next/2011/papers/1569470463.pdf.

Ramaswamy, Ramaswamy, Ning Weng, and Tilman Wolf. 2004. "Characterizing Network Processing Delay." In *Proceedings of GLOBECOM '04: Global Telecommunications Conference,* 3:1629–34. GLOBECOM. Piscataway, NJ, USA: IEEE. ISBN: 978-0-7803-8794-2, accessed May 20, 2015. doi:10.1109/GLOCOM.2004.1378257. http://www.ecs.umass.edu/ece/wolf/pubs/globecom2004.pdf.

Reschke, Julian F. 2014a. *Initial Hypertext Transfer Protocol (HTTP) Authentication Scheme Registrations.* RFC, Internet Requests for Comments 7236. Status: Informational; Stream: IETF. RFC Editor, June. Accessed May 20, 2015. doi:10.17487/RFC7236. http://www.rfc-editor.org/rfc/pdfrfc/rfc7236.txt.pdf.

Reschke, Julian F., ed. 2014b. *Initial Hypertext Transfer Protocol (HTTP) Method Registrations.* RFC, Internet Requests for Comments 7237. Status: Informational; Stream: IETF. RFC Editor, June. Accessed May 20, 2015. doi:10.17487/RFC7237. http://www.rfc-editor.org/rfc/pdfrfc/rfc7237.txt.pdf.

———. 2015. *The Hypertext Transfer Protocol Status Code 308 (Permanent Redirect).* RFC, Internet Requests for Comments 7538. Status: Proposed Standard; Obsoletes: RFC 7238; Stream: IETF. RFC Editor, April. Accessed May 20, 2015. doi:10.17487/RFC7538. http://www.rfc-editor.org/rfc/pdfrfc/rfc7538.txt.pdf.

Rescorla, Eric. 2015a. *New Handshake Flows for TLS 1.3.* Working Draft, Internet-Draft draft-rescorla-tls13-new-flows-01. State: Expired. February 17. Accessed May 20, 2015. https://tools.ietf.org/pdf/draft-rescorla-tls13-new-flows-01.pdf.

———. 2015b. *The Transport Layer Security (TLS) Protocol Version 1.3.* Work in Progress, Internet-Draft draft-ietf-tls-tls13-05. March 9. Accessed May 20, 2015. https://tools.ietf.org/pdf/draft-ietf-tls-tls13-05.pdf.

Ristić, Ivan. 2014. *Bulletproof SSL and TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Wep Applications.* London, GB: Feisty Duck Ltd. ISBN: 978-1-907117-04-6, accessed May 20, 2015. https://www.feistyduck.com/books/bulletproof-ssl-and-tls/.

Rivest, Ron L., Adi Shamir, and Leonard Adleman. 1978. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." *Communications of the ACM* 21 (2): 120–26. Revised version: https://people.csail.mit.edu/rivest/Rsapaper.pdf, ISSN: 00010782, accessed May 20, 2015, doi:10.1145/359340.359342. https://www.cs.drexel.edu/~jjohnson/sp03/cs300/lectures/p120-rivest.pdf.

Rizzo, Juliano, and Thai Duong. 2012. "The CRIME." In *Ekoparty Security Conference 8th Edition,* 2012:1–43. Ekoparty. Buenos Aires, AR, September. Accessed May 20, 2015. http://www.ekoparty.org/archive/2012/CRIME_ekoparty2012.pdf.

Roskind, Jim, ed. 2013. *QUIC: Design Document and Specification Rationale.* Google, Inc. Accessed May 20, 2015. https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/.

Salowey, Joseph, Abhijit Choudhury, and David McGrew. 2008. *AES Galois Counter Mode (GCM) Cipher Suites for TLS.* RFC, Internet Requests for Comments 5288. Status: Proposed Standard; Stream: IETF. RFC Editor, August. Accessed May 20, 2015. doi:10.17487/RFC5288. http://www.rfc-editor.org/rfc/pdfrfc/rfc5288.txt.pdf.

Salowey, Joseph, Harry Zhou, Pasi Eronen, and Hannes Tschofenig. 2006. *Transport Layer Security (TLS) Session Resumption without Server-Side State.* RFC, Internet Requests for Comments 4507. Status: Proposed Standard; Obsoleted by: RFC 5077; Stream: IETF. RFC Editor, May. Accessed May 20, 2015. doi:10.17487/RFC4507. http://www.rfc-editor.org/rfc/pdfrfc/rfc4507.txt.pdf.

Salowey, Joseph, Harry Zhou, Pasi Eronen, and Hannes Tschofenig. 2008. *Transport Layer Security (TLS) Session Resumption without Server-Side State.* RFC, Internet Requests for Comments 5077. Status: Proposed Standard; Obsoletes: RFC 4507; Stream: IETF. RFC Editor, January. Accessed May 20, 2015. doi:10.17487/RFC5077. http://www.rfc-editor.org/rfc/pdfrfc/rfc5077.txt.pdf.

Santesson, Stefan, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. 2013. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP.* RFC, Internet Requests for Comments 6960. Status: Proposed Standard; Obsoletes: RFC 2560, RFC 6277; Updates: RFC 5912; Stream: IETF. RFC Editor, June. Accessed May 20, 2015. doi:10.17487/RFC6960. http://www.rfc-editor.org/rfc/pdfrfc/rfc6960.txt.pdf.

Sawall, Achim. 2014. "Internet Provider bietet symmetrische 1 GBit/s für 53 Euro: Ungedrosselt" [in German]. *golem.de: IT-News für Profis* (Berlin, Germany) (May 22). Accessed May 21, 2015. http://www.golem.de/1405/106667.html.

Schneier, Bruce, and David Wagner. 1996. "Analysis of the SSL 3.0 Protocol." In *Proceedings of EC '96: The Second USENIX Workshop on Electronic Commerce,* 2:29–40. EC. Oakland, CA, USA: USENIX Press. Accessed May 20, 2015. https://www.schneier.com/paper-ssl.pdf.

Sebayang, Andreas. 2013. "Vom Alptraum zur schlimmen Realität: 30C3 und NSA" [in German]. *golem.de: IT-News für Profis* (Berlin, Germany) (December 27). Accessed May 21, 2015. http://www.golem.de/1312/103594.html.

Selvi, Jose. 2014. "Bypassing HTTP Strict Transport Security." In *Black Hat Europe 2014.* Amsterdam, Netherlands: UBM Tech, October. Accessed May 20, 2015. https://www.blackhat.com/docs/eu-14/materials/eu-14-Selvi-Bypassing-HTTP-Strict-Transport-Security-wp.pdf.

Seow, Steven C. 2008. *Designing and Engineering Time: The Psychology of Time Perception in Software.* 1st ed. Upper Saddle River, NJ, USA: Addison-Wesley, May. ISBN: 978-0-321-50918-5, accessed May 20, 2015.

Servy, Hervé. 2013. "Evaluating the Performance of SPDY-enabled Web Servers." *Neotys Blog: Everything about Performance, Load and Stress Testing for your Web and Mobile Applications* (blog), July 17. Accessed May 22, 2015. https://www.neotys.com/blog/performance-of-spdy-enabled-web-servers/.

Sheffer, Yaron, Ralph Holz, and Peter Saint-Andre. 2015. *Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS).* RFC, Internet Requests for Comments 7525. Status: Best Current Practice; Stream: IETF. RFC Editor, May. Accessed May 20, 2015. doi:10.17487/RFC7525. http://www.rfc-editor.org/rfc/pdfrfc/rfc7525.txt.pdf.

Shneiderman, Ben. 1984. "Response time and display rate in human performance with computers." *ACM Computing Surveys* 16 (3): 265–85. ISSN: 0360-0300, accessed May 20, 2015. doi:10.1145/2514.2517. https://www.cs.umd.edu/~ben/papers/Shneiderman1984Response.pdf.

Shor, Peter W. 1994. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring." In *Proceedings of FOCS '94: 35th Annual Symposium on Foundations of Computer Science,* 35:124–34. FOCS. Santa Fe, NM, USA: IEEE Computer Society. ISBN: 0-8186-6580-7, accessed May 20, 2015. doi:10.1109/SFCS.1994.365700. http://www.computer.org/csdl/proceedings/focs/1994/6580/00/0365700.pdf.

Smart, Nigel P., ed. 2013. *Algorithms, Key Sizes and Parameters Report – 2013 Recommendations.* Technical report P/18/12/TCD Lot 2. Version 1.0. European Union Agency for Network and Information Security Agency (ENISA), October 29. Accessed May 20, 2015. http://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/algorithms-key-sizes-and-parameters-report/at_download/fullReport.

Smith, Peter. 2012. *Professional Website Performance: Optimizing the Front-End and Back-End.* 1st ed. Wrox Programmer to Programmer. Hoboken, NJ, USA: John Wiley & Sons, Inc., November. ISBN: 978-1-118-55172-1.

Sniffen, Brian. 2014. "TLS Is Not for Privacy." *Sniffen Packets: With a name like Sniffen, it's got to smell good* (blog), May 16. Accessed May 28, 2015. http://weblog.evenmere.org/blog/2014/05/16/tls-is-not-for-privacy/.

Souders, Steve. 2009. *Even Faster Web Sites: Performance Best Practices for Web Developers.* 1st ed. Sebastopol, CA, USA: O'Reilly Media, Inc. ISBN: 978-0-596-52230-8.

Stefanov, Stoyan, ed. 2012. *Web Performance Daybook, Volume Two: Techniques & Tips for Optimizing Website Performance.* 1st ed. Sebastopol, CA, USA: O'Reilly Media, Inc. ISBN: 978-1-4493-3291-4.

Stenberg, Daniel. 2014. "libressl vs boringssl for curl." *daniel.haxx.se: tech, open source and networking* (blog) (August 5). Accessed May 20, 2015. http://daniel.haxx.se/blog/2014/08/05/libressl-vs-boringssl-for-curl/.

———. 2015. "TLS in HTTP/2." *daniel.haxx.se: tech, open source and networking* (blog) (March 6). Accessed May 20, 2015. http://daniel.haxx.se/blog/2015/03/06/tls-in-http2/.

Sundaresan, Srikanth, Walter de Donato, Nick Feamster, Renata Teixeira, Sam Crawford, and Antonio Pescapè. 2011. "Broadband Internet Performance: A View From the Gateway." ACM SIGCOMM. *Computer Communication Review: Proceedings of SIGCOMM 2011 & Best Papers of the Co-Located Workshops* (New York, NY, USA) 41, no. 4 (October): 134–45. ISSN: 0146-4833, accessed May 20, 2015. doi:10.1145/2043164.2018452. http://conferences.sigcomm.org/sigcomm/2011/papers/sigcomm/p134.pdf.

Tan, Jiaqi, and Jayvardhan Nahata. 2013. *PETAL: Preset Encoding Table Information Leakage.* CMU-PDL 13-106. Parallel Data Lab (PDL), April. Accessed May 20, 2015. http://www.pdl.cmu.edu/PDL-FTP/associated/CMU-PDL-13-106.pdf.

Thoma, Jörg. 2014. "Perma-Cookie in manipulierten Datenpaketen: Verizon" [in German]. *golem.de: IT-News für Profis* (Berlin, Germany) (October 28). Accessed May 21, 2015. http://www.golem.de/1410/110141.html.

Thomas, Stephen A. 2000. *SSL & TLS Essentials: Securing the Web.* Hoboken, NJ, USA: John Wiley & Sons, Inc. ISBN: 978-0-471-38354-3.

Thomson, Martin, Salvatore Loreto, and Greg Wilkins. 2012. *Hypertext Transfer Protocol (HTTP) Keep-Alive Header.* Working Draft, Internet-Draft draft-thomson-hybi-http-timeout-03. State: Expired. July 16. Accessed May 20, 2015. https://tools.ietf.org/pdf/draft-thomson-hybi-http-timeout-03.pdf.

Turner, Sean, and Cindy Morgan. 2014. *Transport Layer Security WG Charter.* IETF Secretariat, February 7. Accessed May 20, 2015. http://datatracker.ietf.org/doc/charter-ietf-tls/withmilestones-05.txt.

Turner, Sean, and Tim Polk. 2011. *Prohibiting Secure Sockets Layer (SSL) Version 2.0.* RFC, Internet Requests for Comments 6176. Status: Proposed Standard; Updates: RFC 2246, RFC 4346, RFC 5246; Stream: IETF. RFC Editor, March. Accessed May 20, 2015. doi:10.17487/RFC6176. http://www.rfc-editor.org/rfc/pdfrfc/rfc6176.txt.pdf.

United Nations Economic Commission for Europe (UNECE). 2009. *Making Data Meaningful Part 2: A Guide to Presenting Statistics.* Guide ECE/CES/STAT/NONE/2009/3. United Nations (UN), March. Accessed May 25, 2015. http://www.unece.org/fileadmin/DAM/stats/documents/writing/MDM_Part2_English.pdf.

Wang, Xiao S., Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. 2014. "How Speedy is SPDY?" In *NSDI '14: Processings of the 11th USENIX Symposium on Networked Systems Design and Implementation,* 387–99. Seattle, WA, USA: USENIX Association, April. ISBN: 978-1-931971-09-6, accessed May 22, 2015. https://www.usenix.org/conference/nsdi14/technical-sessions/wang.

Welsh, Matt. 2011. "Measuring the Mobile Web Is Hard." *Volatile and Decentralized: The Internet Has Nowhere To Hide* (blog), August 4. Accessed May 29, 2015. http://matt-welsh.blogspot.com/2011/08/measuring-mobile-web-is-hard.html.

Welsh, Matt, Ben Greenstein, and Michael Piatek. 2012. "SPDY Performance on Mobile Networks." *Google Developers,* April 30. Accessed May 22, 2015. https://developers.google.com/speed/articles/spdy-for-mobile.

Wilk, Alyssa, Ryan Hamilton, and Ian Swett. 2015. "A QUIC update on Google's experimental transport." *Chromium Blog: News and developments from the open source browser project* (blog) (April 17). Accessed May 20, 2015. https://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html.

Wilson, Kathleen. 2014. "Phasing out Certificates with 1024-bit RSA Keys." *Mozilla Security Blog* (blog) (September 8). Accessed May 20, 2015. https://blog.mozilla.org/security/2014/09/08/phasing-out-certificates-with-1024-bit-rsa-keys/.

———. 2015. "Phase 2: Phasing out Certificates with 1024-bit RSA Keys." *Mozilla Security Blog* (blog) (January 28). Accessed May 20, 2015. https://blog.mozilla.org/security/2015/01/28/phase-2-phasing-out-certificates-with-1024-bit-rsa-keys/.

Winkler, Martin. 2014. "Wie HTTP 2.0 das Surfen beschleunigt" [in German]. *heise Netze* (Hannover, Germany) (April 7). Accessed May 20, 2015. http://heise.de/-2164146.

Zhao, Li, Ravi Iyer, Srihari Makineni, and Laxmi Bhuyan. 2005. "Anatomy and Performance of SSL Processing." In *ISPASS 2005: IEEE International Symposium on Performance Analysis of Systems and Software,* 197–206. Austin, TX, USA: IEEE Computer Society, March. ISBN: 0-7803-8965-4, accessed May 20, 2015. doi:10.1109/ISPASS.2005.1430574. http://www.cs.ucr.edu/~bhuyan/papers/ssl.pdf.

Glossary

a denotes the message authentication code in equations. 37, 38

Akamai Technologies is one of the largest CDNs worldwide serving, among others, Facebook's monthly 1.32 billion users with static resources. 9, 10, 97

Alexa Internet is a subsidiary of Amazon.com Incorporation and offers commercial web traffic analysis services. The company and their service is best known for its ranking data and the free online publication of the "Alexa Top Sites": http://www.alexa.com/topsites 11, 63, 64, 82, 139

Amazon.com Incorporation is the largest Internet-based retailer in the United States with subsidiaries around the world. The company offers various Internet related services, like Alexa Internet and the Amazon Web Services (AWS), and serves content to billions of users around the world. 29, 108, 139

b denotes bandwidth in equations. 15

Blink is a web browser rendering engine, which Google forked from WebKit in 2013. Maintaining their own codebase allowed Google to drop a large amount of code, and remove the need for several patches that were necessary to use WebKit for their own web browser Chrome. 141

c denotes the speed of light in equations. 7, 8

Chrome is Google's freeware web browser, which has an estimated worldwide market share of over 50 %. 1, 57, 71, 72, 74, 78, 80–82, 84, 87, 89, 139, 141

Cron is a time-based job scheduler software on various Unix-like operating systems to automate execution of programs, scripts, or commands at fixed times, dates, or intervals. The name derives from the Greek word chronos for time. 97

Cybercrime (or **computer crime**) is a collective noun for any crime that involves modern telecommunication networks to be committed. Examples include cracking, copyright infringement, child pornography, or drug and weapon sales. 1

$\boldsymbol{d_{\text{nodal}}}$ denotes the total nodal delay in equations. 6, 8, 15

$\boldsymbol{d_{\text{proc}}}$ denotes the nodal processing delay in equations. 6

$\boldsymbol{d_{\text{prop}}}$ denotes the propagation delay in equations. 7, 8

$\boldsymbol{d_{\text{queue}}}$ denotes the queue delay in equations. 6

$\boldsymbol{d_{\text{trans}}}$ denotes the transmission delay in equations. 7

$\boldsymbol{f}$ denotes a function in equations. 38

$\boldsymbol{f_{mac}}$ denotes a MAC function in cryptography and equations. 38

$\boldsymbol{f_{sym}}$ denotes a symmetric encryption function in cryptography and equations. 38

Firefox is Mozilla's free and open source web browser, which has an estimated worldwide market share of 12 % to 20 %. 1, 57, 71

Git is a distributed revision control and source code management system designed and developed by Linus Torvalds for the Linux kernel development in 2005. It is the most widely adopted version control system for software development. 72, 140

GitHub is a git repository hosting service, which offers unlimited repositories for open source code. The service was launched in April 2008, and has become the largest code host in the world with over 3.4 million users in 2014. 78, 80, 89, 103

HTTP cookie is sent from a server to the client in form of the `Set-Cookie` HTTP header. It can contain arbitrary data plus a few attributes, which are defined in RFC 6265. Clients decide if they accept cookies or not, with the latter generally making web browsing complicated. This is why most UAs have them activated by default. 14

Internet Explorer is Microsoft's famous web browser and the default for all their OSs. It had an estimated market shared of 95 % during 2002 and 2003, but will be discontinued with the release of Windows 10 in favor of a new and modern browser known as Microsoft Edge (codename "Project Spartan"). 48, 110

$\boldsymbol{\mathcal{K}}$ denotes a key space in cryptography and equations. 33, 40

$\boldsymbol{k}$ denotes a key in cryptography and equations. 33, 36–39

$\boldsymbol{k_{private}}$ denotes a private key in cryptography and equations. 38, 39

$\boldsymbol{k_{public}}$ denotes a public key in cryptography and equations. 38, 39

Kerberos is a distributed authentication service and network protocol for insecure networks and defined in RFC 4120. The protocol's current revision is at 5, the first public revision was 4. The system uses symmetric cryptography and requires a trusted third party. It found wide deployment through Microsoft's Active Directory directory service. 36, 43

$\boldsymbol{l}$ denotes length measurements in equations. 7

$\boldsymbol{n}$ denotes the refractive index in equations 7, 8

Nginx (pronounced "engine x") is an open source web, cache, and reverse proxy server for various Internet protocols as well as a load balancer. The main focus of the development is high concurrency, performance and low system resource usage. It utilizes an asynchronous event-driven mechanism to handle requests for highest throughput. 14, 54, 57, 59, 60, 71, 75, 76, 78, 95–102

OpenSSL is the de facto standard open source security library for SSL and TLS, and utilized by most web servers. It features basic general purpose cryptography functions for certificate and key management. Development started in 1995 under the name SSLeay and was renamed to OpenSSL in 1998. 57, 60, 76–78, 95, 96

PhantomJS is a headless WebKit based web browser with a scriptable JS API. Its main use cases are functional website tests, web page automation, screen capturing, and network monitoring via the HAR format. 64, 69

$\boldsymbol{r}$ denotes throughput in equations. The letter t is not used because it may be confused with *time*. 7, 9, 10

$\boldsymbol{s}$ denotes distance measurements in equations. The word derives from the Latin word *spatium* (space). The letter d is not used because it may be confused with the various *delay* variables. 8

Safari is Apple's default web browser on their OSs. It uses the open source rendering engine WebKit. 57, 141

Shell Script is a computer program designed to be run by the Unix shell command-line interpreter written in one of the various shell scripting languages. A portable shell script that is supposed to run on as many platforms as possible should generally be written against the default POSIX shell `/bin/sh`. 98, 99

$\boldsymbol{t}$ denotes time measurements in equations. 9, 17, 18, 25

$\boldsymbol{u}$ denotes user(s)/client(s) in equations. The letter n is not used because it may be confused with the *refractive index*. 10, 36

$\boldsymbol{v}$ denotes velocity (speed) measurements in equations. 8

Web Real-Time Communication is a W3C drafted assortment of standards, protocols, and JS APIs, together they enable browsers to provide peer-to-peer audio, video, and file sharing without plug-ins. 9, 113

WebKit is an open source web browser rendering engine used by Safari and earlier versions of Chrome. Google forked the WebCore component of WebKit in 2013 and announced that they will develop their own rendering engine in the future: Blink. 139, 141

WebSocket refers to the protocol defined in RFC 6455 and the W3C API. It provides full-duplex, message-oriented streaming of text and binary data between client and server. The only connection to HTTP is that the initial handshake is performed via an upgrade request. The standard also introduces two new URI schemes, namely `ws:` (unencrypted) and `wss:` (encrypted). 21

x denotes an arbitrary measurement in equations and the *cleartext* in cryptography. The letter *n* is not used because it may be confused with the *refractive index*. 9, 17, 25, 27, 33, 38, 142

X.509 is a standard from the ITU-T for the public-key and privilege management infrastructure, which describes the formats for certificates, revocation lists, and path validation algorithms. 43

y see *x* and denotes the *ciphertext* in cryptography. 25, 27, 33, 38

95

z see *x* (not used in cryptography). 95

Appendices

A Test System

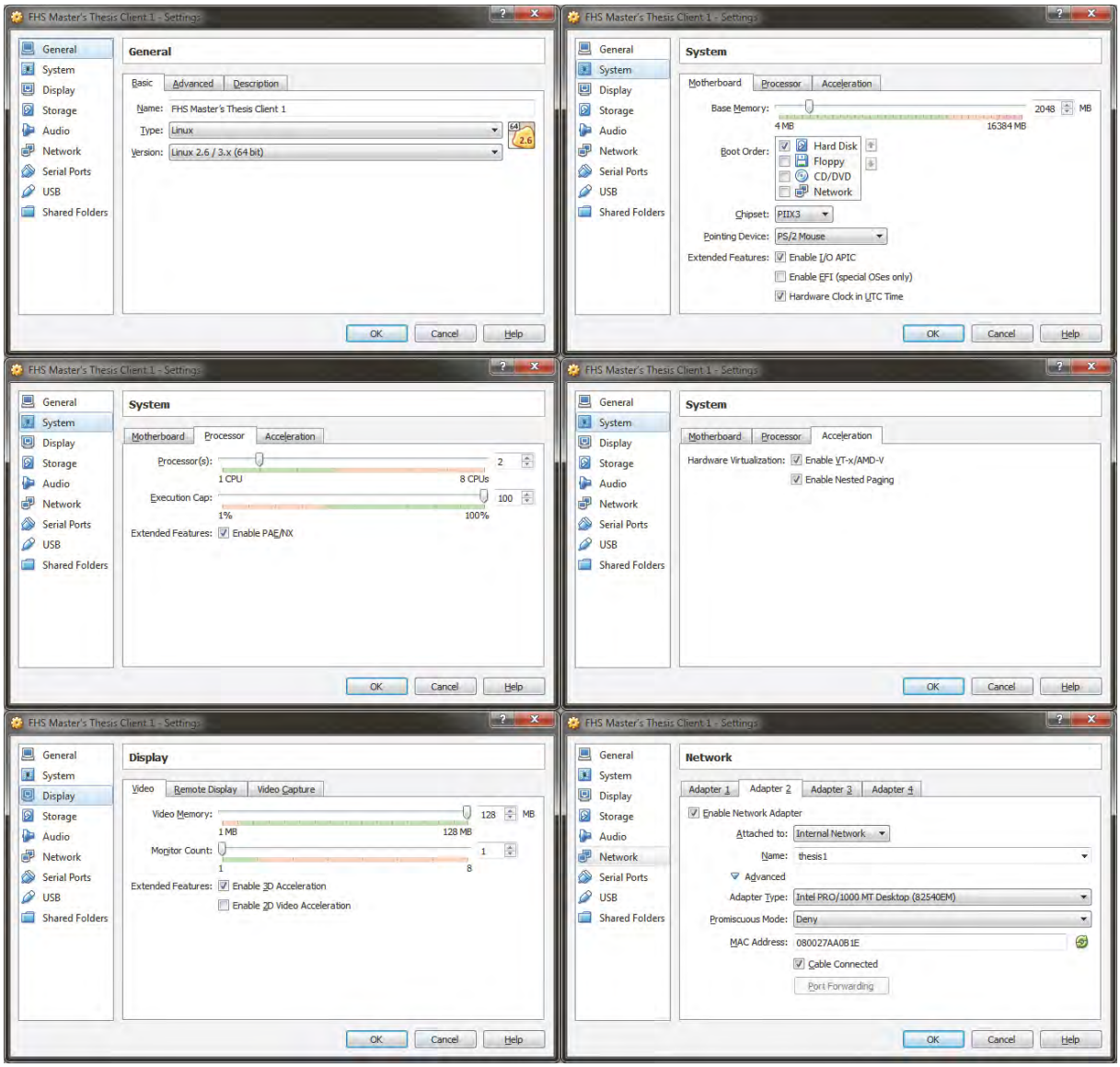


Figure A.1: VirtualBox Client Configuration Overview

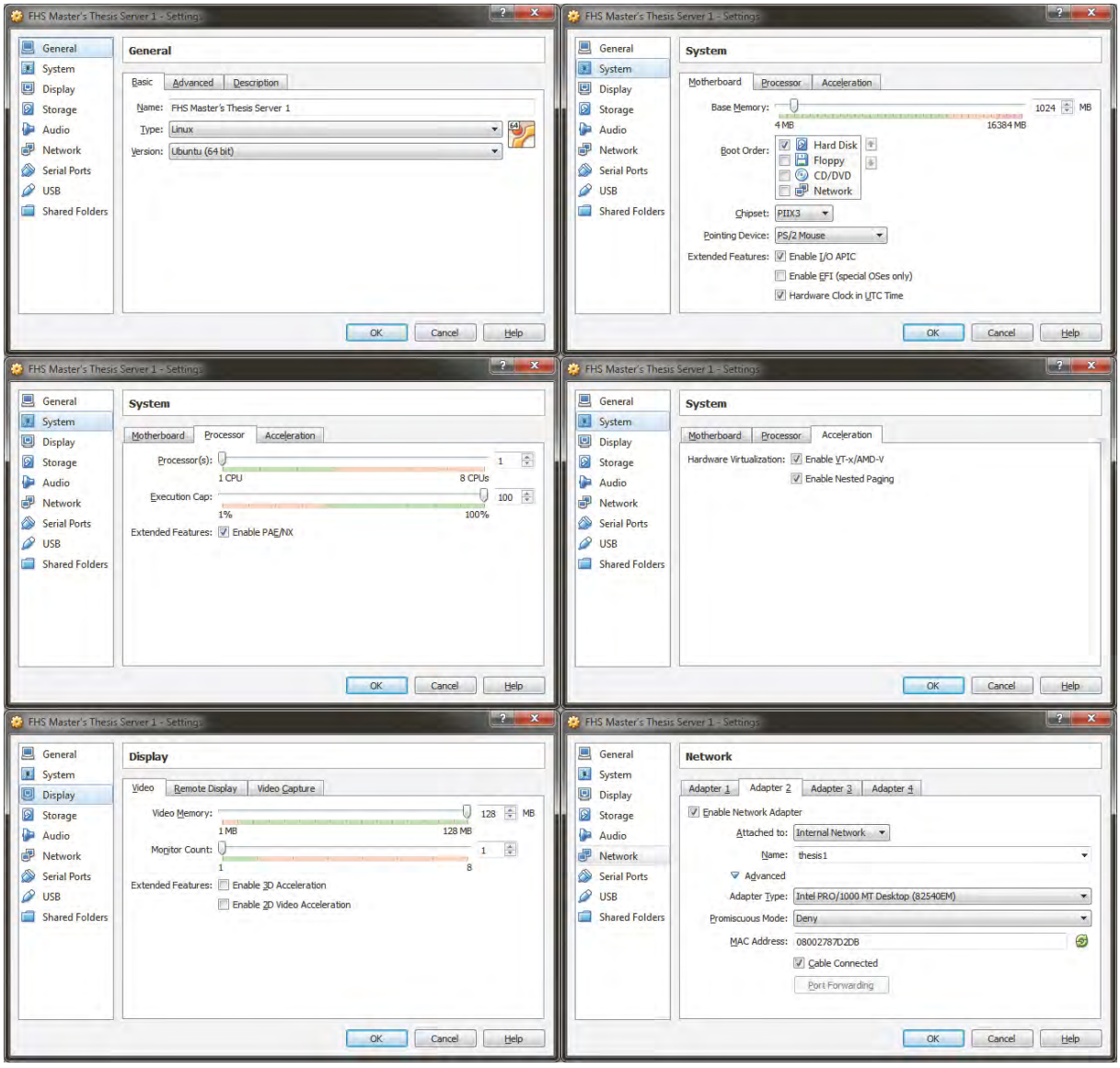


Figure A.2: VirtualBox Server Configuration Overview

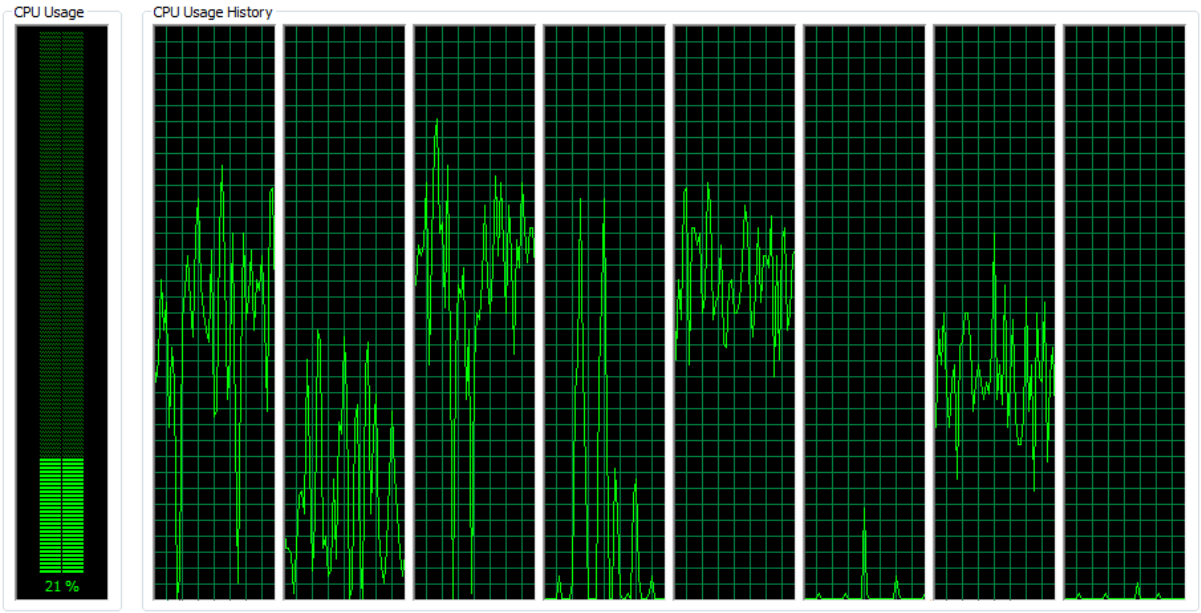


Figure A.3: Host System CPU Usage During Tests

usb3Monitor.exe *32	Fleshgri...	00	1,444 K	usbmonitor
VBoxSVC.exe	Fleshgri...	00	13,860 K	VirtualBox Interface
VirtualBox.exe	Fleshgri...	00	1,688 K	VirtualBox.exe
VirtualBox.exe	Fleshgri...	00	68,304 K	VirtualBox.exe
VirtualBox.exe	Fleshgri...	00	1,724 K	VirtualBox.exe
VirtualBox.exe	Fleshgri...	00	112,732 K	VirtualBox.exe
VirtualBox.exe	Fleshgri...	00	1,692 K	VirtualBox.exe
VirtualBox.exe	Fleshgri...	00	1,720 K	VirtualBox.exe
VirtualBox.exe	Fleshgri...	24	107,784 K	VirtualBox.exe
VirtualBox.exe	Fleshgri...	09	67,900 K	VirtualBox.exe
VirtualBox.exe	Fleshgri...	00	1,720 K	VirtualBox.exe
VirtualBox.exe	Fleshgri...	00	1,720 K	VirtualBox.exe
VirtualBox.exe	Fleshgri...	00	1,692 K	VirtualBox.exe
VirtualBox.exe	Fleshgri...	00	1,692 K	VirtualBox.exe
winlogon.exe		00	3,700 K	

Figure A.4: Host System Memory Usage During Tests

B Analysis Source Code

```
45 COMPUTE realTime=blocked + dns + connect + send + wait + receive.
46 EXECUTE.
47
48 DATASET DECLARE DataSetAvgTime.
49 AGGREGATE
50    /OUTFILE="DataSetAvgTime"
51    /BREAK=protocol algorithm sharding iteration
52    /avgRealTime "Average Time"=MEAN(realTime).
53 DATASET ACTIVATE DataSetAvgTime.
54
55 DEFINE !outputdatadirectory ()
56    !path+!folder+"convergence-plot"
57 !ENDDEFINE.
58
59 SPSSINC SPLIT DATASET SPLITVAR=protocol algorithm sharding
60    /OUTPUT DIRECTORY=!outputdatadirectory DELETECONTENTS=NO
61    /OPTIONS NAMES=VALUES.
62
63 SPSSINC PROCESS FILES INPUTDATA=!outputdatadirectory+"\*.sav"
64    SYNTAX=!path+"convergence-plot2csv.sps"
65    OUTPUTDATADIR=!outputdatadirectory
66    VIEWERDIR=!outputdatadirectory
67    CLOSEDATA=YES
68    LOGFILEMODE=OVERWRITE
69    /MACRODEFS ITEMS.
```

Listing A.1: SPSS Syntax to calculate real time and export groups into separate CSV files.

```
1 DEFINE !out ()
2    !quote(!concat(!unquote(!eval(!job_datadir)), "\",
     ↪  !unquote(!eval(!job_datafileroot)), ".csv"))
3 !ENDDEFINE.
4
5 GET FILE="JOB_INPUTFILE".
6 SAVE TRANSLATE OUTFILE=!out
7    /TYPE=CSV
8    /MAP
9    /REPLACE
10   /FIELDNAMES
11   /CELLS=VALUES
12   /DROP=protocol algorithm sharding.
```

Listing A.2: SPSS Syntax CSV conversion helper for listing A.1.

```php
<?php
// ...
    $sum = 0;
    $medians = [ ];
    while ($input->valid()) {
        $data = $input->fgetcsv();
        if ($data[0] !== null) {
            if ($input->key() === 0) {
                $data = [ "Iteration", "Time", "Average", "Median" ];
            } else {
                array_push(
                    $data,
                    $this->average($sum, $input->key(), $data[1]),
                    $this->median($medians, $input->key(), $data[1])
                );
            }
            $output->fputcsv($data);
        }
        $input->next();
    }
// ...
private function average(&$sum, $count, $sample) {
    $sum += $sample;
    return $sum / $count;
}
// ...
private function median(array &$medians, $count, $sample) {
    $medians[] = $sample;
    if ($count === 1) {
        return $sample;
    }
    sort($medians, SORT_NUMERIC);
    $middle = (int) floor($count / 2);
    $median = $medians[$middle];
    if ($count % 2 === 0) {
        $median = ($median + $medians[$middle - 1]) / 2;
    }
    return $median;
}
// ...
```

Listing A.3: PHP script to create data points for average and median convergence plot (reformatted and simplified for brevity).

```
45 COMPUTE realTime=blocked + dns + connect + send + wait + receive.
46 EXECUTE.
47
48 SORT CASES BY protocol algorithm sharding iteration.
49 SPLIT FILE LAYERED BY protocol algorithm sharding.
50
51 DATASET ACTIVATE DataSet.
52 DATASET DECLARE DataSetReused.
53 AGGREGATE
54    /OUTFILE="DataSetReused"
55    /PRESORTED
56    /BREAK=protocol algorithm sharding
57    /TlsReused=SUM(TlsReused)
58    /AvgTlsReused=MEAN(TlsReused)
59    /MedTlsReused=MEDIAN(TlsReused)
60    /ConnectionReused=SUM(connectionReused)
61    /AvgConnectionReused=MEAN(connectionReused)
62    /MedConnectionReused=MEDIAN(connectionReused)
63    /DnsReused=SUM(DnsReused)
64    /AvgDnsReused=MEAN(DnsReused)
65    /MedDnsReused=MEDIAN(DnsReused)
66    /Entries=N.
67 DATASET ACTIVATE DataSetReused.
```

Listing A.4: SPSS Syntax to determine the reuse of DNS and *connection* per protocol, used for table 10.

```
45 COMPUTE realTime=blocked + dns + connect + send + wait + receive.
46 EXECUTE.
47
48 SORT CASES BY protocol algorithm sharding iteration.
49
50 DATASET DECLARE DataSetEntries.
51 AGGREGATE
52    /OUTFILE="DataSetEntries"
53    /PRESORTED
54    /BREAK=protocol algorithm sharding iteration
55    /Time=MEAN(realTime)
56    /Receive=MEAN(receive)
57    /Wait=MEAN(wait)
58    /Send=MEAN(send)
59    /Connect=MEAN(connect)
60    /TLS=MEAN(tls)
61    /DNS=MEAN(dns)
62    /Blocked=MEAN(blocked).
63 DATASET ACTIVATE DataSetEntries.
64
65 COMPUTE Connect=Connect - TLS.
66 EXECUTE.
67
68 STRING dataSet (A64).
69 COMPUTE dataSet = CONCAT(RTRIM(CONCAT(protocol, "-", algorithm), "-"), "-", sharding).
70 EXECUTE.
71
72 DATASET DECLARE DataSetMedEntries.
73 AGGREGATE
74    /OUTFILE="DataSetMedEntries"
75    /BREAK=dataSet
76    /Time=MEDIAN(Time)
77    /Receive=MEDIAN(Receive)
78    /Wait=MEDIAN(Wait)
79    /Send=MEDIAN(Send)
80    /Connect=MEDIAN(Connect)
81    /TLS=MEDIAN(TLS)
82    /DNS=MEDIAN(DNS)
83    /Blocked=MEDIAN(Blocked).
84 DATASET ACTIVATE DataSetMedEntries.
85
86 SAVE TRANSLATE OUTFILE=!path+!folder+"-timing-breakdowns\"+!file+".csv"
87    /TYPE=CSV
88    /MAP
89    /REPLACE
90    /FIELDNAMES
91    /CELLS=VALUES.
```

Listing A.5: SPSS Syntax to aggregate all timings and create a stack chart as seen in figure 21.

```
45 COMPUTE realTime=blocked + dns + connect + send + wait + receive.
46 EXECUTE.
47
48 FREQUENCIES VARIABLES=realTime /FORMAT=NOTABLE /HISTOGRAM NORMAL /ORDER=ANALYSIS.
```

Listing A.6: SPSS Syntax for generation of normal distribution histograms.

C License

C.1 Using Creative Commons Public Licenses

Creative Commons public licenses provide a standard set of terms and conditions that creators and other rights holders may use to share original works of authorship and other material subject to copyright and certain other rights specified in the public license below. The following considerations are for informational purposes only, are not exhaustive, and do not form part of our licenses.

Considerations for licensors: Our public licenses are intended for use by those authorized to give the public permission to use material in ways otherwise restricted by copyright and certain other rights. Our licenses are irrevocable. Licensors should read and understand the terms and conditions of the license they choose before applying it. Licensors should also secure all rights necessary before applying our licenses so that the public can reuse the material as expected. Licensors should clearly mark any material not subject to the license. This includes other CC-licensed material, or material used under an exception or limitation to copyright. More considerations for licensors: https://wiki.creativecommons.org/Considerations_for_licensors_and_licensees#Considerations_for_licensors.

Considerations for the public: By using one of our public licenses, a licensor grants the public permission to use the licensed material under specified terms and conditions. If the licensor's permission is not necessary for any reason—for example, because of any applicable exception or limitation to copyright—then that use is not regulated by the license. Our licenses grant only permissions under copyright and certain other rights that a licensor has authority to grant. Use of the licensed material may still be restricted for other reasons, including because others have copyright or other rights in the material. A licensor may make special requests, such as asking that all changes be marked or described. Although not required by our licenses, you are encouraged to respect those requests where reasonable. More considerations for the public: https://wiki.creativecommons.org/Considerations_for_licensors_and_licensees#Considerations_for_licensees.

C.2 Legal Code

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

C.2.1 Definitions

1. *Adapted Material* means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synced in timed relation with a moving image.
2. *Adapter's License* means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.
3. *BY-NC-SA Compatible License* means a license listed at https://creativecommons.org/compatiblelicenses, approved by Creative Commons as essentially the equivalent of this Public License.
4. *Copyright and Similar Rights* means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section C.2.2 2a and item 2b are not Copyright and Similar Rights.
5. *Effective Technological Measures* means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.
6. *Exceptions and Limitations* means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.
7. *License Elements* means the license attributes listed in the name of a Creative Commons Public License. The License Elements of this Public License are Attribution, NonCommercial, and ShareAlike.
8. *Licensed Material* means the artistic or literary work, database, or other material to which the Licensor applied this Public License.
9. *Licensed Rights* means the rights granted to You subject to the terms and conditions

of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

10. *Licensor* means the individual(s) or entity(ies) granting rights under this Public License.
11. *NonCommercial* means not primarily intended for or directed towards commercial advantage or monetary compensation. For purposes of this Public License, the exchange of the Licensed Material for other material subject to Copyright and Similar Rights by digital file-sharing or similar means is NonCommercial provided there is no payment of monetary compensation in connection with the exchange.
12. *Share* means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.
13. *Sui Generis Database Rights* means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.
14. *You* means the individual or entity exercising the Licensed Rights under this Public License. Your has a corresponding meaning.

C.2.2 Scope

1. *License grant.*
   (a) Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:
      i. reproduce and Share the Licensed Material, in whole or in part, for NonCommercial purposes only; and
      ii. produce, reproduce, and Share Adapted Material for NonCommercial purposes only.

   (b) *Exceptions and Limitations.* For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.
   (c) *Term.* The term of this Public License is specified in Section C.2.6 item 1.
   (d) *Media and formats; technical modifications allowed.* The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply

making modifications authorized by this Section C.2.2 item 1d never produces Adapted Material.

(e) *Downstream recipients.*

i. *Offer from the Licensor –Licensed Material.* Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

ii. *Additional offer from the Licensor –Adapted Material.* Every recipient of Adapted Material from You automatically receives an offer from the Licensor to exercise the Licensed Rights in the Adapted Material under the conditions of the Adapter's License You apply.

iii. *No downstream restrictions.* You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

(f) *No endorsement.* Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section C.2.3 item 1(a)i.

2. *Other rights.*

(a) Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

(b) Patent and trademark rights are not licensed under this Public License.

(c) To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties, including when the Licensed Material is used other than for NonCommercial purposes.

C.2.3 License Conditions

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

1. *Attribution.*

(a) If You Share the Licensed Material (including in modified form), You must:

i. retain the following if it is supplied by the Licensor with the Licensed Material:

A. identification of the creator(s) of the Licensed Material and any others

designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);
B. a copyright notice;
C. a notice that refers to this Public License;
D. a notice that refers to the disclaimer of warranties;
E. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

ii. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and
iii. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

(b) You may satisfy the conditions in Section C.2.3 item 1a in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.
(c) If requested by the Licensor, You must remove any of the information required by Section C.2.3 item 1(a)i to the extent reasonably practicable.

2. *ShareAlike.*
In addition to the conditions in Section C.2.3 item 1, if You Share Adapted Material You produce, the following conditions also apply.

(a) The Adapter's License You apply must be a Creative Commons license with the same License Elements, this version or later, or a BY-NC-SA Compatible License.
(b) You must include the text of, or the URI or hyperlink to, the Adapter's License You apply. You may satisfy this condition in any reasonable manner based on the medium, means, and context in which You Share Adapted Material.
(c) You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, Adapted Material that restrict exercise of the rights granted under the Adapter's License You apply.

C.2.4 Sui Generis Database Rights

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

1. for the avoidance of doubt, Section C.2.2 item 1a grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database for NonCommercial purposes only;
2. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material, including for purposes of Section C.2.3 item 2; and

3. You must comply with the conditions in Section C.2.3 item 1 if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section C.2.4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

C.2.5 Disclaimer of Warranties and Limitation of Liability

1. *Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors, whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.*
2. *To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.*
3. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

C.2.6 Term and Termination

1. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.
2. Where Your right to use the Licensed Material has terminated under Section C.2.6 item 1, it reinstates:

   (a) automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or
   (b) upon express reinstatement by the Licensor.

   For the avoidance of doubt, this Section B.2.6 item 2 does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.
3. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

4. Sections C.2.1, C.2.5, C.2.6, C.2.7, and C.2.8 survive termination of this Public License.

C.2.7 Other Terms and Conditions

1. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.
2. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

C.2.8 Interpretation

1. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully be made without permission under this Public License.
2. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.
3. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.
4. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

C.3 Creative Commons Notice

Creative Commons is not a party to its public licenses. Notwithstanding, Creative Commons may elect to apply one of its public licenses to material it publishes and in those instances will be considered the “Licensor”. Except for the limited purpose of indicating that material is shared under a Creative Commons public license or as otherwise permitted by the Creative Commons policies published at https://creativecommons.org/policies, Creative Commons does not authorize the use of the trademark “Creative Commons” or any other trademark or logo of Creative Commons without its prior written consent including, without limitation, in connection with any unauthorized modifications to any of its public licenses or any other arrangements, understandings, or agreements concerning use of licensed material. For the avoidance of doubt, this paragraph does not form part of the public licenses.

Creative Commons may be contacted at https://creativecommons.org/.