TUM School of Computation, Information and Technology
Technische Universität München

Improving the Security, Performance, and Availability of TLS-based Systems on the Internet

Markus Andreas Sosnowski

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz: Prof. Dr.-Ing. Jörg Ott

Prüfende der Dissertation:

1. Prof. Dr.-Ing. Georg Carle
2. Prof. Dr. Matthias Wählisch

Die Dissertation wurde am 19.02.2025 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 10.02.2026 angenommen.

Abstract

The Internet is a complex system that has become essential to modern society. Improving the challenges of security, performance, and availability is critical for achieving a future-proof digital infrastructure. Given that most Internet communication relies on Transport Layer Security (TLS), systems based on TLS are of special interest in addressing these challenges. This thesis investigates contributions to these challenges in three main areas:

First, we focus on security and explore methods to actively derive innovative Cyber Threat Intelligence (CTI) from TLS-based systems. We examine the potential and limitations of Internet-wide measurements in collecting and modeling TLS characteristics to serve as CTI. We investigate Active TLS Fingerprinting and propose two methods: one utilizing a fixed set of scanning probes created with a heuristic entropy optimization and another using exhaustive scanning through a model-driven design of scanning probes. Both methods aim to maximize the coverage of collectible information. Additionally, we introduce an Internet-wide graph model for the propagation of threat information. We demonstrate the effectiveness of these methods by classifying potentially malicious Command and Control servers.

Second, we focus on security and performance by analyzing how negotiated cryptographic algorithms impact the latency of TLS-secured communication. We evaluate the effects of new Post-Quantum Cryptography (PQC) developments on TLS-based systems through experiments in a controlled testbed environment under simulated network conditions. While most PQC algorithms were fast, we identified the large key sizes associated with these algorithms as potential bottlenecks, particularly in low bandwidth or high packet loss environments. Moreover, our findings suggest that adjusting the mapping between TLS messages and Transmission Control Protocol (TCP) segments and tuning the initial TCP Congestion Window can enhance performance.

Third, we focus on performance and availability by investigating a novel web caching technique that uses Conflict-free Replicated Data Types (CRDTs) to expand the applicability of web caching to a broader range of TLS-based systems. This CRDT Web Caching (CWC) method allows web proxies to immediately respond to read and write requests while synchronizing any data changes with the origin server at a later time. We compare CWC with existing web caching techniques in a simulated Content Delivery Network. The results indicate that CWC matches conventional caching performance based on a Time to Live and achieves an eventual consistency similar to invalidation-based caching while uniquely accelerating state-changing client requests.

This thesis offers valuable insights into measuring, modeling, and analyzing TLS-based systems to contribute towards a more secure, performant, and available Internet.

Acknowledgements

My academic journey resulting in this thesis would not have been possible without the support of many individuals. I want to use this opportunity to thank you all. Your support has been invaluable, and I wish to highlight some key roles in this endeavor.

First and foremost, I would like to thank my supervisor, Prof. Dr.-Ing. Georg Carle, for allowing me to pursue my doctoral degree under your guidance. I learned a lot during this time, and you sparked many ideas by always having a grander vision in mind. I am grateful for your mentorship and the insightful discussions we have shared.

I would also like to thank my second examiner, Prof. Dr. Matthias Wählisch, for the invitation to Dresden, the insights you shared, and for your role in evaluating this thesis. My appreciation also goes to Prof. Dr.-Ing. Jörg Ott for his role as chair of the examination committee and for his contributions to this process.

Moreover, I would like to thank my colleagues who accompanied and supported me during the last six years and made my experience at the chair not only productive but also enjoyable. I would like to especially thank the colleagues who also worked with me as co-authors: Eric, Florian, Johannes, Juliane, Lion, Patrick, Richard, Sebastian, and Tim.

Lastly, I owe a great deal of thanks to my family and friends. Your support and unwavering belief in me provided me the strength to continue. This goes especially towards my parents, Alexandra and Christian. Without your care and support, I could never have followed this path in life. Thank you, Kira, for always being thoughtful, loving, and supportive in my pursuits.

Thank you all once again. With your help, I find myself at the conclusion of an important chapter in my life and eagerly look forward to the adventures that lie ahead.

Contents

Chapter 1

Introduction

The Internet is a globally distributed network of interconnected systems and has become essential to modern society. It enables global communication, information sharing, and commerce, supporting social interactions, businesses, and critical infrastructure. Transport Layer Security (TLS) constitutes the majority of Internet traffic [94] and has become the standard for secure communication. It provides encryption, authentication, and integrity to communication parties, vital for protecting sensitive information from unauthorized access or tampering.

However, the growing complexity and scale of the Internet pose significant challenges to its security, performance, and availability. Improving these aspects is crucial to ensuring that today's digital infrastructure can meet the increasing demands of our society. Given the significant role that TLS-based systems play on the Internet, they are of particular research interest for achieving these improvements.

Moreover, a more secure, performant, and available Internet provides a stable foundation for developing and deploying new technologies and services, fostering innovation and new use cases in the future.

1.1 Challenges

This thesis focuses on TLS-based systems and the following three key challenges: security, performance, and availability.

Security: A more secure Internet can better respond to and defend against cyber threats, protect users' sensitive data, and prevent attackers from accessing critical infra-

structure. Key aspects of this security include closing potential attack vectors, adapting to new technological developments, and researching and developing techniques to detect malicious activities, ideally before an attack occurs.

Performance: A high-performant Internet is vital for meeting the needs of current and future users and digital services. Important aspects of the Internet's performance include communication speed, resource costs, and the scalability of services. Improving the performance can result in faster loading times and an enhanced user experience, which often involves identifying and overcoming bottlenecks and selecting the right technology for a concrete use case. Additionally, performance is interconnected with security, as users are more likely to adopt secure technologies if those technologies do not negatively impact performance.

Availability: An available Internet guarantees that digital communication remains accessible and that digital services and resources can be reached anytime and anywhere. One of the challenges in maintaining digital service availability is ensuring accessibility even during high traffic, technical failures, or cyberattacks. Enhancing availability can be achieved by improving the fault tolerance of the communication and the accessed services, minimizing single points of failure, developing strategies to overcome service downtimes, improving the distributed architecture of the Internet, and promoting the development of distributed services.

1.2 Research Objectives

This thesis is structured around three Research Objectives (ROs), each addressed in a separate chapter and devided into Research Questions (RQs). The following paragraphs introduce each objective, the applied methodologies, key findings, and how the RQs contribute toward the investigated key challenges. The objectives, research questions, and key findings are summarized in Table 1.1.

RO1: Deriving Cyber Threat Intelligence by modeling and measuring TLS server characteristics. Our first research objective focuses on enhancing the security of the Internet by developing innovative methods to derive Cyber Threat Intelligence (CTI) from TLS-based systems through active measurements. Our primary approach involves collecting TLS characteristics from servers, modeling the data to identify security-relevant aspects, and classifying servers based on these characteristics. This methodology enabled us to introduce an Internet-wide context to threat detection approaches. The hope behind this is that by broadening the limited perspective of threat detection approaches based

on monitoring local networks to a more global scale, we can improve their effectiveness in identifying malicious activity. Our methods can benefit Intrusion Detection System (IDS), network operators, and security researchers in searching and identifying malicious activity. The chapter explores four main topics, as outlined below.

First, we investigate efficient methods for obtaining representations of the TLS stack on servers through Active TLS Fingerprinting. By differentiating between fixed-probe and exhaustive fingerprinting, we propose two methodologies for each: EFACTLS and DissecTLS. Our large-scale studies demonstrated that these methods can identify potentially malicious Command and Control (C2) servers using a threshold-based classification.

Next, we explore using multiple scanning probes to enhance information gathering during active TLS measurements. Because a single handshake provides only limited information, multiple handshakes can be used to acquire a more comprehensive view. The challenge is determining the best combination of probes to yield enough information to be useful while enabling scalable Internet-wide measurements. We propose a heuristic Entropy optimization strategy for fixed-probe scanning and a model-driven design for exhaustive scanners to improve the amount of information collected.

The third main topic examines modeling data from the TLS Ecosystem as an Internet-wide Property Graph to leverage the relationships among domains, Internet Protocol (IP) addresses, and X.509 certificates. This graph model facilitates the propagation of threat information, enabling the identification of new C2 servers before they are added to blocklists. This methodology provides an effective framework for analyzing the extensive data forming from the TLS Ecosystem and uncovering previously unknown threats.

Finally, we explor the creation of a dataset for global certificate revocations, which can serve as a valuable source of CTI or provide a foundation for deriving new CTI. Our study evaluates the Online Certificate Status Protocol (OCSP) and Certificate Revocation List (CRL) embedded in certificates and published in the Common CA Database (CCADB) to establish an Internet-wide scanning and revocation collection pipeline. We found that CCADB CRLs provided the most extensive view of global certificate revocations.

RO2: Modeling TLS protocol characteristics to identify performance implications of negotiated algorithms. Our second research objective focuses on the performance of TLS-based systems in the context of new cryptographic developments. Adopting new cryptographic algorithms on the Internet may be necessary to maintain secure com-

munication; however, this can impact performance. Using passive measurements in a controlled testbed environment, we model TLS handshake characteristics to measure and analyze the performance implications of the negotiated algorithms. Given the complexity of modern networks, even small changes to the protocol can lead to unforeseen side effects, making it crucial to assess these performance implications before deploying the changes to a broader audience. The primary focus of this chapter is on recent developments in Post-Quantum Cryptography (PQC), covering the following three main topics.

First, we explore a methodology to model and measure TLS 1.3 handshakes for performance analyses in a local testbed designed to minimize measurement bias and emulate different network conditions. We analyze the handshake in two separate phases to better understand the implications of various negotiated algorithms.

Next, we examine the performance implications of PQC in TLS, comparing it with established algorithms such as Elliptic Curve DH (ECDH) and Rivest–Shamir–Adleman (RSA). Our findings indicate that while PQC is generally efficient, the larger key sizes can hinder performance, and choosing the most suitable post-quantum algorithm sometimes requires balancing computational and bandwidth costs.

Finally, we investigate new optimization opportunities for TLS-based systems when using post-quantum algorithms, primarily due to the larger key sizes involved in the handshake process. Adjustments to how TLS messages are mapped to Transmission Control Protocol (TCP) segments and increasing the initial Congestion Window (CWND) can enhance performance.

RO3: Extending the applicability of web caching to a broader range of TLS-based systems. Our third research objective focuses on enhancing the availability and performance of TLS-based systems on the Internet through an improved web caching technique. With the increasing adoption of TLS, the widespread adoption of Content Delivery Networks (CDNs), and the separation of static and dynamic content in web applications, new opportunities emerge for optimizing web application performance and availability. Current web proxies can efficiently cache read requests; however, to achieve a fully responsive application from the user's perspective, even during outages of the central application server, web proxies have to be capable to deal with request that change application state. We propose the concept of a more data-ware web proxy that operates more as an information broker than a traditional Hypertext Transfer Protocol (HTTP) proxy. As a result, such a proxy can go beyond the current state-of-the-art and; e. g., independently process state-changing client requests and automatically ensure cached content ist up-to-date. The chapter explores three main topics:

The first topic discusses a concrete method for caching mutating requests on web proxies based on Conflict-free Replicated Data Types (CRDTs), which we call CRDT Web Caching (CWC). This method enables web proxies to handle these requests independently while synchronizing changes with the origin server at a later time.

Next, we investigate the integration of CWC into existing applications. We propose a protocol upgrade for Representational State Transfer (REST) Application Programming Interfaces (APIs) that can be initiated on demand, allowing for seamless incorporation into existing systems. This upgrade ensures that application developers retain control over the applied logic.

Lastly, we analyze which types of API communication benefit most from the CWC approach and compare it to state-of-the-art caching strategies. We demonstrate that the most significant improvements can be achieved when API communication is expressable in simple Create Read Update Delete (CRUD) operations, as this allows for the caching of mutating requests and enhances application availability. In other scenarios, CWC still offers advantages in terms of performance and consistency due to enabling push-based cache updates. We experimentally evaluate different caching techniques using a simulated CDN and show that CWC matches the performance of caching based on a Time to Live (TTL) and the consistency of invalidation-based caching while uniquely capable of caching write operations.

Table 1.1: Overview of the main research objectives, research questions, and key findings of this thesis.

Focus	Research Question (RQ)	Key Findings
security	**Chapter 2: Deriving Cyber Threat Intelligence by modeling and measuring TLS server characteristics**	
security	**RQ1.1**: How can we efficiently obtain a representation of the TLS stack on a server to uncover undisclosed relationships and commonalities?	Active TLS Fingerprinting enables the detection of potentially malicious C2 servers. We propose and evaluate both a fixed-probe (EFACTLS) and exhaustive (DissecTLS) approach.
security	**RQ1.2**: How can we increase the information gathered through active TLS measurements by combining multiple scanning probes?	Heuristic Entropy optimization is a black-box approach to increase the information of a fixed-probe and a model-driven design of requests a white-box approach for exhaustive scanners.
security	**RQ1.3**: How can we leverage Internet-wide relations within the TLS Ecosystem to propagate threat information?	An Internet-wide Property Graph servers as basis for a threat propagation algorithm, enabling us to use existing blocklists and detect new C2 servers on average 3 months before they were added to the lists.
security	**RQ1.4**: How can we effectively collect and analyze revoked trust on a global scale?	Compared to OCSP and CRLs from certificates, the CCADB CRLs provided the most extensive view of global certificate revocations.
performance	**Chapter 3: Modeling TLS protocol characteristics to identify performance implications of negotiated algorithms**	
performance	**RQ2.1**: How can we model and measure TLS 1.3 handshakes for performance analyses in a local testbed?	Abstracted in two phases, they can be passively analyzed without decryption. Emulations provide insights into a wide range of network conditions.
performance	**RQ2.2**: What are the performance implications of post-quantum-safe TLS?	While the algorithms are generally fast, the large key sizes are a bottleneck. Some algorithms are a trade–off between compuational and bandwith costs.
performance	**RQ2.3**: Do new optimization opportunities emerge when post-quantum algorithms are deployed in a TLS network stack?	Due to the large key sizes, the mapping of TLS messages to TCP segments and the initial TCP CWND can be tuned for increased performance.
availability	**Chapter 4: Extending the applicability of web caching to a broader range of TLS-based systems**	
availability	**RQ3.1**: How can we process requests that change application state on Web proxies?	CRDTs allow web proxies to independently process mutating requests and synchronize the received changes with the origin later in time.
availability	**RQ3.2**: How can we integrate a novel web proxy technology into existing applications?	A proposed on-demand protocol upgrade between proxy and origin allows seamless integrate of the caching technology into existing applications for specific API endpoints.
availability	**RQ3.3**: What types of API communication benefit from the proposed web caching approach?	Availablity and performance improvements for communication that can be expressed in simple CRUD operations. In other cases, performance benefits due to the push-based caching semantic.

1.3 Outline

This thesis is structured around the three ROs introduced in Section 1.2. Chapter 2 focuses on RO1 and investigates methodologies for actively measuring, modeling, and analysing TLS-based systems to acquire Internet-wide CTI. Chapter 3 focuses on RO2 and examines measuring and modeling the TLS protocol to identify performance implications of the negotiated cryptographic algorithms for TLS-based systems. Chapter 4 focuses on RO3 and explores a methodology for extending the applicability of web caching to a broader range of TLS-based systems. Finaly, Chapter 5 concludes this thesis with a summary of the key contributions and a discussion of open research questions worth investigating.

Chapter 2

Deriving Cyber Threat Intelligence from TLS Servers

This chapter investigates methods for deriving CTI from TLS-based systems connected to the Internet. The contents of this chapter are based on four conference publications [1, 4, 5, 6] and a published journal article [8]. Changes and new contributions compared to the original publications are described in Section 2.12.

2.1 Introduction

Expanding the often limited perspective of actors searching for threats on the Internet to a more global scale can enhance their effectiveness in identifying these threats, thereby allowing them to better secure the Internet. For instance, an IDS vendor typically has a limited view of only the specific sites where the IDS is deployed. Such an IDS could significantly improve its effectiveness by incorporating global threat data into its detection processes. Similarly, such global threat data can be valuable for network operators monitoring their servers and security researchers seeking to identify unknown threats or malicious actors across the Internet.

This chapter explores methods for actively measuring, modeling, optimizing, and analyzing data from the TLS Ecosystem to provide security-relevant information. We understand the TLS Ecosystem as a combination of the TLS protocol, the Domain Name System (DNS), Public Key Infrastructures (PKIs), and the applications using TLS. A more detailed explanation is given in Section 2.7.1. We focus on the TLS Ecosystem for two main reasons:

i) Whenever TLS is employed, metadata (such as HTTP headers) and the concrete message payload used for detection become inaccessible to passive observers. In these cases, alternative information channels are necessary.

i) A common data basis for modeling is important to effectively compare and analyze different systems. TLS and DNS is extensively used across various application layer communications, providing a common basis while still being complex protocols and concepts that yield sufficient metadata for large-scale analyses.

While the studies in this chapter primarily focuses on C2 detection, the proposed methodologies are versatile enough to detect other malicious TLS-based systems. Furthermore, they can be used to monitor benign servers, providing insights into their deployments, and point out anomalies that should be investigated.

In the rest of this chapter, we will explore the following main research questions:

RQ1.1: How can we efficiently obtain a representation of the TLS stack on a server to uncover undisclosed relationships and commonalities? Active TLS Fingerprinting can reveal undisclosed information from TLS servers. We distinguish between fixed-probe and exhaustive fingerprinting and propose two methodologies for each category: EFACTLS and DissecTLS. Through large-scale active DNS and TLS measurement studies, we demonstrate that both methodologies can detect potentially malicious C2 servers using a threshold-based classification. We outline effective Active TLS Fingerprinting requirements and empirically compare our methods with related work.

RQ1.2: How can we increase the information gathered through active TLS measurements by combining multiple scanning probes? In a single TLS handshake, a server reveals only a subset of the information available in response to a request from a client. Conducting multiple handshakes can increase the amount of acquirable knowledge. However, finding the most effective combination of scanning probes is challenging, especially when aiming for a scalable solution for Internet-wide measurements. We propose using heuristic Entropy optimization as a black-box approach to maximize the information obtained with a fixed-probe scanner and a model-driven design that collects the maximum information based on the model as a white-box approach for exhaustive scanners.

RQ1.3: How can we leverage Internet-wide relations within the TLS Ecosystem to propagate threat information? We propose modeling data from the TLS Ecosystem as an Internet-wide Property Graph. This model captures relationships between domains, IP

addresses, and X.509 certificates based on active Internet-wide DNS and TLS measurements, allowing the propagation of threat scores across entities. Our approach enabled us to utilize existing blocklists as input and identify new C2 servers, on average, three months before they were added to the respective blocklist. Together with statistical and manual analyses, we illustrate that our methodology is an effective abstraction of the vast data collectible from the TLS Ecosystem that can be used for subsequent applications, such as identifying previously unknown threats on the Internet.

RQ1.4: How can we effectively collect and analyze revoked trust on a global scale? A global perspective on revoked trust can serve as a valuable source of CTI. However, acquiring a comprehensive dataset can be challenging. We evaluate an Internet-wide scanning and revocation collection pipeline based on the OCSP, CRLs from certificates, and CRLs from the CRLs of the CCADB. Our analyses indicate that the CRLs obtained from the CCADB offered the most extensive view of global certificate revocations.

2.2 Background

Research [46, 94] has shown that the TLS protocol family forms a significant part of web traffic and has become the standard for encrypted communication over the Internet. Moreover, cybercriminals are increasingly using TLS to hide their communication. Due to continuous development and the need for backward compatibility, TLS has become a complex system that provides various meta information related to client and server capabilities during the initial TLS handshake. In addition, TLS, in conjunction with the DNS and the Web PKI, forms an interconnected ecosystem that underpins most of the current Internet. The widespread adoption and inherent complexity, resulting in a large amount of collectible metadata, makes TLS an ideal basis for large-scale analyses, with the potential to provide valuable CTI.

One example of such CTI is an indicator of whether the server is a botnet C2 server. Ideally, a C2 is preventively detected before it has caused any harm and is commonly known to be malicious. Such detection can be interpreted as a classification problem predicting a threat score, with a higher score indicating a higher likelihood of a server being considered a C2 server. This metric can be used by an IDS as one of many indicators used to classify observed network flows or by security researchers searching for unknown threats on the Internet. Previous research [132] has shown that active measurements of TLS allow gathering interesting metadata about servers, their configuration, or the actor behind a deployment. Moreover, blocklists like the abuse.ch SSLBL [15], which list certificates used by C2 servers, indicate that cybercriminals re-

peatedly use the same TLS configuration on their servers. Leveraging such relations by collecting and modeling TLS metadata from C2 servers promises a valuable basis for developing novel detection approaches.

2.3 Entropy as Information Metric for Active Measurements

In 1948, C. Shannon [137] defined a theoretical approach for measuring the amount of "information" contained in arbitrary communication. We propose to apply his definition to model the amount of information collected via active measurements.

On an abstract layer, active measurements are a method where devices are scanned and the observations are recorded. If we have only a single observation, we cannot apply Entropy yet. However, if we have collected a lot of observations we can see patterns, relations, and commonalities in the data. In research, we are usually interested in such findings an attribute them a higher value if we learn something new. For example, if we are interested in the deployment of TLS version in the Internet, we actively scan servers with a TLS handshake, record the negotiated versions, and group the servers depending on a common version.

Entropy is a metric that assigns an observation a low value if it is likely to be observed; on the other hand, an observation that rarely occurs has a high value. Intuitively, this also reflects our understanding of active measurements: scanning for a parameter that is the same on every server has no value. Even worse, this causes unnecessary scanning costs and should be avoided due to ethical reasons. However, a parameter observed only from a few servers is very valuable because it indicates a rare commonality. A downside of Entropy is that it can only be applied to parameters that group servers based on useful commonalities. For example, recording the current timestamp as scan output would result in a unique observation for each server and the highest possible Entropy. However, grouping servers based on such a timestamp would have no value because it does not reveal a useful commonality.

Nevertheless, Entropy can be an effective metric to evaluate actively collected data if the usefulness of a grouping based on the collected data is shown through other means; e. g., a measurement study like the one we perform in Section 2.6.8.

C. Shannon [137] abstracted a data source as a Random Variable $\mathcal{Q}$ that emits symbols $x \in \mathcal{X}$ with a certain probability $p : \mathcal{X} \to [0, 1]$. He interpreted the "uncertainty" of

each symbol as its information

$$I(x) = -log_2\, p(x). \tag{2.1}$$

The overall information, defined as Entropy, of a data source $\mathcal{Q}$ is the average across all possible symbols:

$$H(\mathcal{Q}) = \sum_{x \in \mathcal{X}} p(x) I(x). \tag{2.2}$$

The Entropy definition can be applied to active measurements by modeling the set of scanned servers as a single data source and Random Variable $\mathcal{Y}$. Every collected datum $d \in D$ is abstracted as a symbol that $\mathcal{Y}$ emits. In our case, a datum is the set of investigated parameters collected from a single server. The probability $p(d) \in [0, 1]$ that a specific datum occurs is unknown. However, we can measure the rate at which it occurred in a single measurement by dividing the number of observations from a specific datum $o(d) \in \mathbb{N}$ by the total number of scanned servers $t \in \mathbb{N}$. According to the law of large numbers [65], the occurrence rate approximates the actual probability, given that the set of scanned servers is large enough. Hence, we can measure the collected information from a single active measurement as

$$H'(\mathcal{Y}) = -\sum_{d \in \mathcal{D}} p'(d) \cdot log_2\, p'(d) \quad \text{, with } p'(d) = \frac{o(d)}{t}. \tag{2.3}$$

To conclude, the Shannon Entropy can be used to measure the collected information of an active measurement. Moreover, Entropy is a metric that allows to compare the effectiveness of different scanning approaches. We will discuss limitations we have observed when applying the Entropy metric to active measurements in Section 2.6.9.

2.4 Threshold-based Classification

This section presents the threshold-based classifier we use to implement different classification use cases based on active measurement data. Well-known classification metrics allow the evaluation of this classifier, particularly precision and recall.

The classifier works as follows. First, all servers are labeled according to the use case. The labels can be unspecific (e. g., whether a server is considered a C2 server or not) or more precise (e. g., a TrickBot C2 server). The former is considered a binary and the latter a multi-label classification. Then, we calculate a prediction for each server depending on the actively collected data and used approach. The prediction is a score in $[0, 1]$ that indicates a concrete label. A server can have multiple predictions for

different labels. However, the threshold-based classification only considers predictions above a certain threshold. For example, an approach predicts the two labels TrickBot and QakBot with a score of 12% and 30%, respectively. For a threshold of 20%, the classifier would predict the QakBot class; for a threshold of 50%, the classifier would produce no prediction.

This threshold-based classification enables us to transform continuous prediction scores into discrete classes. Having a discrete class both as label and classification result we can check if they match and calculate the well-known metrics precision and recall. Both metrics are defined only in the context of a specific label. Whenever the classification result matches the label, it is counted as true positive (TP); if they do not match, as false positive (FP). The true observationss (PP) are the total number of times a label occurred. The precision and recall is defined as

$$Precision = \frac{TP}{TP + FP} \quad \text{and} \quad Recall = \frac{TP}{PP}. \tag{2.4}$$

Intuitively, precision is the rate of correct classifications of a label and recall gives the rate of correctly identified servers compared to the total amount of servers that could have been identified. Due to the threshold, only some servers are necessarily assigned a class, leading to a lower recall. The threshold serves as a tuning parameter that can increase the precision at the cost of a sometimes lower recall.

In summary, the threshold-based classifier is a simple but effective approach to implement and evaluate the classification use cases described in Section 2.2.

2.4.1 Threshold-based Classification using Active Fingerprinting

Active fingerprinting "is the process of actively interacting with the target entity" [162] to reveal undisclosed information, like the type and version of software running on a device and it can be used for classification purposes. In this thesis we use active fingerprinting to generate a discrete fingerprint per server. Each server is assigned a single fingerprint; however, multiple servers can have the same fingerprint. The actual fingerprint depends on the scanning approach.

We used such fingerprints for a threshold-based classification by splitting the servers S into training and evaluation sets T and E, respectively. All servers are assigned a set and the two sets are distinct: $S = E \cup T$ and $E \cap T = \emptyset$. Each server has a fingerprint $s_f : S \to F$ and label $s_l : S \to L$. For each fingerprint and label, we calculate a score $s : F \times L \to [0, 1]$ by dividing the number of times a fingerprint was seen with each label by the total number of occurrences of the respective fingerprint using the servers from

the training set:

$$s(f,l) = \frac{\sum_{s \in T} o_f(f,s) o_l(s,l)}{\sum_{s \in T} o_f(f,s)} \quad \text{, with} \tag{2.5}$$

$$o_f(f,s) = \begin{cases} 1 & s_f(s) = f \\ 0 & else \end{cases} \quad \text{and} \quad o_l(f,l) = \begin{cases} 1 & s_l(s) = f \\ 0 & else \end{cases} . \tag{2.6}$$

The scores reflects the probability that a particular fingerprint predicts a label. Due to the calculation, the scores for each predicted label for a single fingerprint will add up to one. This means when applying the threshold-based classification with a threshold above 50%, only a single label will be predicted. Afterward, we apply the predictions to the servers in the evaluation set to classify each server with scores above the configurable threshold. For example, a particular fingerprint appeared 100 times in the training set: 12 times from servers labeled TrickBot and 30 times QakBot. Observing the said fingerprint in the evaluation set would result in the scores of 12% TrickBot and 30% QakBot.

2.4.2 Continuous Classification over Time

In this thesis we have several classification use cases were we use measurement data for both training and evaluation. It is a common practice to use distinct sets for training and evaluation to show the generalizability of the approach. For example, defining the combination of IP address and domain name as fingerprint, an approach can trivially achieve a high precision on the training data but the fingerprints would not work on any other data set containing different servers. Training and evaluation sets are often split randomly. However, we want to show the applicability of Active TLS Fingerprinting for real-world use cases. Hence, we propose a continuous classification over time when dealing with active measurement data. Such a classification can be applied if a constantly changing input set of server (e. g., a block or top list) is repeatedly scanned over time. In such a setting, we can use the data collected from the past as training data for classifying the current data set. This idea is modeled after an IDS that can use past observations for life traffic classification. However, as the same servers tend to appear in multiple consecutive data sets we propose to only classify the new additions. In other words, if we applied a continuous classification over time, we evaluated the classifier on every new target added to the data set during week $n + 1$ based on the training data from weeks $[1..n]$.

2.5 Active DNS and TLS Measurements under Ethical Considerations

To collect metadata from the TLS Ecosystem we used an active measurement pipeline based on established tools and by following basic ethical principles.

The pipeline takes a list of IP addresses and domain names as input. MassDNS [36] and a local Unbound [120] server resolve the domains to their IPv4 and IPv6 addresses, resulting in a set of (IP address, domain) pairs we call targets. In some of our measurements we scanned servers with multiple TLS Client Hellos (CHs), in these cases the input targets were appended with an additional CH argument; thus, resulting the input targets to be (IP address, domain, CH) triples. Scanning with a set of CHs resulted the final scan input to be a randomly ordered cross-product of the target list and the CHs. IP addresses can be augmented with a TCP port that should be used instead of the default 443 port. We used the TUM goscanner [73] to perform a TLS handshake for each input target and collect the TLS metadata. The TUM goscanner is a TLS scanner designed for Internet-wide usage and initially implemented by Amann et al. [20]. If a domain name was available for an IP address, we used it as the Server Name Indication (SNI). We designed a custom TLS library based on the Golang standard library that allows the definition of arbitrary CHs as input for each TLS connection and extracts detailed TLS handshake metadata. Both the scanner and library are open-sourced [73]. We extended the TUM goscanner with the additional EFACTLS [8], DissecTLS [5] and JARM [19] functionality. In the latter two cases the scanner performed multiple consecutive handshakes for a single target depending on the approach.

The TUM goscanner checked during scan time whether the certificate provided by the server is *valid*, utilizing only functions provided by the standard Golang library. Essentially, it confirmed that the certificate meets basic X.509 requirements, a chain of trust existed to the Debian X.509 root store relying only on the provided peer certificates, and ensured that the server performed a correct signature with the certificate's private key. Moreover, the library checked if the requested domain appears as a Subject Alternative Name (SAN) when scans are conducted with SNIs.

2.5.1 Top and Blocklist Measurements

For some of our studies we used top and blocklist as input for our measurements. This was our approach to reduce the scanning and evaluation costs by limiting our scans to a subset of the Internet, resembled by the input lists, and still obtain a representative data set. We used this approach when repeatedly scanning the same target; e. g., with multiple CHs. Using blocklists as input has another advantage, sometimes the entries

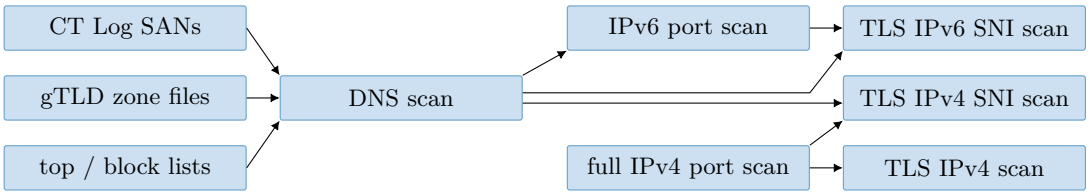


Figure 2.1: Internet-wide DNS and TLS Scanning Pipeline

are published with ports. This was the case for some of the servers on our C2 blocklists. In general, we used port 443 for scanning, but used the published ports instead if available.

Focusing Internet scans on top and blocklist allows to reduce scanning and evaluation costs while preserving a representative data set to evaluate subsequent approaches. If the required information can be obtained efficiently—e. g., with a single handshake—TLS measurements can be scaled up Internet-wide.

2.5.2 Internet-wide Measurements

For studies requiring information obtainable with a single handshake, e. g., if we were mostly interested in the certificate of a server, we collected the data with the help of Internet-wide DNS and TLS measurements. The scanning pipeline is illustrated in Fig. 2.1.

Input for the DNS scans were:

ii) domains from 1.1×10^3 gTLD zone files obtained from the Centralized Zone Data Service (CZDS), including *.com*, *.net*, and *.org*;

ii) top and blocklists, *i.a.*, Majestic [98], Umbrella [52], Chrome UX Report [75], Chromium HSTS Preload [48], Cloudflare Radar [53], and Openphish [122]; and

ii) domains we extracted from the SAN extension in Certificate Transparency (CT) log certificates.

We resolved the more than 600 million domains to get the A and AAAA records. For our TLS measurements, we focused on TLS-capable addresses on port 443, the standard Hypertext Transfer Protocol Secure (HTTPS) port, expecting a high rate of deployed servers supporting TLS. For IPv4, we used ZMap [63] on the complete address space to identify servers with an open port 443. Afterward, we used the TUM goscanner to conduct a full TLS handshake without a SNI. We relied on the previous A and AAAA DNS resolutions to collect further TLS data bound to specific domains. We scanned each resolved IPv6 address for an open port 443 using ZMapv6 [72]. Afterwards, we filtered

all (domain, IP address)-pairs for targets with an open port 443 based on our previous ZMap(v6) scans and established a TLS connection to all targets with the respective domain as SNI. Selecting targets with an open port reduced the scanning overhead and network load.

2.5.3 Ethical Considerations

We reduced the impact on third parties by following the best practices described by Durumeric et al. [63]. Our work does not harm individuals or reveal private data, as [59] and [126] cover, and focuses on publicly reachable services. We used rate limiting, maintained a blocklist, used dedicated scan servers with abusive contacts, configured informative reverse DNS entries, and hosted websites that informed scanned parties about our research. We also provided contact information for further details or scan exclusion. Our input targets were randomly shuffled to spread the interference on a single party to a more extensive time frame; e. g., if multiple domains resolve to the same IP address.

2.6 Active TLS Fingerprinting

Active fingerprinting "is the process of actively interacting with the target entity" [162] to reveal undisclosed information, like the type and version of software running on a device and it can be used for classification purposes. TLS is designed as a client-server architecture, hence, it is only possible to actively interact with the server. Consequently, *Active TLS Fingerprinting* is the process of actively interacting with a TLS server to reveal undisclosed information. It is an approach utilizing the TLS metadata revealed in the TLS handshake and a straightforward yet effective methodology for examining TLS servers to uncover information about their deployment. TLS server metadata that depends on the server's implementation, configuration, and hardware can be used to identify hidden relations and similarities among different server setups. As a result, passive TLS fingerprinting has various applications in the field of security (*cf.*, [17, 18, 21, 83, 104]), such as identifying hosts running vulnerable software, detecting network anomalies, and uncovering malicious entities. Effective Active TLS Fingerprinting can contribute to a better understanding, modeling, and securing of the Internet. If fingerprints can indicate the level of trust in infrastructure, they become valuable cyber-threat intelligence, particularly given cybercriminals' increasing use of TLS [136].

In this section we examine various methodologies for acquiring and leveraging metadata used for Active TLS Fingerprinting and investigates how effective fingerprinting can be conducted. In general, approaches have to balance the necessary information collection

for fingerprinting and the associated scanning costs and we categorized them into two main strategies: fixed-probe and exhaustive fingerprinting.

Fixed-probe fingerprinting involves using a small set of fixed requests designed to be good in differentiating server deployments. It is a lightweight approach intended for Internet-wide scans. However, using a fixed set of probes may lead to redundant data collection and overlooked information, potentially limiting the performance of subsequent applications such as detecting C2 servers. On the other hand, exhaustive fingerprinting performs resource-intensive scans that dynamically adapt to the server until the maximum information about the server deployment is acquired. The effectiveness of exhaustive TLS scanners depends on their model of the TLS stack and their ability to handle a wide range of server responses. However, these approaches are generally more resource-intensive and may have variable scan durations due to their dynamic nature.

Some potential use cases for Active TLS Fingerprinting include:

vi) IDSs use fingerprinting to identify servers seen in network traffic and compare the results with known malicious fingerprints on demand;

vi) security researchers utilizing fingerprints obtained from Internet-wide measurements to identify unknown threats; and

vi) regular server monitoring to detect unintended software changes or malware infections when deviations from a fingerprint baseline are detected.

Internet scanning companies, such as censys.io, have recognized the importance of the information provided by Active TLS Fingerprinting and have incorporated tools like JARM [19] into their portfolio, as shown by their host data definition [45]. JARM is an example of an open-source fixed-probe Active TLS Fingerprinting tool that has recently gained prominence in its usage.

In Section 2.6, we provide the following topics:

vi) a detailed definition of Active TLS Fingerprinting and the requirements for a fingerprint;

vi) EFACTLS as a methodology for effective fixed-probe fingerprinting based on an extensive feature extraction and heuristically optimized CHs serving as scanning probes;

vi) DissecTLS as a methodology for effective exhaustive fingerprinting based on a model of the TLS stack on a server;

vi) an approach and measurement study to confirm the consistency requirement of EFACTLS and DissecTLS through Internet measurements conducted over time; and

vi) approaches for comparing the effectiveness of active TLS scanners for fingerprinting based on a local testbed, Internet measurements, and a C2 classification use case.

2.6.1 Definition and Requirements

Active TLS Fingerprinting exploits the TLS protocol to discover similarity relations among servers by fingerprinting their Server Behavior. We define *Server Behavior* as the totality of the capabilities, the interpretation (deviations from the standard or implementation of undefined parts, such as the order of extensions) and the configuration of TLS on a server, which can influence the outcome of the TLS handshake. Our work assumes that every TLS server has a specific Server Behavior that depends on the implementation, capabilities, and configuration of the TLS library, hardware, and application utilizing the TLS. Identifying these behaviors allows characterizing server deployments either directly or in conjunction with additional data (e. g., obtained on the HTTP layer). A peculiarity of Active TLS Fingerprinting is that TLS reveals little about the server per design. Clients initiate TLS handshakes, and servers only need to react to the initial handshake request (e. g., a server chooses one cipher from a list that the client previously proposed). Therefore, the Server Behavior we want to fingerprint is not directly revealed by the server; only the reaction to different requests (i. e., CHs) is visible. Using multiple CHs increases the acquirable knowledge and coverage of the Server Behavior.

We define *Active TLS Fingerprinting* as a concatenation of features that depend on the Server Behavior. The concrete format is irrelevant as long as the same Server Behavior always result in the same fingerprint. Although multiple Server Behaviors can have the same fingerprint, different fingerprints should only be caused by different Server Behaviors. However, the level of descriptiveness has to be considered and there is the risk of including too descriptive features of the Server Behavior into the fingerprint that provide no value for further use cases. For example, including the IP address as feature would result in very specific fingerprints; however, it would not be useful for use cases that try to detect an implementation deployed on multiple IP addresses.

In summary, we propose that the following requirements are essential for Active TLS Fingerprinting and should be considered when deciding which features to include in the fingerprints:

Table 2.1: Summary of related active Active TLS Fingerprinting tools (*cf.*, Sosnowski et al. [8]).

SSLyze	testssl.sh	TLS Prober	JARM	EFACTLS	DissecTLS	
E	**E**	**F**	**F**	**F**	**E**	(**F**)ixed Probes / (**E**)xhaustive Scanning
✔	✔	✖	✔	✔	✔	Covers TLS Configuration
✖	✖	✔	✔	✔	✔	Covers TLS Interpretation
✔	✔	✖	○	✔	✔	Supports TLS 1.3
≈ 430[2]	≈ 155[2]	10–295[1]	10	10	≈ 24[2]	CH Usage
[58]	[170]	[108]	[19]	[8]	[5]	Reference

Legend: ✔ Yes ✖ No ○ Partly

[1] TLS Prober has a "quick scan" mode that stops after 10 probes suggest the same implementation.

[2] Experimentally determined in a study on the Tranco top 10k domains published in [5].

i) Consistency. Fingerprinting the same Server Behavior will result in the same fingerprint. If multiple handshakes are necessary for fingerprinting, each handshake should connect to a server with the same Server Behavior.

i) Unambiguity. The relationship between Server Behaviors and fingerprints must be a function. This means that while multiple Server Behaviors can produce the same fingerprint, only distinct Server Behaviors can yield different fingerprints.

i) Usefulness. The relation created by matching fingerprints is useful for the intended use case. This requirement is arguably the most challenging to fulfill. We show the usefulness of our fingerprinting approaches with a C2 detection study in Section 2.6.8, verifying the usefulness for this particular use case.

2.6.2 Related Approaches and Tools

This section presents related TLS scanning and fingerprinting approaches. We selected them because they all extract metadata from the TLS layer that can be used for fingerprinting. Additionally, they have in common to implement specialized probing mechanisms that are able to extract more information from the TLS layer than it is possible by conducting a single or trivial handshakes. However, their different goals, requirements, and approaches result in different models of the TLS stack on a server, that can result in a varying effectiveness of the active fingerprinting. We summarize our findings on related approaches and tools in Table 2.1. None of the other Active TLS Fingerprinting works has used Entropy or an equivalent metric to evaluate or optimize their approach.

To the best of our knowledge, the closest related work we can directly compare our EFACTLS approach to is the JARM tool developed by Althouse et al. [19]. It is a pop-

ular open-source tool for TLS server fingerprinting. Compared to JARM, our approach differs in the concrete choice of CHs and the extracted features from the TLS handshake. They use 10 custom-defined CHs for fingerprinting that "have been specially crafted to pull out unique responses in TLS servers" [19]. In contrast to EFACTLS, they do not complete the TLS handshake, only use unencrypted data, and do not consider TLS alerts nor extension data besides the Application Layer Protocol Negotiation (ALPN) protocol; hence, they use only part of the data offered by TLS 1.3. We will show in Section 2.6.8 that JARM also enables C2 server detection; however, our study suggests that methods from this thesis can improve the approach's effectiveness.

A fundamentally different approach for collecting TLS data is to dynamically search for each piece and change the scanning probes during the fingerprinting based on already learned information. A scanner can adapt scanning probes on the fly if it contains a model interpreting the TLS stack on the scanned server. The scanner would use the model to interpret previous server responses and generate a new probe most likely to fill the gaps in the already collected data. This allows to exhaustively scan a server until no new information can be retrieved. However, the quality of the collected data relies on the quality of the underlying model because it has to explain any observed behavior. Adapting the scan per server can result in a variable and unpredictable scan duration. In contrast, the fixed probing from EFACTLS and JARM have a constant scan duration independent of the inner workings of TLS server implementations. Our DissecTLS [5] approach shows that it is possible to implement such a dynamic scanning approach efficiently enough to use it for large-scale measurements.

DissecTLS provided a higher C2 detection precision compared to EFACTLS and JARM (*cf.*, Section 2.6.8) at the cost of sending more requests (24 CHs per server on average [5]). Similarly, TLS server debugging tools like testssl.sh [170] or SSLyze [58] dynamically collect data about the TLS servers. Results we will present in Section 2.6.7 demonstrate that the data from both tools can be used for Active TLS Fingerprinting, although it is necessary to sanitize their output first. However, their extensive use of requests per server makes Internet-wide scanning time-consuming and ethically questionable. Additionally, their focus on the configurable part of TLS on servers (e. g., supported cipher suites) results in neglecting fingerprintable implementation-specific features like the extension order. In contrast, TLS Prober [108] solely focuses on such features. Mapping out edge cases of the TLS protocol enabled the author to fingerprint only the TLS library. The tool uses up to 295 probes to reveal implementation-specific behavior, such as the mapping of TLS messages to TLS records or how the implementation reacts to erroneous or unusual input. It is possible that the data obtained with TLS Prober can be used in conjunction with ours to improve detection further in the

future. However, the tool has not been updated recently and only supports TLS 1.2. Section 2.6.8 will present how additional data sources can improve C2 detection.

2.6.3 Fixed Probe Fingerprinting with EFACTLS

EFACTLS [8] is an Active TLS Fingerprinting approach that uses a fixed number of specifically crafted CHs as scanning probes to fingerprint a server. Only session and server independent parameters are fixed in these CHs; e. g., cryptographic keys or the SNI are configured per TLS handshake. Hence, EFACTLS falls into the category of fixed-probe fingerprinting. Another example for this category is JARM [19]. Due to the large parameter space of TLS and the fact that a server reveals only little about the Server Behavior in each response, any fixed-probe fingerprinting can only capture a part of the Server Behavior without sending an unreasonable amount of probes. However, the captured part can still be large enough for effective fingerprinting. Which CHs provide the most effective Active TLS Fingerprinting probes is an open research question and can be interpreted as optimization problem of increasing the collected information while reducing the scanning costs.

Server responses are a reaction to the initiating handshake request; e. g., a server can only choose a cipher suite that the client previously proposed. Consequently, the features obtained with a single CH are only comparable in the context of the same CH and the CH used to generate a particular responses is a feature of the resulting fingerprint. Hence, a fixed-probe Active TLS Fingerprint $f \in F$ can be modeled as a tuple of the CH used as scanning probe and the extracted handshake features:

$$F \subseteq CH \times Features \tag{2.7}$$

Extracting Features from a Single TLS Handshake

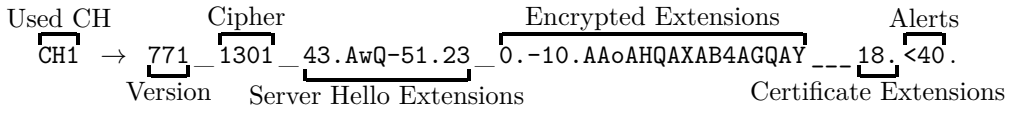
EFACTLS uses several CHs as scanning probes to perform multiple TLS handshakes with a single server. For each CH, the TLS version, cipher, and TLS extension data from different types of TLS messages is extracted from the server responses to construct a fingerprint. The extracted features are carefully selected to contain no information specific to the current TLS session, a specific server instance, or the TLS certificate such that the captured fingerprint depends only on the Server Behavior. The concrete format of the EFACTLS fingerprints is irrelevant for the purpose of fingerprinting because they only have to be compared based on equality. The only requirement for the representation is that a fingerprinted Server Behavior produces always the same fingerprint. Still, we decided to use a format that is human-readable for explainability and debugging purposes. The extracted features are the selected version, cipher suite, received alerts

Table 2.2: TLS extension values used EFACTLS for fingerprinting (*cf.*, Sosnowski et al. [8]).

ID	Name	ID	Name
1	Max Fragment Length	19	Client Certificate Type
7	Client Authentication	20	Server Certificate Type
8	Server Authentication	24	Token Binding
9	Cert Type	27	Compress Certificate
10	Supported Groups	28	Record Size Limit
11	EC Point Formats	43	Supported Versions
13	Signature Algorithms	47	Certificate Authorities
15	Heartbeat	50	Signature Algorithms Cert
16	ALPN	51	Key Share (only selected group)

TLS extension Identifiers (IDs) as specified by IANA [84].

and extension data. The extension data is represented as an ordered list of key-value pairs from the *Server Hello*, *Encrypted Extensions*, *Certificate Request*, *Hello Retry Request*, and *Certificate* TLS messages. The extension identifier serve as the key and the content of the extension as the value. However, only the content of several hand-picked extensions are included. In the EFACTLS publication we selected the extensions in Table 2.2, listing the mapping between identifier numbers and names according to Internet Assigned Numbers Authority (IANA) [84]. This list might have to be adapted to provide effective fingerprints in case of protocol changes, new developments, or novel TLS features. An example of a textually represented fingerprint is



(2.8)

Design reasons for these concrete handshake features are the following. The version, selected cipher suite, and used extensions directly depend on the capabilities and the configuration of the TLS library used by the server and are, therefore, part of the fingerprint. The content of each extension can contain information independent of the Server Behavior, and its fingerprinting value depends on the respective extension. The specific extensions listed in Table 2.2 were selected based on an analysis of the respective TLS specifications [9, 129] and observations in TLS scans. An exception to this schema is the Key Share extension, where the session-specific part is removed and only the selected group used for the Diffie–Hellman (DH) key exchange is kept. The fingerprints are defined in a format that can easily be adapted (e. g., to include values specified in future extensions). TLS alerts sent by the server are included in the fingerprints because error handling is implementation-specific. However, no fingerprint was created if the TCP layer caused the error. EFACTLS cannot differentiate whether the error was part of the Server Behavior or a nondeterministic failure of the TCP stack. The

order of extensions is considered valuable and implementation-specific information that is included in the fingerprints. Because the presence of the Status Request extension can be nondeterministic (*cf.*, [51, 8]), the extension is not included in the fingerprints, trading the information about the OCSP stapling support of a server for consistency.

Combining Features from Multiple TLS Handshakes

Fixed probe fingerprinting can only capturing a part of the Server Behavior. To increase this part and enable effective fingerprinting, EFACTLS uses multiple CHs to increase the acquirable knowledge and coverage of the Server Behavior.

A single CH can reveal only a potentially small request-dependent subset of the information about the target server, multiple request–response pairs allow the collection of complementary information and, thus, a more complete picture of the Server Behavior. Increasing the number of CHs is a trade–off between learned information and measurement costs. However, the benefit of sending multiple CHs decreases with every additional CHs one sends because of the limit to which a Server Behavior can influence the TLS handshake. Moreover, the number of CHs should be limited based on time, resources, and ethical factors. Hence, the input set CH of CHs is an optimizable parameter influencing the effectiveness of fingerprinting. Let $f(s, c)$ return the features from a server s with a consistent Server Behavior given a specific CH $c \in CH$ in a fingerprintable format. Because fixed-probe Active TLS Fingerprints should be only compared in the context of the same CH, a combination can be modeled as

$$fp(s) = \bigcup_{c \in CH} (c, f(s, c)). \tag{2.9}$$

TLS cryptographically ensures end-to-end communication with the server terminating the TLS connection; hence, all features extracted in the TLS handshake depend on a single Server Behavior. This is not guaranteed when performing multiple handshakes and Layer 3 or 4 load balancers can cause multiple servers with different Server Behaviors to respond to the same IP address. This is the main limitation of combining multiple TLS handshakes for fingerprinting because it requires that each handshake is answered by the same Server Behavior. However, we confirmed that the EFACTLS fingerprints were consistent for 99% of the targets in the measurement study described in [8] scanning top list servers for more than a year.

Fixed probe Active TLS Fingerprinting uses fixed parameters for the CH and extracts handshake features in a fingerprintable format. The static parameters of the used CH and the extracted features form the actual fingerprint. Active TLS fingerprints depend on the Server Behavior such that the same Server Behavior produces the same fin-

gerprint and different fingerprints indicate different Server Behaviors. Using multiple complementary CHs as scanning probes allows to increase the acquired information and enable more effective fingerprinting.

2.6.4 Heuristic Optimization of TLS Scanning Probes

For Active TLS Fingerprinting, the number of requests and the design of effective CHs are crucial parameters that can be optimized to maximize the amount of collectible information while minimizing measurement costs and respecting ethical aspects.

The internal mechanism of TLS servers is a black box for active scanners. Without knowledge about the implementation of every TLS server, finding the best method for fingerprinting is impossible. However, more effective fingerprints can be developed by optimizing their distinctiveness. We propose to use an heuristic design of CHs by analyzing a large pool of randomly generated candidates to find an optimal subset maximizing a given metric. The Entropy (*cf.*, Section 2.3) can serve as metric to find general-purpose CHs usable for a wide range of use cases. If the use case is known (e. g., detecting C2 servers), a different strategy could be minimizing the necessary probes needed for the classification.

Because we modeled combined fixed-probe Active TLS Fingerprints only in the context of the same Server Behavior, performance metrics like the Entropy of a concrete combination of CHs can only be computed when the same Server Behavior was scanned with each CH from the combination. However, when using random CHs the pool of candidates has to be very large because CHs can have poor performance or not be functional at all. It is possible that a random combination of parameters in a CH cannot function; e. g., performing a TLS 1.3 handshake without the required extensions. Additionally, scanning a single server with an unreasonable amount of CHs is very resource expensive and ethically questionable. Therefore, we chose a two step approach to generate the scanning probes for EFACTLS:

1. We generated 10^4 random CH candidates and scanned top list servers to gain a first impression of good-performing CHs. We iterated over the candidates with a round-robin algorithm and sent a maximum of 13 CHs per server to increase the variation of the different sets of CHs sent to a single server.

2. Afterward, we selecting the best-performing CHs and conducted a second measurement of the top list servers and fingerprinted each target with all 50 CHs. 50 CHs per server were a pure trade-off between scanning speed and data quality; still, the scan took more than four days.

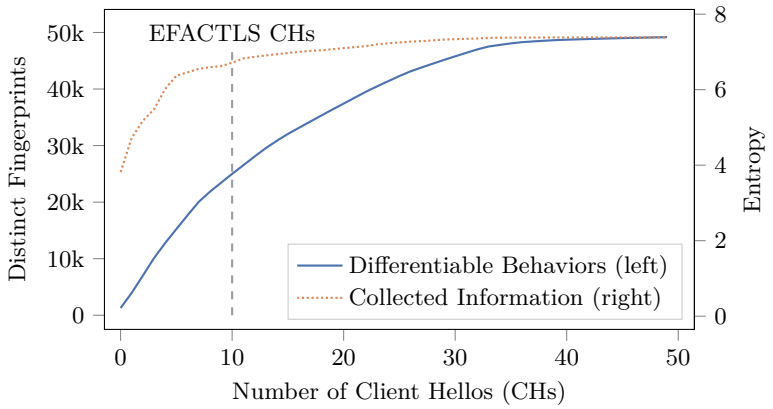


Figure 2.2: Heuristic optimization of the 10 scanning CHs used by EFACTLS based on a hill-climbing algorithm (*cf.*, Sosnowski et al. [8]).

After we collected the measurement data we continued with the actual optimization. Figure 2.2 shows the number of differentiable behaviors a subset of the 50 CHs could distinguish and the collected information according to the Shannon Entropy (introduced in Section 2.3). A simple hill-climbing algorithm generated the subsets, maximizing the number of distinguishable Server Behaviors. The algorithm worked as follows: it iteratively selected the next CH that most increased the number of distinct Server Behaviors. We considered only servers for which every CH produced a fingerprint to remove the potential bias from nondeterministic TCP errors. Figure 2.2 shows that every added CH enabled the differentiation of additional Server Behaviors; however, the information gain decreases the more CHs used. We could not reach the upper limit of distinguishable behaviors and collected information. Based on this analysis, we selected 10 general-purpose CHs with a good performance distinguishing Server Behaviors. We thought that selecting 10 CHs was adequate for our use cases because the Entropy increase was low when using more than 10 CHs, we can directly compare related work, and the number seemed acceptable for Internet scanners like `censys.io` already fingerprinting with JARM, *cf.*, [45]. We only manually adapted some cryptographic parameters of these CHs that were too CPU-expensive, such as the 512-bit version (`secp521r1` [40]) of the elliptic curve domain parameters for the precomputed TLS 1.3 Key Share. Enabling `secp521r1` would have more than doubled our scanning time.

Through the heuristic optimization of scanning probes described in this section, we gained the 10 fixed general-purpose CHs used for the EFACTLS fingerprinting approach. They are a good trade-off between limiting the number of requests and the resulting impact on the scanned infrastructure and providing a high distinctiveness of the Server Behaviors.

Table 2.3: Model of TLS configuration properties on a server and their representations (*cf.*, Sosnowski et al. [5]).

Property	Representation
Supported TLS versions	set
Cipher suites	priority list / set[1]
Supported groups	priority list / set[1]
ALPNs	priority list / set[1]
Selected extension values[2]	map(id → value)
Inappropriate Fallback support	bool
Order of TLS extensions	DAG[3] / set
Error behavior	one of {TCP, TLS, Ignore}

[1] Only sets can be collected if the server uses the client preferences.
[2] EC Point Formats, Signature Algorithms (Cert), and Heartbeat.
[3] A Directed Acyclic Graph (DAG) is used if the server responds consistently.

2.6.5 Exhaustive Fingerprinting with DissecTLS

DissecTLS [5] is an exhaustive Active TLS Fingerprinting approach that we designed to reconstruct the configuration that caused a particular observed TLS behavior as best as possible and in a scalable manner that can be used even for Internet-wide scans. To achieve such scalability, the number of requests have to be reduced as far as possible. We defined a general model of the TLS configuration on a server and try to scan in a single CH for as many parameters as possible, effectively learning different parameters in parallel. Additionally, we defined the output such that it can be used for fingerprinting; i. e., exclude session, timing, and instance related data. Depending on the previous responses from a TLS server we use the model to craft the most promising CH that should reveal new information about the server.

The DissecTLS Model of the TLS Configuration on a Server

To design an approach that is able to extract the parameters of a TLS server configuration, these parameters need to be defined first. We analyzed popular web server configurations (e. g., provided by Mozilla [111]), TLS server debugging tools (testssl.sh [170] and SSLyze [58]), passively captured TLS handshakes, and the TLS 1.2 and 1.3 specification [129, 130] to derive the model from Table 2.3. This model reflects our understanding of TLS and how it is applied in the Internet. It is not complete as discussed in Section 2.6.9. TLS servers support a set of versions and either answer with the correct version, abort the handshake, or attempt a downgrade to a lower version. There are three priority lists used in the handshake where the client offers a list of options and the server selects one according to its internal preferences. Iteratively removing each parameter from new requests that was previously selected by the server, the full list of length n can be scanned with $n + 1$ requests. This is the optimal approach using

the "lowest number of connections necessary [...] for one host", explained by Mayer and Schmiedecker [103]. However, if the server prefers client preferences, only a set of supported parameters can be acquired instead of a priority list. Clients can inform the server about their own priorities through the order of parameters in the CH. We tested whether servers respect this priority as follows: after learning at least two parameters, we also learned which one the server selected first. Then, we send a new CH where the order of the two is reversed; we know a server prefers its own preferences if this had no influence on the selection. We scan cipher suites, supported groups, and ALPNs with the 350, 64, and 27 possible values listed by IANA [85] during the development of the approach, respectively. Some servers provide the full list of supported groups [129] directly as extension, in these cases we do not explicitly scan them. However, the presence of a pre-computed key share can influence the priorities of the supported groups; hence, we collect the preference without a pre-computed key share and afterwards test whether the presence influenced the decision. Support of most TLS extensions is indicated by their presence or absence and does not need a particular logic, they just need to be triggered in the CH with their presence. Others need specific logic because they modify the encryption (Encrypt Then Mac (ETM) and Extended Master Secret (EMS)), are mutually exclusive with other extensions (Record Size Limit and Max Fragment Size), or multiple values can be send (Heartbeat). Sometimes, the content of extensions is of interest because it reveals information about the server capabilities, and in these cases we store the raw byte content. The Man-in-the-Middle (MITM) inappropriate fallback protection needs special logic because it only makes sense to send the signaling cipher [107] if multiple TLS versions are detected. Lastly, servers can respond differently in cases of problems, some report an error on the TCP layer, some send TLS alerts, and others just ignore the problematic part of the handshake (e. g., using a default value).

In summary, this model is an abstract and human-readable representation of the TLS configuration on a server that can explain its behavior in TLS handshakes.

Representing Multiple Observations of Extension Orders

The order of extensions is not defined in the TLS standard; however, we argue most servers have a consistent order as result how they are implemented in the code. We confirmed this by checking the source code of the Golang TLS library we modified to implement our tool. Moreover, in [8] we found that more than 99% of the servers in the study responded with a consistent fingerprint, therefore also a consistent order.

The presence of extensions depends on the request and not all extensions can be observed at the same time (e. g., the key share extension is only present in a TLS 1.3 handshake). This means, every response from the server reveals part of the true order and multiple

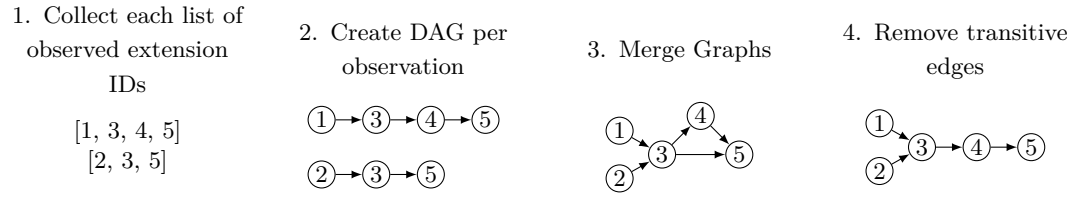


Figure 2.3: Example for merging multiple observations of TLS extensions into a single format. If the graph contains cycles after merging, the extension order is inconsistent (*cf.*, Sosnowski et al. [5]).

Table 2.4: Overview of the collected data for the stability analysis and the longitudinal C2 fingerprinting study (Adapted from Sosnowski et al. [8]).

Input Source	Total Samples	Targets	Domains	Successful (with all three scanners)		
				Total Samples	Targets	Domains
Tranco	102.3×10^6	8.4×10^6	1.9×10^6	70.1×10^6	6.0×10^6	1.5×10^6
Feodo	136.9×10^3	4.5×10^3		9.8×10^3	1.1×10^3	
SSLBL	4.9×10^3	628		111	28	
Total	102.4×10^6	8.4×10^6	1.9×10^6	70.1×10^6	6.0×10^6	1.5×10^6

observations can be combined to reconstruct the internal order on the server as close as possible. Therefore, we modeled the order of TLS extensions as DAGs for each TLS message, merged these graphs, and removed duplicates and transitive edges. If the DAG contains cycles after merging, the observations were inconsistent and the extension order cannot be reconstructed. An example for this process is illustrated in Fig. 2.3.

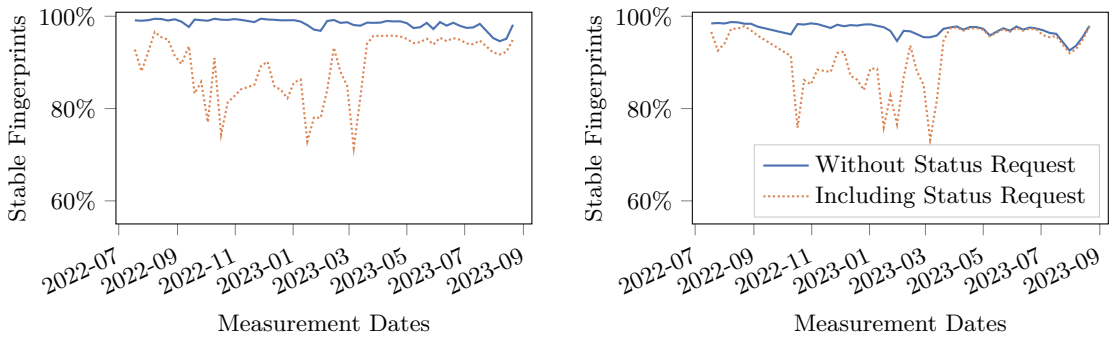
In conclusion, a DAG allows to represent multiple observations of TLS extensions in a compact format that is as close as possible to the true order on the server.

2.6.6 Consistency of Active TLS Fingerprints

Active TLS fingerprints only only provide value for identification purposes if they can be unambiguously assigned to a server, and this assignment does not change; in other words, it is stable. To investigate the applicability and stability of EFACTLS and DissecTLS on the Internet, we measured top lists and two C2 blocklists over one year.

Dataset

We scanned servers from a top list and the two C2 blocklists over 60 weeks using 57 weekly snapshots starting July 4, 2022. Three scans failed due to infrastructure problems and an issue in the scanning scripts. We skipped these weeks. This is the dataset used by the EFACTLS [8] paper and published over mediaTUM [151].



(a) EFACTLS (*cf.*, Sosnowski et al. [8])

(b) DissecTLS

Figure 2.4: Percentage of targets with the same fingerprint on the $n-1$ and nth measurement in relation to the total targets fingerprinted on both weeks.

Table 2.4 presents the number of scanned servers. A *target* is the scanned combination of IP address, TCP port, and domain name. We used the Tranco [96] top 1 Million list as the top list. The last 30 days were used for the SSLBL [15], while the current list was utilized for the Feodo Tracker [14]. We took a considerable time frame for the SSLBL because the published ports and IP addresses are just indicators and the actual blocklist consists of certificate hashes. In the following analyses, we only considered servers to be blocked by the SSLBL if they returned one of the blocked certificates. The combined list of around 1.8×10^6 weekly targets was taken as input to the scanning pipeline, as described in Section 2.5. We scanned the targets with the EFACTLS, DissecTLS, and JARM approaches. We not only used 10 CHs designed in Section 2.6.4 as scanning probes but also the 10 CHs modeled after *JARM* [19], which allows us to compare both approaches better. We consider targets as successfully scanned if a fingerprint for all three approaches was collected.

Stability Analysis

For each measurement, many servers were already seen in the last measurement ($\approx 66\%$ each week); hence, their fingerprints can be compared over time.

Figure 2.4 shows the relative number of targets remaining stable during each measurement for the EFACTLS and DissecTLS approaches. We noticed the stability dropped throughout the measurement period and could attribute this to the nondeterministic presence of the Status Request extension. Without considering the extension for fingerprinting, the targets remained stable 99% and 97% of the time for EFACTLS and DissecTLS, respectively. However, the stability dropped to an average of 89% and 92% if the fingerprinting included the Status Request extension for EFACTLS and DissecTLS, respectively. Interestingly, after March 13, 2023, the stability improved even when in-

cluding the Status Request extension. We did not change our methodology; the stability improvement was due to a change seen from Cloudflare servers. Before the date, we observed inconsistent results from Cloudflare of up to 14%, and after the date, around 3%. Cloudflare is very dominant in our data and was able to cause such statistically significant effects (i. e., the Cloudflare Autonomous System (AS) was able to serve 29% of the evaluated Tranco domains at least once). We confirmed the behavior change with Cloudflare; apparently, they improved their "OCSP Fetcher service" [158] during this time, increasing the number of certificates with OCSP staples available.

This analysis concludes that Status Request extensions should not be considered for obtaining useful fingerprints, although the merging multiple extension observations into a DAG of DissecTLS reduces the impact of the nondeterministic presence of the extension. After March 13, 2023, we could not observe a downside of including Status Request in DissecTLS. EFACTLS and DissecTLS fingerprints are, without the extension, a very stable and consistent feature to identify servers.

2.6.7 Comparing Active TLS Scanners and their Ability to Differentiate TLS Configurations

Active TLS scanners are designed to extract data from servers. In general, the more information is extracted the larger is the coverage of the Server Behavior and the better the output is for fingerprinting. However, the ability to of different scanners to extract information can vary.

We propose two approaches to evaluate the effectiveness of TLS scanners to extract information, both with their own advantages and downsides.

1. Systematically altering the TLS configuration of a server in a local testbed and testing whether the evaluated scanners can detect each permutation. Using such a ground truth allows to reason about the capabilities of each scanner. However, it is unlikely the testbed can encompass every TLS server implementation and configuration.

2. Scanning toplist servers in the Internet. Although the scanned targets remain a black box in the evaluation, it can be assumed that the more diverse the output of a scanners is, the more information it was able to collect.

Both approaches view the evaluated TLS scanners as black boxes and compared their performance to extract information by measuring their ability to distinguish different TLS server configurations. Without analyzing the scanner output we argue that whenever a scanner is able to differentiate two different TLS configurations, the scanner has detected the relevant piece of information. The more configurations it can differentiate,

the more value has its output. However, we also measured the costs of the scanner by counting the amount of requests it needed to perform the scan. The lower the costs are, the more servers can be scanned in the same time and the lower the impact is on individual servers. An ideal approach collects a high amount of information with low costs.

In the rest of this section we demonstrate the two approaches by comparing the DissecTLS, EFACTLS, JARM [19], SSLyze [58], and testssl.sh [170] approaches. We selected them because from our knowledge they are the relevant representatives that either fingerprint or reconstruct the TLS configuration. We configured DissecTLS in two versions, one tries to fully reconstruct the TLS configuration (DissecTLS), the other stops early after 10 handshakes (DissecTLS lim.). We interpret the textual output of each scanner as its representation, or fingerprint, of the server. If two outputs are equal, they detected no difference in the configuration, if they are distinct, they could capture a difference.

In summary, by comparing the ability of TLS scanners to differentiate TLS server configurations an indication of their information collecting effectiveness can be gained. A local testbed allows to leverage a ground truth and for systematic alterations of the server, Internet measurements allow for more realism in the scanned servers. Bit first, it has to be confirmed that all scanners produce a fingerprintable output.

Confirming Fingerprintable Scanner Outputs

The comparison of this section is based on the assumption that the output of the evaluated TLS scanners is only based on the TLS configuration and fulfills the requirements of a fingerprint. This means that the output contains no session or instance specific information. We propose to check this by scanning the same set of servers (testbed or top list) at least twice and confirm that the majority of the fingerprints remains the same. We confirmed this for EFACTLS and DissecTLS already in Section 2.6.6. Because JARM uses a subset of EFACTLS, its output fulfills the fingerprint requirements. For the rest of the tools we used our testbed to evaluate their output. We had to remove information regarding timing (e. g., scan time), sessions (e. g., cryptographic keys), and server instances (e. g., the domain name) from the output of testssl.sh and SSLyze to get stable results for the same TLS configuration. Additionally, we disabled the vulnerability detection of these tools.

Scanner Comparison in a Local Testbed

Using a local testbed we compared the TLS scanners based on a ground truth. We challenged them in different scenarios where we systematically made alterations to a

Table 2.5: Detected number of Nginx configurations for each test case (*cf.*, Sosnowski et al. [5]).

Test Case	DissecTLS unlimited	DissecTLS limited	EFACTLS	JARM	SSLyze	testssl.sh	Goal
TLS Versions	15	15	13	11	15	14	15
Cipher Suites	1 956	359	115	11	63	1 956	1 956
ALPNs	2	2	2	2	1	2	2
Preferences	2	2	2	2	1	2	2
Session Tickets	2	2	2	2	2	2	2
Used CHs per Server							
Minimum	8.0	8.0	9.0	9.0	423.0	9.0	
Average	14.3	10.0	10.0	10.0	450.1	132.7	
Maximum	42.0	15.0	12.0	10.0	455.0	224.0	

server and checked whether the scanners were able to detect it. The code for the experiment is published under [149].

The experiment was designed as follows: we selected a parameter we could configure on the TLS server (Test Case), launched an Nginx 1.23 docker container for each configuration we could generated for this parameter, and scanned the containers with every scanner. We used tcpdump [159] to measure the number of CHs the scanners were using. The results can be seen in Table 2.5. An ideal scanner detects every alteration made on the server and finds "Goal" number of configurations. Nginx allowed to configure four TLS versions, resulting in 15 working combinations ($2^4 - 1$). We only used six TLS 1.2 ciphers because this number was still scannable in a reasonable time. These six ciphers resulted in 1 956 configurations (every permutation of every combination of the six ciphers). ALPNs, Server Preferences, and Session Tickets were scanned either en- or disabled. The table shows that only DissecTLS and testssl.sh were able to detect every alteration we made on the server. The limited version of DissecTLS tried to detect the TLS versions, then data in extensions, and lastly the ciphers; hence, it usually detected only the first few ciphers from the server and could not detect configurations that differ in the lower cipher priorities. Testssl.sh could not detect one case where only TLS 1.3 was enabled because at the time of the experiment it included an OpenSSL version that was not TLS 1.3 capable. SSlyze was not able to detect the order of ciphers, therefore, could not detect any permutation we performed on the ciphers. The two fingerprinting approaches EFACTLS and JARM were not able to detect every alteration on the servers. This was expected as they use a fixed number of requests. However, as this experiment is artificial, it is possible that the obtained fingerprints are still good enough for fingerprinting use cases. Regarding the scanning costs, the picture is reversed. The fingerprinting tools and the limited version of DissecTLS used

Table 2.6: Comparison of TLS scanners regarding the number of detected configurations and Client Hello usage on the resolved (IPv4 and IPv6) top 10k Tranco domains (*cf.*, Sosnowski et al. [5]).

	DissecTLS unlimited	DissecTLS limited	EFACTLS	JARM	SSLyze	testssl.sh
Configurations	3.5×10^3	1.8×10^3	2.0×10^3	1.3×10^3	2.7×10^3	3.2×10^3
Total CHs	530.6×10^3	235.6×10^3	238.5×10^3	209.4×10^3	9.5×10^6	3.4×10^6
Average CHs	24.0	10.7	10.8	9.5	430.0	154.9
Total Scanned Targets						22 075

the least number of requests, DissecTLS slightly more, and testssl.sh and SSlyze being the most costly. We expect testssl.sh and SSLyze to be used on a small scale where scanning costs do not matter; however, we can see that the former is more optimized and uses fewer requests to collect more information. We can see a difference in JARM and EFACTLS regarding the maximum number of used CHs: both initially use 10 CHs, but the latter completes handshakes; therefore, we sometimes observe an additional CH from the scanner as response to a Hello Retry Request (servers can send them to request a different key share from the client). DissecTLS makes use of this TLS feature to reconstruct the supported group preferences of the server in case no key share is present because its presence might influence the decision. Therefore, we observe up to 15 CHs for the maximum of 10 handshakes.

To conclude, DissecTLS competes both with testssl.sh regarding the amount of collected information and with active TLS fingerprinting tools regarding their low scanning costs. However, this analysis only includes a single TLS implementation and artificial test cases; therefore, to get a more complete view the next section compares the scanners in a more realistic setting on toplist servers.

Scanner Comparison on Toplist Domain Servers

This section compares the five TLS scanners on the top 10k domains from the Tranco [96] toplist. Because the ground truth is unknown, only their performance to differentiate servers can be compared. The scan took 6 days to complete because of the low request rate testssl.sh and SSLyze were able to achieve.

The number of configurations each tool was able to detect and the number of requests necessary to collect this information can be seen in Table 2.6. DissecTLS was able to detect the most configurations, followed by testssl.sh. However, this does not mean a scanner collected only a super-set from another, as discussed in Section 2.6.9. DissecTLS uses just a sixth of the requests compared to testssl.sh, with 24 CHs on average. JARM used less than 10 requests on average because sometimes the TCP connection failed

and no CH was sent. In contrast to the last section, the limited version of DissecTLS performed a bit worse than EFACTLS. Apparently, DissecTLS only detects the finer details that help to differentiate TLS configurations when it completes the scan. An alternative metric to the total number of detected configurations is the Entropy (*cf.*, Section 2.3). We will later show in Table 2.8 that Entropy can provide a better indication of the collected information than the number of distinct configurations.

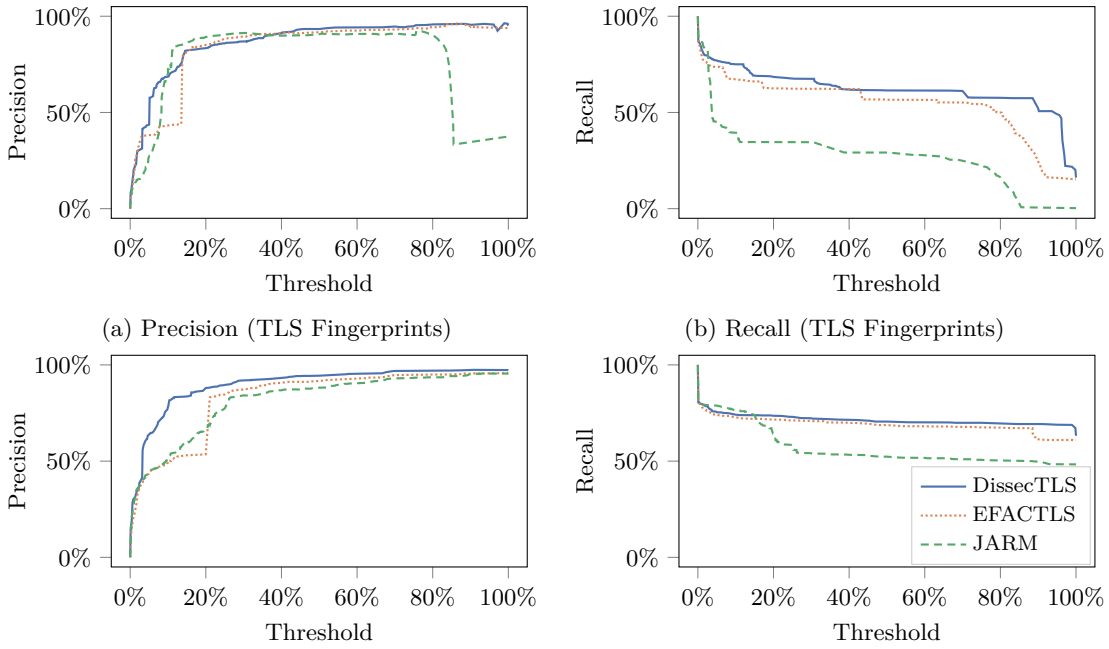
This sections showed that the dynamic scanning approach from testssl.sh, DissecTLS, and SSLyze is superior to the fixed selection of CH regarding collected data. However, this comes with increased scanning costs. We argue that only JARM, EFACTLS, and DissecTLS are resource efficient enough to be used for large-scale measurements. Additionally, in the following we refrain from limiting the number of requests of DissecTLS. While roughly doubling the scanning costs it provides a more complete; hence, a more useful view on the Server Behavior.

2.6.8 Fingerprinting C2 Blocklist Servers

Active TLS Fingerprinting can identify and track potentially malicious targets like C2 servers. This is one of the main use cases we described in our motivation in Section 2.2. We used blocklists containing C2 servers as an indicator of malicious behavior. Classifying the new additions to the blocklist allows us to compare the effectiveness of DissecTLS, EFACTLS, and JARM regarding a C2 server detection. The analysis from this section uses the measurement data described in Section 2.6.6. The data is initially used for the EFACTLS paper and published over mediaTUM [151].

We used the threshold-based classifier from Section 2.4 combined with a continuous classification over time (Section 2.4.2) to predict whether server are C2 server from one of our blocklists. A server was labeled a "C2 server" if its IP address was on the Feodo Tracker or the certificate was on the SSLBL. We evaluated the classifier on every new target added to the top list or blocklist during week $n + 1$ based on the training data from weeks $[1..n]$. We measured the precision and recall for each threshold according to Section 2.4. The threshold serves as a tuning parameter; e. g., selecting a value of 80% means a fingerprint must be observed more than 80% from C2 servers such that new observations with this fingerprint are classified as C2 server.

Figure 2.5 shows the classification results when using the fingerprints obtained with DissecTLS, EFACTLS, or JARM as classification input. Input for the threshold-based classifier was either only the TLS fingerprint from the respective tool or the fingerprint combined with HTTP data. The classifier performance significantly increases if we add HTTP data to the fingerprints by concatenating the respective TLS fingerprint with



(a) Precision (TLS Fingerprints)

(b) Recall (TLS Fingerprints)

(c) Precision (TLS Fingerprints + HTTP Data

(d) Recall (TLS Fingerprints + HTTP Data)

Figure 2.5: Precision and recall to identify new observations on our input lists as C2 servers using the respective input for the threshold-based classification.

the HTTP Server header and using the result as fingerprint. The HTTP Server header alone is not ideal for a classifier as shown by [8]. Combining the fingerprints from DissecTLS, EFACTLS, and JARM with the HTTP data achieved a maximum precision of 97%, 96%, and 95% for 63%, 61%, and 48% of the added C2 servers for a threshold of 100%, respectively. The lower recall indicates that neither fingerprinting was fine-grained enough to detect the differences in the deployments needed to identify all C2 servers. Augmenting the fingerprints with HTTP data was our solution to improve the granularity for more effective fingerprinting.

This analysis demonstrates that many C2 servers had unique TLS behaviors that allowed us to identify them via fingerprinting. We show that DissecTLS provided the most effective fingerprinting followed by EFACTLS and JARM. We presented how a classification of these servers is possible on a large scale and that such an approach can achieve high precision. Furthermore, a potential fingerprint database for C2 servers would live much longer than IP addresses on a blocklist, which means they can provide valuable information about newly deployed C2 servers until their IP address is publicly known.

Table 2.7: Precision (P) and Recall (R) for a threshold-based classification per C2 label using the EFACTLS fingerprints combined with the HTTP Server header (*cf.*, Sosnowski et al. [8]).

C2 Label	50% Threshold		70% Threshold		90% Threshold		100% Threshold	
	P [%]	R [%]	P [%]	R [%]	P [%]	R [%]	P [%]	R [%]
QakBot	99	97	99	97	99	97	99	47
BumbleBee	96	81	99	81	99	79	100	31
Emotet	84	85	88	85				
Ransomware	46	28	50	18	50	14	46	12
Dridex	6	25						
AsyncRAT	71	63	61	31	67	23	89	23
Gootkit								
DCRat	29	12	33	6	50	6	50	6
Pikabot								

Note: Some combinations resulted in no prediction which are left empty.

Multi-label C2 detection

The previous section demonstrated the general feasibility of C2 detection using Active TLS Fingerprinting using a binary classifier. A more fine-grained classification can help explain the decision of the C2 classifier and provide additional valuable information. In this section we want to demonstrate the feasibility and advantages of the multi-label classification. To simplify the analysis results we focus this section only on the EFACTLS approach. This also means that we count a target as successfully fingerprinted if only the EFACTLS produced a fingerprint. Using the other Active TLS Fingerprinting approaches should work similar, although it could result in a slightly better or worse performance.

We performed the threshold-based classification over time from Section 2.4.2 for each C2 label provided by the blocklists. This means every new observation is classified only based on the data from previous weeks. The Ransomware, AsyncRAT, Gootkit, and DCRat labels are from the SSLBL [15]. The remaining labels are from the Feodo Tracker [14]. Only this time, we calculated the evaluation metrics Precision and Recall on the combined data from every week. We focus only on four selected thresholds: 50%, 70%, 90%, and 100%. Table 2.7 presents the precision and recall per threshold and label. We can see that the high precision from Figs. 2.5c and 2.5d was due to the detection of a few C2 server types, especially QakBot and BumbleBee. We could not classify some labels (i. e., GootKit and PikaBot) and others above a certain threshold (i. e., Emotet and Dridex). In these cases, the fingerprints obtained from the new observations were either unknown to us or seen too often (depending on the threshold) from servers on top lists. Note that the sample size of the evaluation set was low for some labels; therefore, the calculated precision should be treated with care. For example, we found only 35

Table 2.8: Comparing the effectiveness of fingerprints obtained with EFACTLS, JARM, and DissecTLS considering both the dimensions of feature selection and used CHs (*cf.*, Sosnowski et al. [8]).

Feature selection	JARM		EFACTLS		DissecTLS
CH generation	JARM	EFACTLS	JARM	EFACTLS	
Unique fingerprints	41.8×10^3	50.2×10^3	57.0×10^3	67.5×10^3	159.7×10^3
Entropy	5.9	7.0	6.0	7.2	8.0
Unique C2 fingerprints	8	16	14	23	52
Unique C2 targets	32	242	38	249	306

new observations labeled as AsyncRAT, and the recall of 23% means we successfully identified 8 of them. Arguably, more than such a small sample size is needed to generalize the precision of 89% for the 100% threshold.

This section demonstrated that Active TLS Fingerprinting can be used beyond a simple binary classification and can distinguish the type of C2 servers. We showed that detecting some C2 server types works well, with a precision above 99%, while other labels could not be detected at all.

Influence of the Feature Selection and Scanning Client Hellos on the C2 Detection

In our measurement study we scanned every target with the EFACTLS approach using both our optimized CHs and CHs modeled after JARM, as well as the DissecTLS approach. Thus, we can compare the three approaches on a large data set and separately analyze the influence of the feature selection and the CH generation.

To separately analyze the impact of the extracted handshake features and the CH generation approach, we used the EFACTLS approach with the JARM CHs. This gave us two fingerprints for which we each extracted the subset of features that JARM uses to gain the perspective of the feature selection. In particular, we stripped fingerprints from alerts, any TLS message besides the Server Hello, and any extension data besides the ALPN (i. e., the IDs and the order of the extensions remained intact, and we only removed the data contained in these extensions). Table 2.8 compares how selecting features and CHs affects the fingerprinting results. DissecTLS works fundamentally differently; thus, we can only evaluate the overall performance instead of the two dimensions. EFACTLS consistently provide better results compared to JARM while maintaining the number of requests necessary for fingerprinting the same (i. e., 10 CHs). In total, EFACTLS can differentiate 62% more Server Behaviors than JARM. Considering the C2 servers, the improved differentiation resulted in 15 unique C2 fingerprints and 7.8 times more C2 servers identifiable with these unique fingerprints. In contrast,

DissecTLS identified 2.4 times more Server Behaviors than EFACTLS, resulting in 57 more uniquely identifiable C2 targets. "Unique" means we observed no overlap with any server found on a top list. Interestingly, the number of distinct fingerprints suggests our feature selection had a higher impact on the improved detection than the used CHs. In contrast, the Entropy and number of unique C2 fingerprints indicate the opposite was the case. This indicates that Entropy is the superior metric for modeling the information collected with Active TLS Fingerprinting compared to the distinct number of fingerprints.

In conclusion, EFACTLS fingerprinting tools like JARM can benefit from the advanced feature extraction and the systematic design of the CHs proposed in this work to improve the approach's effectiveness. Dynamic scanning tools like DissecTLS can surpass both approaches; however, the increased effectiveness comes with higher resource usage.

2.6.9 Discussion

By developing and analyzing Active TLS Fingerprinting approaches, we gained interesting insights into TLS server deployments, active measurements, and the Internet in general. We discuss some of them in the following paragraphs.

The Success of Random CHs

Initially, we used the standard CHs from the Go library and CHs mimicking popular browsers for fingerprinting. However, they could not extract enough information from servers to be effective in use cases because the requests were similar, focusing on a few popular TLS parameters. In contrast, the Random CHs were heuristically optimized to distinguish servers and have unusual combinations and order of parameters. They vary in the combination of TLS versions, ciphers, ALPN values, and supported groups and, sometimes, are not realistic (e. g., offer ALPNs unsuitable for web servers). Interestingly, two CHs use TLS 1.3, and none use TLS 1.2 as the `legacy_version`; neither conforms to the standard [129]. In contrast, JARM uses more realistic CHs with fewer parameter variations. They defined them through systematic subsets (e. g., the top half) or a reversed order from a fixed input.

In conclusion, the Random CHs are very successful for fingerprinting because they have a fuzzy character triggering more distinctive responses.

Adequate number of CHs for C2 detection

We designed the 10 CHs as a general-purpose configuration to provide a good base for classification. However, for specific use cases, they can be different. We scanned every server with 10 optimized and 10 JARM CHs; hence, we can recompute the classification

Figure 2.6: Influence of the CHs on the C2 detection with an 80% threshold (*cf.*, Sosnowski et al. [8]).

performance shown in Fig. 2.6 using up to 20 CHs as input for an 80% threshold. We used a similar optimization strategy to find the next CH for the classification input, as in Section 2.6.4. We chose the Entropy as a maximization metric because Table 2.8 revealed that the Entropy correlates better with the effectiveness of the C2 detection than the number of unique fingerprints. While the precision was high after three CHs, scanning with additional CHs mainly increased the number of classified servers (visible in the increasing recall). 18 CHs achieved the maximum precision and recall, but the gain was minimal after 10 CHs. Interestingly, two of the first 10 CHs were from JARM, providing a slightly higher precision (+0.8‰) than our 10 general-purpose CHs

To conclude, for the C2 detection use case, multiple CHs are necessary, but a few less than 10 would have also provided good results. Additionally, future work could implement an adaptive scanning approach where additional requests are only sent to a server if the precision of its current classification needs to be higher or the Entropy of server's fingerprint is too low.

Granularity of the Fingerprinting

For our classification problems we assigned multiple fingerprints to a single label because sometimes they uncovered fine-grained differences among the same label (e. g., the type of C2 server). In these cases the fingerprints were too specific; however, we mitigated the problem by using a large data set from the beginning to build our fingerprint database. Hence, it covered most of these variations. One example are the 2 854 EFACTLS fingerprints we collected from Cloudflare servers we identified through the AS. Mostly, this were slight variations in the Certificate Authority (CA) extension which is used to authenticate clients. Apparently, Cloudflare uses different CAs for different customers. If we observed an CA extension from Cloudflare the first time, it resulted in a new fingerprint. Arguably, the EFACTLS fingerprints are too fine-grained in this case if we only want to detect Cloudflare servers in general. However, this was not an issue

because we just included every alteration of the Cloudflare fingerprint in our database. We could not observe too fine grained fingerprints from C2 servers. The opposite was the case and we augmented the fingerprints with HTTP data and still could only detect 63% of the C2 servers (visible in the Recall). The rest either had exactly the same Server Behavior as benign servers, or the fingerprinting was not fine-grained enough to detect the difference.

Active TLS Fingerprinting approaches have to balance the level of granularity of the features included in their fingerprints. It is both possible to collect not enough or too fine-grained information to provide a good base for detection use cases. However, the latter can be compensated with a large fingerprint database to a certain degree.

Unstable Fingerprints

Some targets had inconsistent fingerprints that could be directly caused by the server or more complex setups. Sometimes, we saw indicators of load balancers in the HTTP Server header, indicating that the actual fingerprinted server changed during the scan process. Inconsistent responses are also the main limitation of our approach because it relies on the server to behave consistently and multiple TLS connections to connect to the same Server Behavior. Exhaustive scanners like DissecTLS could deal with some inconsistencies of the server if the model was designed flexible enough; e. g., DissecTLS falls back to a simple set of extensions of the server responds in an inconsistent order. However, besides the Status Request extension, inconsistent servers were rarely an issue as analyzed in Section 2.6.6).

Unstable fingerprints are a main limitation of Active TLS Fingerprinting. Exhaustive scanners could circumvent some inconsistent server behavior if it was considered in their implementation. Active fingerprinting studies should always confirm the consistency of the used fingerprints as proposed in Section 2.6.6. Even known approaches could produce unstable fingerprints when facing unforeseen developments.

Completeness of the DissecTLS Server Model

Section 2.6.7 showed that DissecTLS and testssl.sh were able to detect the most TLS configurations. However, looking into their output, no scanner provided a super-set of the other; hence, our proposed model cannot be complete. We manually investigated cases where testssl.sh was able to differentiate configurations while DissecTLS was not, and vice versa. On reason were inconsistent server responses from server as both scanners rely on a consistent view on to the Server Behavior. If servers behave inconsistently, both scanners might have collected an incomplete view of the TLS stack and reported different configurations on each connection attempt. We expect testssl.sh

is a bit more resilient to this behavior due to its excessive but thoroughly scanning in contrast to limiting the amount of CHs as possible. DissecTLS was able to find more configurations than testssl.sh; *i.a.*, through differentiating the error behavior and by merging the observed extensions into a single DAG. However, testssl.sh detected more details sometimes: e. g., it was able to detect variations for non-elliptic TLS 1.2 DH Key Agreement (KA) key sizes, collected cipher priorities per TLS version, detected typical server failures like being unable to handle certain CH sizes, differentiated whether a session resumption was implemented through IDs (legacy) or tickets, and used a service detection (e. g., detecting HTTP). To support these cases with DissecTLS, we would need to increase the number of sent CHs and implement the missing TLS features in the library. Whether the additional data would provide a benefit for use cases like the C2 detection is an open question for future work because we could not include testssl.sh in our C2 server detection study in Section 2.6.8, due to its limited scalability.

To conclude, neither scanner collected a super-set from the other and we argue that it is impossible to build an ideal scanner without knowledge about every TLS implementation and how TLS will evolve in the future.

Limitations of the Information Metrics

In this work, we used the number of fingerprints and the Entropy as metrics to compare the effectiveness of fingerprinting approaches. However, both are only useful metrics if the obtained fingerprints represent real-world server characteristics and remain stable for the same Server Behavior. Unstable fingerprints could trick both metrics into an unjustified high value. Unstable fingerprints can happen either because the methodology has flaws (e. g., it contains session-specific information and produces a new fingerprint in each connection) or because the servers produce a high entropy (e. g., by shuffling the order of TLS extensions or other parameters). Too fine-grained fingerprints would also result in a high Entropy but the fingerprints would not be useful for detection use cases. Entropy can only compare the effectiveness of fingerprinting approaches if each approach is guaranteed to provide only useful fingerprints. In the measurement study from Section 2.6.8 we confirm the usefulness of DissecTLS, EFACTLS, and JARM.

Entropy should only be applied to measure the collected information in active measurement studies if the collected data represents stable fingerprints and the granularity of the data is useful.

Alternative Comparison Metrics

We selected the Entropy to model the amount of information obtained via active fingerprinting approaches. Initially, we used the number of fingerprints as metric; however,

Table 2.8 indicates that Entropy is a better metric because it correlates more with the number of detected C2 servers. Although alternatives to Entropy were proposed it remains "the main tool in the analysis of the concept of information" [135] since Shannon's publication in 1948 [137]. An extension to our use of Entropy would be to model Conditional Entropy that considers already learned information. For example, a fingerprint might provide only a little information if we already know from the IP address prefix that the Server Behavior is most likely from a Cloudflare CDN cache. However, Conditional Entropy drastically increases the complexity of the metric because of the many possibilities to model the condition. An alternative metric is to compare fingerprinting approaches based on the performance of use cases; e. g., the DissecTLS work [5] compares approaches based on precision and recall of a C2 detection. However, such classification metrics rely on the quality of a ground truth.

Entropy has the advantage of providing a simple and intuitive metric based only on the data source itself. Using Entropy, Ground truth is optional and the metric can be calculated before an actual use case is known.

2.6.10 Conclusion

Section 2.6 investigates methodologies for acquiring and leveraging TLS metadata through large-scale active measurements for fingerprinting use cases. We describe the two general concepts of fixed-probe and exhaustive Active TLS Fingerprinting and proposed the approaches EFACTLS and DissecTLS that implement the respective concept. We described how such fingerprinting tools can be empirically compared with a local testbed and Internet measurements. A comparison with the related fixed-probe fingerprinting tool JARM revealed both of our approaches collected more information. Related exhaustive TLS scanners competed with DissecTLS regarding the amount of collected information; however, their high resource usage made them unpractical for large-scale measurements. We showed the value of Active TLS Fingerprinting with a measurement study conducted over a year on the Tranco top list and two blocklists by detection C2 servers. The precision in classifying new C2 servers added to the blocklists reached 97% and 99% for some C2 families.

Our Active TLS Fingerprinting approaches use a reasoned selection of features extracted from TLS handshakes and multiple scanning probes to construct fingerprints of the TLS stack on servers. The EFACTLS scanning probes were heuristically optimized to provide maximum information while minimizing measurement time and the impact on targets. DissecTLS, on the other hand, dynamically adapts it scanning probes based on a TLS stack model and previously learned information. The model was used to explain server responses at scan-time and to craft new requests that should reveal new data.

This thesis describes in detail how effective Active TLS Fingerprinting can be conducted and demonstrates the applicability of the approach to real-world classification problems, such as C2 detection, demonstrating its relevance in security contexts. Moreover, the extended feature extraction and improved CH design can improve existing Active TLS Fingerprinting tools while maintaining their scanning effort. Additionally, we show that an exhaustive TLS parameter scanner can be implemented efficiently enough to be used on a large scale. In the future, Active TLS Fingerprinting can help to acquire a global view on the TLS server configurations to deepen our understanding of the TLS Ecosystem and aid in security relevant use cases.

2.7 Propagating Threat Scores with an Internet-wide Graph

The Internet is shaped by independent actors and heterogeneous deployments. The wide adoption of TLS [94] has led to the development of an interconnected ecosystem that, in conjunction with the DNS and a global X.509 PKI, underpins the majority of the current Internet.

Acquiring a comprehensive view on this ecosystem can enable novel possibilities to identify malicious activity. Actively collected Internet-wide DNS and TLS metadata can provide the basis for such large-scale analyses because interesting metadata about servers, their configuration, or the actor behind the deployments (*cf.*, Sections 2.2 and 2.6) can be collected, especially considering the increased usage of TLS by cyber-criminals [136]. Blocklists like the abuse.ch SSLBL [15] (listing certificates used by botnet C2 servers) indicate that certificates can reveal relations among C2 server IP addresses. Moreover, domains might resolve to these IP addresses and could be embedded in other certificates. Leveraging such relations can provide a detailed view of malicious infrastructure and offer valuable cyber-threat intelligence. However, a comprehensive model and efficient approach for navigating the vast volumes of data are necessary to provide this information.

This section investigates a graph-based model of the TLS Ecosystem that utilizes the relationships between servers, domains, and certificates. A Probabilistic Threat Propagation (PTP) algorithm is then used to propagate a threat score across entities to find new, potentially malicious servers.

In Section 2.7, we provide the following topics:

vii) A versatile graph model of the TLS Ecosystem built around the deliberate actions of an actor controlling a domain, IP address, or certificate;

vii) The application of a message-based implementation of PTP to propagate a threat score throughout the graph using blocklists as input.

vii) A one-year-long measurement study covering 13 monthly Internet-wide DNS and TLS measurements. We analyzed newly found domains and IP addresses with a high threat score via manual inspection and external threat intelligence services. Additionally, we evaluated whether they would appear on the respective blocklists over time.

2.7.1 The TLS Ecosystem

The TLS Ecosystem is an interplay of the DNS [60], X.509 PKIs [38], the TLS protocol, and the applications using TLS (e. g., HTTPS) to provide a level of trust in our communication over the Internet.

IP addresses are used to route requests to the designated server. These addresses are human-unfriendly Bit sequences; hence, among other goals, the DNS was designed to map access to a resource, identified with a human-readable domains, to one or multiple IP addresses. This means, a client sends the request for a resource to one of the resolved IP addresses. However, clients do not know if the received response actually came from a server eligible to serve the requested resource. A widely adopted solution is using the TLS in combination with a x509 PKIs to encrypt the communication and authenticate the communication peer. In this case, the TLS library on the client verifies whether the server is in possession of a certificate that contains the requested domain as SAN [38] and if it trusts the certificate according to its PKI. PKIs are a concept to transitively pass on trust using digital signatures from a set of pre-configured root certificates and clients only need to construct signature chains up to one of the roots to verify the server is allowed to serve the SANs listed in the certificate. We call this combination of concepts and technologies the TLS Ecosystem.

2.7.2 An Internet-wide TLS Ecosystem Graph Model

We propose modeling actively collected DNS and TLS metadata as a Labeled Property Graph to streamline processing, simplify inspection, and benefit from well-known graph algorithms, ultimately aiding security-relevant use cases. The abstraction as a graph allows to combine various measurement data into a single Internet-wide graph.

Why Modeling a Labeled Property Graph?

Modeling real-world data as graphs is intuitive, especially if the data source is already a network like the Internet. The idea is not new; e. g., the Resource Description Frame-

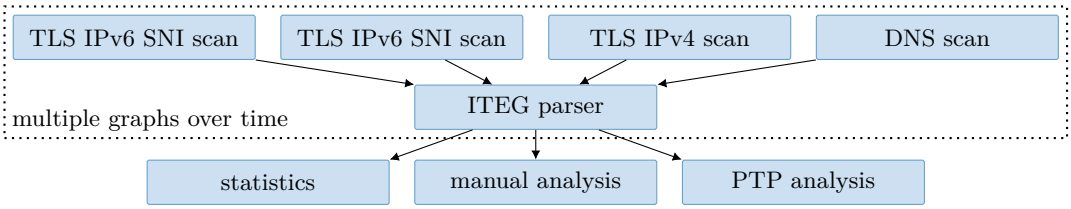


Figure 2.7: Graph Processing Pipeline. It takes the scanning pipeline from Fig. 2.1 as input.

work (RDF) [100] tries to model arbitrary resources on the Internet as a single Internet-wide graph.

A *graph* is an abstraction and simplification that expresses connected data using only *nodes* (or *vertices*) and *edges* [133]. The most popular graph model variant is a *Property Graph*, where nodes and edges can contain properties (expressed as key-value pairs), edges are named and directed, and they always have a start and end node (*cf.*, Ref. [133]). A *Labeled Property Graph* additionally contains labels attached to nodes that distinguish different types. We selected a Labeled Property Graph for our modeling because it is a simple and intuitive representation powerful enough to express the complex data we collect through our active measurements.

Graph Processing Pipeline

In the following, we demonstrate how such a graph model can be constructed using the Internet-wide DNS and TLS scans described in Section 2.5.2 as input. In short, the scans started with domain lists covering a large portion of the Internet and resolved each domain in the DNS scans to one or more IPv4 and IPv6 addresses. Afterward, we conducted Internet-wide TLS scans of the entire IPv4 address space and all the (IP address, domain) tuples we collected during the DNS scan. In the second case, we used the domains as SNIs to collect certificates bound to specific names. The TLS scans targeted TCP port 443, and we stored the received HTTP headers. These scans served as input for our graph processing pipeline as shown in Fig. 2.7. An Apache Spark [22] application transformed the DNS and TLS scan results into a property graph and performed a PTP analysis on the data. The graph and the calculated PTP scores were the basis for our later analyses.

The ITEG Schema

This section describes the concrete ITEG model and the reasons for its design.

Essentially, the TLS Ecosystem links resources (identified by domains) to physical locations (IP addresses) and ensures, with the help of certificates, that only servers eligible

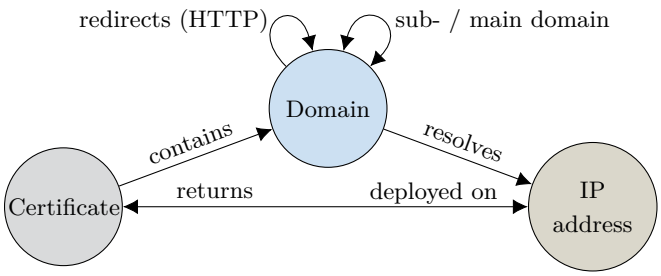


Figure 2.8: ITEG Schema (*cf.*, Sosnowski et al. [1]).

to serve a particular resource do so. In other words, it creates a cyclic relation between a domain, certificate, and IP address, illustrated in Fig. 2.8. A domain can *resolve* to one or multiple IP addresses via the DNS. When performing a TLS handshake with the resolved IP address, the server *returns* a certificate to authorize its response. The certificate *contains* one or multiple domains with the SAN [38] extension. Domains are hierarchical per-design expressed in the *subdomain* and *main domain* relation (e. g., `www.example.org` has an edge to `example.org`). Because we performed our scans on port 443, we received several *redirects* via the HTTP to different domains (e. g., from `example.com` to `example.org`). An important design criterion was that each edge should reflect an actor's intent who is controlling the node. For example, the owner of a domain controls which IP addresses are resolved; however, the owner of an IP address cannot control which domains point to the address. Similarly, the server behind an IP address controls which certificates it returns, and the creator of a certificate chooses the included domains and on which servers the private key is deployed. We decided not to distinguish between valid or self-signed certificates to simplify the model and because the graph already contains this information to a certain degree, as discussed in Section 2.7.7.

In summary, the TLS Ecosystem can be modeled with an Internet-wide Labeled Property Graph. We demonstrated this with our ITEG model which uses Internet-wide DNS and TLS measurements as input. The ITEG is designed to search for malicious activity; hence, relations in the ITEG express a deliberate action of the actor controlling an entity. Alternative use cases could require different graph models.

2.7.3 Probabilistic Threat Propagation on the ITEG

In this section, we describe how to use the graph structure of the TLS Ecosystem to propagate threat indications between connected nodes and find new, potentially malicious nodes given a set of input hints. We used the PTP algorithm developed by Carter et al. [44] to achieve this propagation. They originally designed their approach

Algorithm 1: Message-based Approximate PTP

Input: The $ITEG$ (without loops), a blocklist as $Input$, and a convergence parameter ε

repeat

  for $(src, dst) \in ITEG.edges$ **do**

    $message \leftarrow dst[score]$

    $prev \leftarrow$ score send over edge (dst, src) in the previous iteration

    if $\exists prev$ *and* $dst \notin Input$ **then**

      `// remove error caused by` $prev$

      $message \leftarrow message - \frac{prev}{dst[outDegree]}$

    send $message$ to $e.src$

  for $n \in ITEG.nodes$ **do**

    $M \leftarrow$ messages received by n

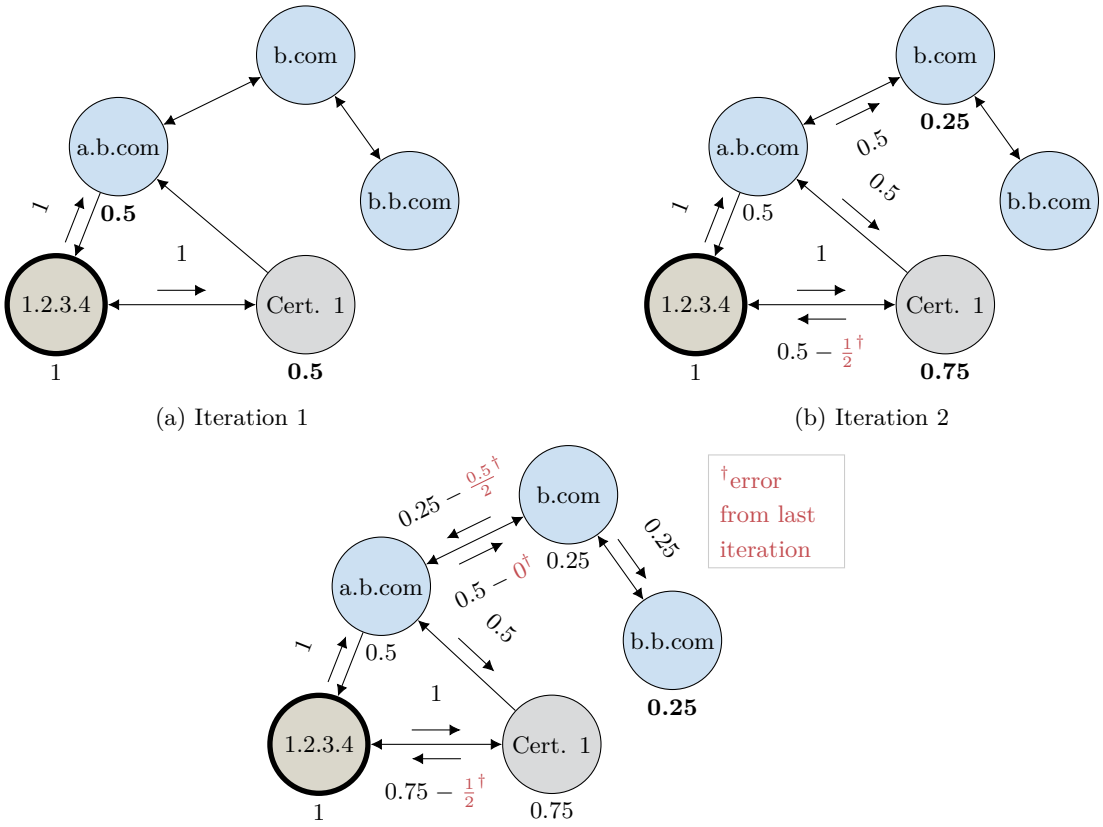
    $$n[score] \leftarrow \begin{cases} 1 & \text{, if } n \in Input \\ \frac{1}{|M|} \sum_{m \in M} m & \text{, otherwise} \end{cases}$$

until *no score changed more than* ε

to propagate a threat score among a graph derived from IP addresses and domains a web proxy server observed. However, we will show that it can be also used on the ITEG.

The directions in the ITEG express a deliberate action of someone controlling a node; therefore, we can use the edges to propagate a score and find other relevant nodes. We decided to propagate scores in reversed graph direction. The intuition of this decision was that nodes deliberately pointing to a known malicious node might also be malicious, but the contrary can not be assumed. For example, a benign actor cannot prevent malicious domains from being resolved to his benign IP addresses. Although, domains resolving to a known malicious IP address might be misconfigured, compromised, or even belong to the respective malicious actor. The other ITEG relations follow the same intuition.

We implemented the *Approximate Inference* ($O(n)$) of the PTP algorithm [44] because the *Exact Inference* ($O(n^2)$) would not scale to our large graph. We realized it as a message-passing algorithm, described in Algorithm 1. Our PTP implementation runs for multiple iterations until convergence. We detected a convergence if neither node changed its score more than an input parameter ε. Additional inputs are the ITEG without loops (edges from and to the same node) and a blocklist from which the scores should spread (all nodes on the blocklist will have their scores fixed to one throughout all iterations). Some values were pre-computed to save time, e. g., the *outDegree* of a node. For each iteration, a node would send its score to all nodes pointing to it; in the case of bidirectional edges, we subtracted the portion of the score that was directly caused by the destination in the previous iteration, reducing the error created by nodes falsely increasing their own score. Limiting this error is a central design aspect of PTP



(a) Iteration 1

(b) Iteration 2

(c) Iteration 3

Figure 2.9: Example PTP on the ITEG. Scores of zero are hidden (*cf.*, Sosnowski et al. [1]).

(more details in Ref. [44]). However, when using the *Approximate Inference*, cycles in the graph can still cause scores to flow back to the originating node. The approximation is the reason we need ε because the graph does not converge to a final state, the error is only reduced in each iteration. However, Carter et al. [44] argue this error is negligible. Then, we computed the average of received scores and assigned it as the new node score. Finally, nodes with high scores can be analyzed.

Figure 2.9 shows an example run over three iterations:

1. Initially, the IP address *1.2.3.4* is placed on a blocklist, and thus, has a fixed score of one. In the first iteration, this score is passed on to all neighbors. The recipients *a.b.com* and *Cert. 1* will also receive a score of zero from *b.com* and *a.b.com*, respectively. We send scores in reversed edge direction; hence, *a.b.com* receives no score from *Cert. 1*. Then, the new scores of *a.b.com* and *Cert. 1* are set to *0.5*, the average of received messages.

2. In the second iteration, *Cert. 1* will increase its score to 0.75, as *a.b.com* now sends a score of 0.5 instead of zero. Similarly, *b.com* will increase its score to 0.25. We ignore messages sent to *1.2.3.4* because its score is fixed to one.

3. In the third iteration, we observe an interesting effect between *a.b.com* and *b.com* due to the bidirectional edge. Without the PTP error correction, the 0.25 score from *b.com* would flow back to *a.b.com*, falsely increasing the score in every iteration. The *b.com* node has an outgoing degree of two, hence the portion received from *a.b.com* is $\frac{prev}{outDegree} = \frac{0.5}{2} = 0.25$, resulting in a score of zero sent to *a.b.com* after the error correction. In the example, the scores will remain stable after the third iteration. However, the third iteration also shows the downside of the *Approximate Inference*, as we can see a score of 0.25 falsely flowing back to *1.2.3.4* via the cycle over *a.b.com* and *Cert. 1*. Although, it is irrelevant in the example since *1.2.3.4*'s score is fixed.

In conclusion, a message-based PTP allows the propagation of a threat score across the ITEG such that new, potentially malicious IP addresses, domains, and certificates can be found. Existing threat intelligence serves as algorithm input.

2.7.4 A Longitudinal Study to Evaluate the ITEG Modeling

To evaluate the effectiveness of our graph-based modeling for security relevant use cases, we performed 13 monthly Internet-wide DNS and TLS measurements starting January 1, 2023. This data was the basis of 13 ITEGs, which modeled the TLS Ecosystem during the respective month. For each ITEG, it took approximately one week to finish scanning, parsing, and calculating the PTP scores. We could have created at most one graph every week, but we decided to do it once a month to minimize computational and storage costs. However, this could mean we may have missed some findings.

Before we present our results, we want to provide an explanation to help interpret the results. Our methodology allowed us to find numerous suspicious domains, IP addresses, and certificates. However, it is important to note that there is no definitive way for us to determine if a suspicious entity is actually malicious or not. To gain insights into the nature of these suspicious entities we will be using external threat intelligence services to check whether the entities also attracted the attention of other actors searching for threats on the Internet; but again, their output is usually only an indicator as well, and it remains unclear whether a domain or IP address is truly malicious or harmless. Nevertheless, by highlighting the overlap between our findings and the data known to these services, we aim to illustrate that our approach can provide valuable indications of malicious activity.

Table 2.9: Overview of the scans performed to create the ITEG from Jan. 1, 2024 (*cf.*, Sosnowski et al. [1]).

	Total Measurements	Success Rate
DNS Scan	6.3×10^8	78.1 %
⌞ CT Log SANs	4.9×10^8	72.1 %
⌞ domain lists	2.8×10^8	95.5 %
TLS IPv4 SNI Scan	5.5×10^8	90.7 %
TLS IPv6 SNI Scan	1.5×10^8	83.1 %
TLS Full IPv4 Scan	5.4×10^7	77.2 %

Table 2.10: Overview of the ITEG from Jan. 1, 2024. Listing the number and distribution of nodes and edges per type (*cf.*, Sosnowski et al. [1]).

(a) Nodes

Node Type	Amount	Distribution
Total nodes	9.0×10^8	
⌞ domains	6.3×10^8	70.0 %
⌞ certificates	1.7×10^8	19.1 %
⌞ IP addresses	9.8×10^7	10.9 %

(b) Edges

Edge Type	Amount	Distribution
Total edges	3.2×10^9	
⌞ resolves	1.3×10^9	39.6 %
⌞ returns	4.0×10^8	12.6 %
⌞ deployed on	4.0×10^8	12.6 %
⌞ contains	3.6×10^8	11.4 %
⌞ main domain	3.6×10^8	11.2 %
⌞ subdomain	3.6×10^8	11.2 %
⌞ redirects	4.5×10^7	1.4 %

Data Overview

This section provides an overview of collected measurement data, the resulting ITEG, and the blocklist data used as input for the PTP algorithm. We will analyze multiple graphs over time in Section 2.7.6; however, for simplicity and because we mainly analyzed our latest graph in the following sections, we focus this overview on data from January 1, 2024. The statistics are similar for the other time frames.

Table 2.9 presents an overview of the measurements necessary to create a single ITEG. It took five days, on average, to complete all scans. We conducted 6.3×10^8 DNS resolutions of domains extracted from CT Log SANs and downloaded from external services. We scanned domains present in both sources only once. The CT Log SANs provided us with the most domains. However, the success rate was higher for the downloaded domains. We used the A and AAAA resolutions for our TLS SNI scans over IPv4 and IPv6 only if we previously detected an open port 443 (*cf.*, Section 2.5.2). Interestingly, the success rate over IPv4 was higher than over IPv6. The full IPv4 address space scan had the lowest success rate compared to the other TLS scans. Although we detected open HTTPS ports, we observed TCP resets, TLS protocol errors, or no response, and the connection ran into a timeout.

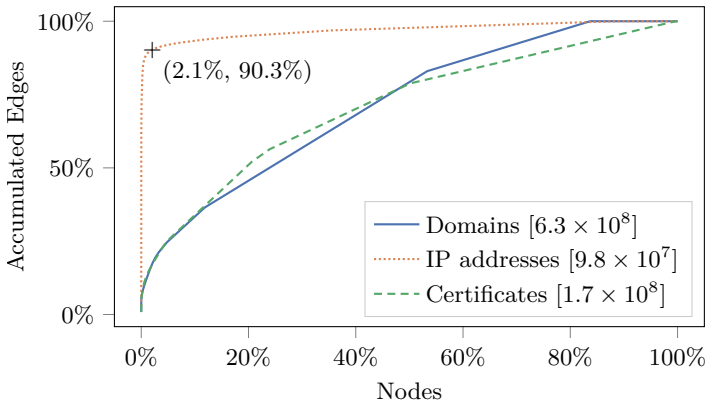


Figure 2.10: Nodes ordered by the fraction of edges they accumulate. The legend contains the total number of nodes (*cf.*, Sosnowski et al. [1]).

Table 2.11: Used blocklists, number of listed entries, the portion also observed in the ITEG, and Iterations necessary until PTP convergence for the dataset from Jan. 1, 2024 (*cf.*, Sosnowski et al. [1]).

Blocklist	Entries	Observed	Iterations
abuse.ch Feodo	174 IP addresses	34 (19.5,%)	12
Blocklist.de Strongips	472 IP addresses	161 (34.1%)	9
abuse.ch SSLBL	5 577 certificates	19 (0.3%)	14
Openphish	9 289 domains	3 461 (37.3%)	42

We used the actively collected data from Table 2.9 to create an ITEG. Table 2.10 shows an overview of the number of resulting nodes and edges. In general, the graphs contained fewer entries because we included information only once. The table reveals that domains and their resolutions dominated the graph. We observed almost twice as many certificates as IP addresses, highlighting the importance of SNI scans to get a comprehensive view of the TLS Ecosystem. To provide another perspective on the ITEG, we investigated node degrees in Fig. 2.10. Node degrees are a straightforward but effective metric for understanding the structure of a graph. The *inDegree* is defined for each node as the number of incoming edges. The figure shows how the edges in our graph are not evenly distributed and accumulate around a few nodes. In particular, 90% of all edges pointing to IP addresses accumulate on only 2% (1.3×10^7), meaning that few addresses were responsible for most connections, resulting in a high centralization. We could attribute the top two IP addresses to a domain parking service from GoDaddy, as shown by Zirngibl et al. [175]. The rest of the top ten addresses were from Cloudflare and website hosters (i. e., Wix.com and Squarespace).

Table 2.12: Suspicious groups of domains and IP addresses identified by a high and uniform threat score (*cf.*, Sosnowski et al. [1]).

Suspicious Cluster	Score	Size
1. Seemingly random domains resolving to an IP address with a blocked certificate	100%	1.5×10^5
2. `unbouncepages` subdomains	33%	3.8×10^4
3. IP addresses without known domain returning a blocked certificate	100%	2.7×10^4
4. Seemingly random domains redirecting to a blocked Openphish domain	51%	3.1×10^3

Running PTP with Blocklists as Input

On the ITEG, we ran the PTP algorithm described in Section 2.7.3 for each blocklist in Table 2.11. We downloaded each list daily and merged all entries from the last month as algorithm input. For example, if the scans for the ITEG on May 1 were finished on May 7, we considered the daily blocklist from Apr. 1 to May 7 as PTP input. As seen in the table, our scans observed only a fraction of the blocked entries. Unresponsive IP addresses, unresolvable domains, and unseen certificates were not included in the graph. The low rates were expected, especially for the SSLBL, because certificates are a strong indicator that do not have to be removed timely after C2 servers stop using them. The PTP algorithm worked as follows: we fixed the threat score to one on each node on the input blocklist. Then, we ran the algorithm using $\varepsilon = 0.01$ until convergence; this took 9 to 42 iterations, depending on the input.

This section gave an overview of the collected data necessary to create a single ITEG and to run the PTP algorithm. Statistics about the graph's structure revealed a high centralization. In the following sections, we will evaluate the generated graph and calculated threat scores more closely.

Manual Analysis of Suspicious Clusters

By manual inspection we found four outliers that the ITEG modeling and the PTP scores revealed in the scans from January 1, 2024.

As shown in Table 2.12, the first outlier were 1.5×10^5 seemingly random domains that resolved to a single IP address returning a blocked certificate (e. g., `figtbnfjxbqjyl[.]in`). A blocked certificate has a SHA-1 hash that was published on the abuse.ch SSLBL blocklist. We did not see these domains in any other context, so the PTP algorithm assigned a threat score of 100%. We extracted only 0.7% of these domains from the CT log; most originated from the zone files retrieved from the CZDS. We assume this large group of domains was automatically generated. The second outliers were 3.8×10^4 subdomains from `unbouncepages[.]com`. Because Openphish listed the main domain, all subdomains were assigned a score of 33%. Unbouncepages might not be malicious (at the

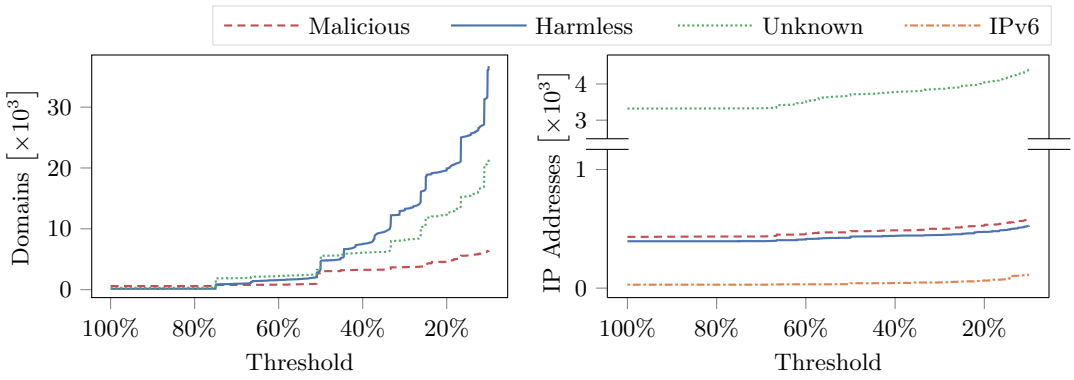
time of writing, it was not listed on the blocklist anymore); although, we think the high threat score for its subdomains is justified assuming the information on the blocklist is correct. The third outliers were 2.7×10^4 IP addresses we observed only in our full IPv4 address scan returning a blocked self-signed certificate. None of our domains resolved to these addresses or were they involved in any other context; hence, the PTP algorithm assigned them a threat score of 100%. The SSLBL contains certificates that botnet C2 servers have used [15], some are default certificates (e. g., embedded in web servers), meaning that the found addresses do not have to be malicious, but they are suspicious. The last outliers were 3.1×10^3 domains with a uniform score of 51%. The domains resolved to an unremarkable IP address, but were redirected to a domain on the Openphish blocklist (i. e., `407979[.]com`). We believe the majority of this group was also generated because they seemed to be made of random 6-character strings with sometimes a prepended "www" (e. g., `www11666x[.]com`). Similar to the above, only 8.0% were from the CT log, and the rest were CZDS domains.

To conclude, the ITEG can help manually analyze the TLS Ecosystem. The PTP algorithm allowed us to quickly find new blocklist-related IP addresses and domains for a more thorough analysis. Moreover, we found several highly suspicious groups of IP addresses and domains.

2.7.5 Comparison with External Threat Intelligence Services

The ITEG allowed us to use a PTP algorithm to propagate a threat score from a set of input nodes listed on blocklists. In the following, we investigate IP addresses and domains with a high score in the graph from January 1, 2024, and check their status with the external threat intelligence services VirusTotal (VT) [49] and Google Safe Browsing (GSB) [76].

VT provides aggregated threat intelligence, and we used it to classify an IP address or domain as *malicious*, *suspicious*, *harmless*, or—in case of a Not Found error—as *unknown*. GSB also provides information about malicious websites; however, their API returned only a *malicious* flag or no data in other cases. Both are unable to check IPv6 addresses or certificates. We combined both sources for our evaluation and flagged an entry as *malicious* if either service identified it as *malicious* or *suspicious*. If present, we used the VT *harmless* class and flagged the rest as *unknown*. However, the VT API was rate-limited, and we could check only a small set of nodes; hence, we removed the first three large clusters we had already identified in the previous Table 2.12 as suspicious from this analysis.



(a) Domains

(b) IP Addresses

Figure 2.11: Cumulative number of domains and IP addresses with a threat score above the threshold, found via PTP, and classified according to the VT and GSB labels (*cf.*, Sosnowski et al. [1]).

Figure 2.11 shows the cumulative number of domains and IP address with threat scores above the shown threshold grouped by the results from VT and GSB. We only analyzed nodes with a score above 10% because the number of affected nodes increased exponentially below this threshold. VT and GSB had no data about several domains and most IP addresses. The figure shows that we found an increasing amount of domains the lower we set the threshold. Most domains were flagged as *harmless*; however, we found an increasing number known as *malicious*. A large portion of the identified domains were unknown to VT and GSB. A different picture is revealed for IP addresses. By far, the largest portion of found IP addresses had a threat score of 100%, mainly due to servers returning a blocked certificate from the SSLBL. We found only a few additional addresses by lowering the threshold; among them were IPv6 addresses we could not check via either service.

For this analysis, we included the smallest cluster from Table 2.12 to evaluate it with VT and GSB. As a reminder, we identified 3.1×10^3 likely automatically generated domains (they had a very similar naming schema) that were redirected to a blocked phishing domain. These domains cause the small sharp increment at the 51% threshold in Fig. 2.11. Interestingly, VT and GSB labeled only 53% of the identified domains *malicious*, 24% *harmless*, and 23% were unknown. Considering all of the indicators, we believe the entire cluster should be classified as *malicious*. It is unclear if VT and GSB did not add these domains due to a lack of visibility or if they were never operationalized. Nevertheless, this highlights our work's importance in identifying gaps in threat detection approaches. However, our approach is only intended as an indicator, and we will discuss false positives in Section 2.7.7.

To conclude, the ITEG modeling and a PTP analysis can be used to propagate thread scores. Many of the newly found domains and IP addresses were already labeled as malicious *by other sources for threat intelligence, highlighting that our approach can identify threat-relevant subsets of the Internet. Moreover, we saw indicators that our work could help fill gaps in the knowledge base of threat intelligence services. However, our findings should not be directly treated as* malicious*, but as a starting point for a more thorough investigation.*

2.7.6 Predicting Blocklist Additions Over Time

One goal of the ITEG modeling was to develop an approach that can find entities on the Internet before becoming known to be malicious. This section analyzes multiple ITEGs over time and whether nodes for which we calculated a high score appeared later on the blocklist. We identified nodes with a "high score" with optimized thresholds, similar to the Threshold-based Classification from Section 2.4. However, instead of scores derived from fingerprints, we used the score calculated with PTP.

We analyzed the 13 monthly measurements from Section 2.7.4. For each month, we created an ITEG and performed the PTP algorithm with the blocklists from Table 2.11 as input. We defined a *new appearance* as an IP address or domain that was not input for the PTP algorithm and which appeared on the input blocklist only after the ITEG measurements were completed and before Feb. 10, 2024. For example, the ITEG from May 1 would use the blocklists downloaded from April 1 to May 7 (because the last TLS scan finished on May 7) as PTP input and the blocklists from May 8, 2023, to Feb. 9, 2024, for the evaluation in this section. Although we identified certificates with high threat scores, none appeared later on the SSLBL. For this reason, we focus on domains and IP addresses in this section.

Optimizing Classification Thresholds

To investigate whether nodes with a high score appeared later on the respective blocklist, we first have to identify the set of relevant nodes. A straightforward approach considers all nodes with a score above a certain threshold. For this analysis, we wanted to use the best-performing thresholds. Hence, we calculated an Appearance Rate for domains and IP addresses for each threshold above 1% across all 13 ITEGs to choose the best option. We define the *Appearance Rate* as the portion of relevant nodes that later appeared on the blocklist. For example, if ten IP addresses scored above 50% and two appeared later on the blocklist, then the IP address Appearance Rate for a threshold of 50% would be 20%. Figure 2.12 illustrates the calculated rates. Then, we selected the best-performing thresholds of 51.3% and 18.0% to identify domains and IP address, respectively.

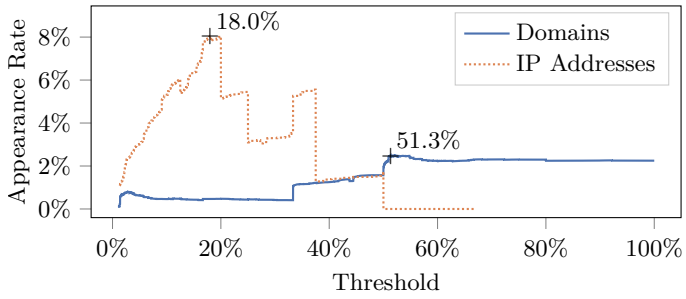


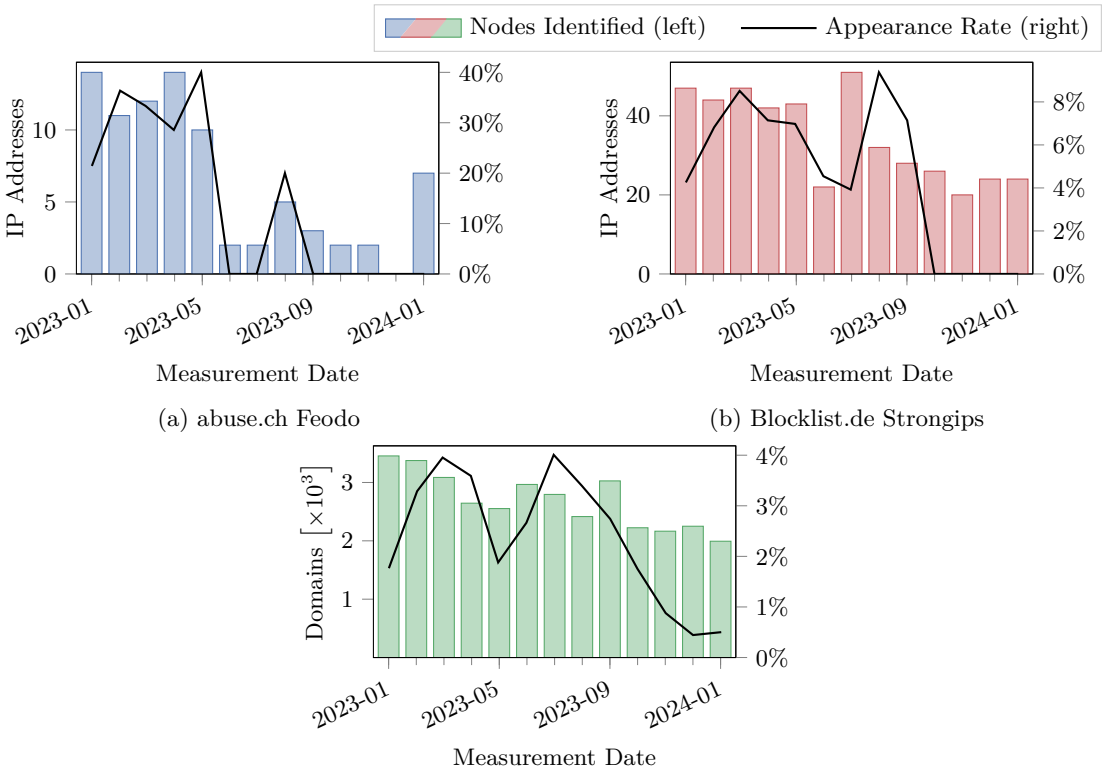
Figure 2.12: Rate of nodes with a score above the depicted threshold that appeared later on the blocklist. Marked are the thresholds providing the maximal rate (*cf.*, Sosnowski et al. [1]).

Detecting Blocklist Additions

Figure 2.13 shows the number of nodes we identified for each of the 13 ITEGs and the percentage of these nodes that appeared later on the respective blocklist. The subfigures reveal that the performance of the overall approach highly depends on the input list and the time of the measurements. Sometimes, it worked very well, and other times, not at all. For example, our approach revealed ten new IP addresses when using the Feodo Tracker as input on April 1, 2023, where four of them appeared later on the blocklist. On the other hand, on December 1, 2023, we identified no new address; either our active measurements missed a relevant piece of data, or there was nothing to find. The Blocklist.de Strongips list was larger, and we identified more IP addresses, although the Appearance Rate was generally lower. The Openphish list contained many more blocked entries, and our approach revealed a few thousand additional domains. Although the Appearance Rate was only between 0.4% and 4.0%, this still means we found 557 domains before they appeared later on the list. Some identified nodes overlapped on consecutive scans; hence, the number of distinct addresses was 5 using Feodo and 6 with Strongips as input. 82% of the IP addresses appearing later on a blocklist returned the same certificate when we identified them and when we scanned them again after they appeared on the blocklist, indicating that the deployment was the same the whole time. The other 18% were unresponsive.

Time Until Blocklist Addition

We can see that the rate of nodes appearing on the blocklists has decreased in the four most recent months. Several nodes we identified with a high score might be added only after writing this paper. Therefore, we analyzed the time until an entry appeared on a blocklist in Fig. 2.14. On average, it took three months for an entry to appear, although it took much longer in some cases (350 days).



(a) abuse.ch Feodo

(b) Blocklist.de Strongips

(c) Openphish

Figure 2.13: IP addresses and domains we identified through their high threat score over time and the rate of them appearing later on the respective blocklist (*cf.*, Sosnowski et al. [1]).

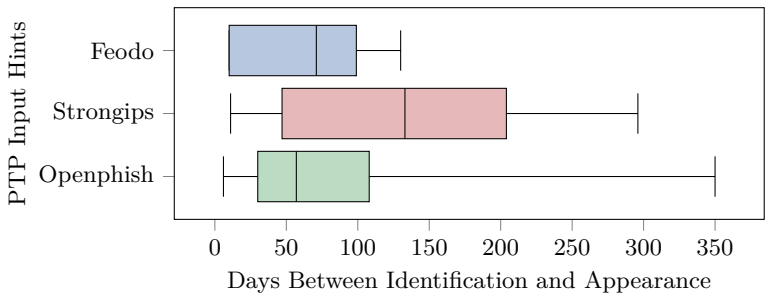


Figure 2.14: Time passed between our identification of an IP address or domain and its inclusion into the blocklist (*cf.*, Sosnowski et al. [1]).

An example for a domain we identified before it was listed by Openphish was D1[1]. During our scans, we could reveal its connection to the domain D2[2] already listed on Openphish. We found the connection because both D1 and D2 resolved to the same IP address and had the same valid Let's Encrypt certificate. Two days after we finished our scans, D1 was listed by Openphish. It is possible that the domain was operationalized only later or that the activity remained unnoticed for several days. Interestingly, D3[3] was also included in the same Let's Encrypt certificate and we assigned it a high threat score, but it was never added to the blocklist at the time of writing. The domain may have never been used, but we believe the threat score is justified due to its suspicious connection to an already blocked domain. The Openphish list only includes entries for which phishing was reported. Hence, this example shows how a low appearance rate is not necessary a problem of the blocklist nor our approach, but a result of different methodologies. Nevertheless, we had false positives, which we will discuss in Section 2.7.7.

In conclusion, the ITEG modeling and PTP allows using existing blocklist data to identify additional IP addresses and domains before they appear on the respective blocklist. While the rate of nodes appearing is generally low, they are present, emphasizing that our approach enables creating valuable cyber-threat intelligence that can serve as a starting point for further investigation. Depending on the input and measurement period, up to 40 % of the identified nodes showed up later on the blocklist.

2.7.7 Discussion

The ITEG modeling and PTP analysis creates new opportunities to analyze the TLS Ecosystem and the Internet. We want to discuss some aspects in the following paragraphs.

Using Graph Libraries and Graph Databases

We propose a graph-based model of the TLS Ecosystem. However, this does not mean the data has to be stored in a graph database. For example, we implemented our message passing in Section 2.7.3 using only a series of SQL joins and aggregations on Apache Spark [22] Dataframes because it allowed us to optimize the algorithm better. Conversely, graph databases can sometimes improve performance because they were

[1] `bluewishlists[.]shop`

[2] `usps[.]speed-mypkg[.]shop`

[3] `usps[.]logistic-info[.]shop`

optimized for typical graph problems; e. g., Abedrabbo et al. [11] observed a "significant degregation of performance" at depths 4 and 5 when comparing graph traversals using a relational and graph database. Additionally, graph-based libraries and databases can provide a convenient interface to interact and visualize data, e. g., we used the Graph Frames [82] library to compute the node degrees in Fig. 2.10 and used Neo4J [117] to visualize and manually inspect our findings.

Optimization of PTP on the ITEG

As we developed our methodology, we used top lists as input due to their smaller size, allowing for faster development. However, we soon realized that our methodology heavily relied on collecting critical pieces of information in a larger puzzle, which was impossible with a small input. This means that the completeness of the graph is very crucial to the efficacy of our approach, and the more data available, the higher the chances of discovering something interesting. Nevertheless, compared to all nodes, only a few blocked nodes exist. Since the PTP algorithm is based on locality, only nodes close to blocked ones can achieve a high score. Thus, we optimized the implementation significantly by considering only nodes that received a score greater than zero, enabling us to run each iteration in minutes instead of hours. We ran the algorithm separately for each blocklist, which enabled us to retrace how concrete scores were created. However, merging the input blocklists into a single list, could save processing resources. Nonetheless, we discovered in Section 2.7.6 that the blocklists behaved differently, and blocklist-type specific thresholds provided the best detection results. Hence, knowing the type of blocklist helps interpreting the results. Future work may involve exploring more advanced models combining numerous blocklists to provide better threat intelligence.

IPv6 Addresses and Blocklists

We could not find IPv6 addresses on our blocklists, and VT and GSB did not support them either. However, we found IPv6 addresses related to the blocked entries; e. g., 29 IPv6 addresses returned a blocked certificate in the scans from January 1, 2024. An approach similar to this work could be a starting point to find malicious IPv6 addresses so that blocklists include them in the future.

Examples for False Positives

Our methodology allowed us to find suspicious IP addresses and domains. However, we noticed also false positives. One cause was that the Openphish blocklist contained Uniform Resource Locators (URLs), but we only modeled domains. Most of the time this was acceptable; however, in some cases this introduced an error. For example, a URL on `sites.google.com` was blocked, but of course not the whole domain should be

treated malicious. This affected smaller websites that redirected their domain to a google site URL and we falsely propagated a threat score to their domains. A similar case was the `bit.ly` domain (a URL shortener) that we falsely labeled as blocked because some URLs were blocked. This affected domains redirecting to `bit.ly`; however, it was an uncommon behavior and several of the domains we found this way were actually known as *malicious* by VT, but most of them as *harmless*. We could have prevented both cases by using a blocklist that considers domains only. Other cases for false positives were subdomains configured insecurely with one of the default certificates blocked by the SSLBL. These subdomains likely hosted services intended for internal or experimental use only (with names like `internal`, `ftp`, `webdisk`, etc.). Due to the score propagation this resulted in a threat score on the other (correctly configured) domains as well.

Valid Certificates in the ITEG

A core aspect of our TLS Ecosystem is that it allows for a decentralized assessment of trust issued by CAs through digital signatures. CAs check whether the private key owner also owns the subject names listed in the certificates. In the context of HTTPS the names are usually domains (rarely, IP addresses). However, we decided not to treat valid certificates (having a valid signature during scanning) differently in the graph and the PTP analysis:

vii) it makes modeling, parsing, and analyzing more straightforward and independent of root stores;

vii) we focus on malicious activity where mostly self-signed certificates are used (e.g., all observed SSLBL certificates were self-signed); and

vii) the graph already contains an intuition of validity as triangles between domains, IP addresses and certificates.

Unless we were subject to a MITM attack, a CA like Let's Encrypt would issue a certificate for all requested domains that resolve to the IP address of the requesting server (basically, creating a triangle) via the ACME [31] protocol. Hence, to a certain extent, the validity of certificates is embedded in the ITEG edges; thus, it is considered in the PTP analysis. However, future work focusing on benign services might model validity differently, as discussed in the next paragraph.

Alternative Graph Models

We designed the ITEG model with the use case of a C2 server detection in mind. Although the general methodology could be the same, different use cases might require alternative graph models. We propose two changes that could be beneficial for solving

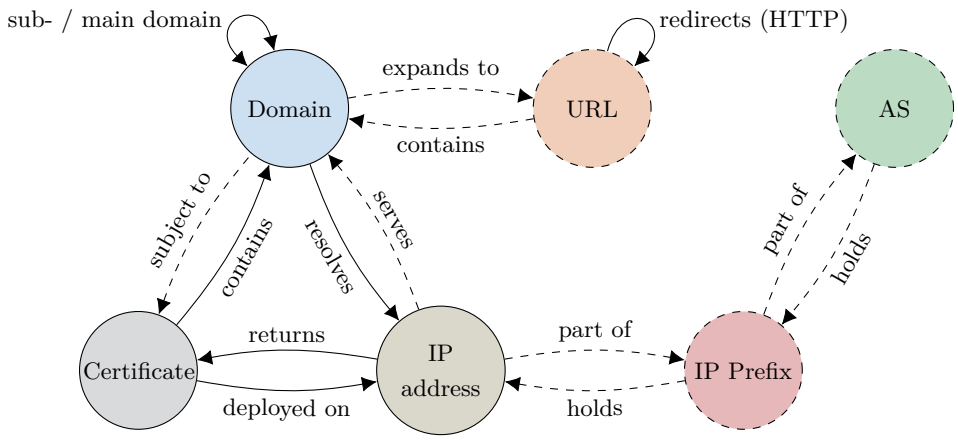


Figure 2.15: Alternative ITEG Schema Enhanced with the Additional Dashed Nodes and Edges.

research questions that investigate the more benign part of the Internet. We illustrate these changes in Fig. 2.15. The first change would be to model certificate validity and whether a server presented the correct certificate and signature during scanning with additional edges. For valid certificates, the issuing CA confirmed that the owner of the private key has control over a domain or is legitimate to server content for this domain. Hence, an additional *subject to* edge can be modeled for all domains embedded in a valid certificate to the certificate itself. Moreover, a *serves* edge can be drawn from IP addresses to a domain if the server presented the correct certificate and handshake signature for the domain requested by our scanner. Another change we suggest is to include URLs in the model, this change could solve some of the false positives we detected due to blocklists containing URLs instead of domains, allowing for more direct modeling of threat indicators. Moreover, this would allow for more direct modeling of the HTTP redirects directly between URLs. We propose connecting URLs with a *contains* and *expands to* relation. Each URL contains a domain name and domains can be expanded to a default URL using an empty path; e. g., `example.org` expands to `https://example.org/`. The last change we discuss is to include Border Gateway Protocol (BGP) prefixes and ASs in the model. Both changes are described with the *part of* and *holds* relations in Fig. 2.15. On a global perspective, routing is based on BGP prefixes and two IP addresses in the same prefix can indicate a common infrastructure to a certain extent. Similarly, IP addresses in IP prefixes that belong to the same AS indicates a commonality. However, a model should only include such relations if they provide a benefit for the use case. We decided against including AS information in our graph because we found that common routing infrastructure is not a good indicator for maliciousness. Even if a single server in a data center hosts a C2 server, the rest of

the data center could be benign. However, if a lot of malicious servers accumulate in a single IP prefix, this could be a valuable threat indicator.

Detection Mitigation

This work relies on malicious actors using the TLS Ecosystem in ways that leak information about their deployments. Such leaks can arise out of necessity (e. g., when a single IP address serves multiple services), limited resources and prioritization on other aspects, or human errors. A malicious actor may mitigate detection by isolating or obfuscating their deployments. The former can be achieved if every malicious activity has its own IP address, domain and certificate. If an address appears on a blocklist and fallback addresses are used, new domains and certificates must also be used to prevent information leakage. However, isolating deployments can be difficult, and the risk of human error remains. It is possible to obfuscate deployments from our PTP algorithm, e. g., by including an unusual number of domains in a certificate. Although the algorithm would not assign a high score in such cases, it would be an outlier in the ITEG, detectible by other means.

Limitations

The data collected with our Internet scans tried to be as comprehensive as possible. However, we only scanned open ports 443 and used a single vantage point. Hence, we missed parts of the TLS Ecosystem (e. g., mail), and our results might be biased due to DNS load-balancing and location [166]. However, it is sufficient to show the potential of this work and provide valuable insights

2.7.8 Conclusion

This work investigates modeling Internet-wide active measurements as a single property graph and applying a PTP algorithm to find new, potentially malicious servers. To evaluate the methodology, we conducted a one-year-long measurement study of 13 monthly Internet-wide DNS and TLS scans and used the collected data to create respective ITEGs. The latest measurement found four highly suspicious clusters among the nodes with high threat scores. Two of the clusters were formed likely due to automatic domain generation. External threat intelligence services were used to confirm a high rate of maliciousness in the rest of the newly found servers. With the help of optimized thresholds, we identified 557 domains and 11 IP addresses throughout the last year before they were known to be malicious. Depending on the input blocklist and the measurement period, up to 40% of the detected nodes appeared later on the respective blocklist used for the PTP input. We do not think nodes with a high score should be directly considered as malicious; however, our approach narrows down the

millions of possible domains and IP addresses, which can be a starting point for a more thorough investigation.

The results were obtained by modeling Internet-wide DNS and TLS measurements as a Labeled Property Graph. Existing blocklists were input for a PTP algorithm that propagated a threat score among IP addresses, domains, and certificates. Our proposed model explains and simplifies the massive data collectible from the TLS Ecosystem. The proposed application of PTP can help to find previously unknown threats on the Internet and provide valuable threat intelligence.

This work proposes a versatile graph model to analyze the TLS Ecosystem and a PTP analysis to help security researchers focus on suspicious subsets of the Internet when searching for unknown threats. In the future, our graph model could be extended using additional data (e. g., routing information) or combined with alternative graph algorithms to answer new security and non-security related questions.

2.8 Incorporating Certificate Revocations in Internet-wide Studies

The TLS Ecosystem has developed due to the need for secure communication over the Internet. A crucial element in achieving such communication is a global X.509 PKI that can authenticate and establish trust in a communication partner. Certificates ensure that the party we are interacting with is who they claim to be through cryptographic signatures, thus preventing man-in-the-middle attacks. These X.509 certificates play a vital role in TLS and are used to authenticate almost any website accessible over the Internet. However, the irrevocable nature of signatures and the complexities involved in distributing revocation information present significant challenges. Currently, there are two main approaches: CRLs and the OCSP CRLs contain a list of all revoked certificates from a single CA, while OCSP can be used to actively request the revocation status of an individual or a few certificates. CAs embed support for either method in each certificate. However, both approaches have performance, reliability, or privacy issues [51].

When modeling the certificate validity of TLS-based systems on a large scale, revocations should also be considered. Additionally, some revocation reason codes can indicate the misuse of a certificate and be a valuable source of threat intelligence for security-related classification use cases. However, collecting a vast amount of revocation data can be challenging. Monitoring the status of a large pool of certificates via OCSP is time-consuming and error-prone. CRLs are better for obtaining an Internet-wide view,

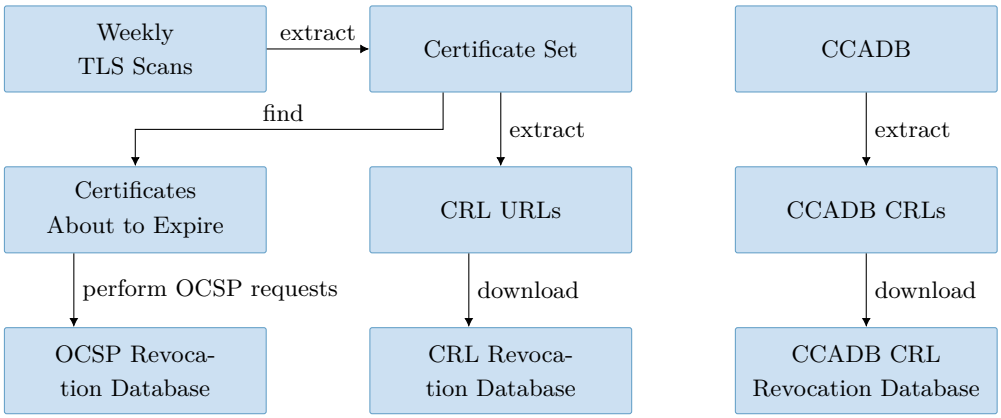


Figure 2.16: Data collection pipeline used to collect Internet-wide certificate revocations (*cf.*, Sosnowski et al. [4]).

as only a single HTTP request provides the necessary information for each CA. However, the majority of certificates do not have CRL endpoints embedded (*cf.*, Section 2.8.2).

As of October 1, 2022, new root store policies from Mozilla [110] and Apple [23] have announced that CAs must publish CRLs in the CCADB [109] for every intermediate certificate that the respective root stores should trust. These new CRLs contain the revoked certificates issued by the intermediate, effectively forcing CAs to provide CRLs if they have not already done so. This policy shift enables a new approach to acquiring a comprehensive view of certificate revocations. Suddenly, a single repository contains references to CRLs covering all relevant TLS Ecosystem revocations.

Collecting an Internet-wide view of certificate revocations is challenging. However, the CRLs published in the CCADB promise significant simplification in collecting a global view on revocations. Creating a single dataset of all revocations is valuable because it allows better modeling of the TLS Ecosystem and also allows for developing and maintaining novel approaches for modeling and distributing revocation information, such as CRLite [95]. In the following section, we compare the three methods for collecting revocation information.

2.8.1 A Pipeline Collecting Internet-wide Revocation Information

Tracking certificate revocations on the Internet can be challenging due to its decentralized nature. No central repository exists, and multiple methods were developed to distribute this information. Therefore, we implemented our own data collection pipeline to create a dataset that is as comprehensive as possible within the possibilities we have.

Our data collection pipeline, outlined in Fig. 2.16, is primarily based on a thorough collection of X.509 certificates for three reasons:

i) OCSP information can only be requested for a known certificate;

i) CRLs do not contain the actual certificates, only the serial numbers, which limits further studies; and

i) we wanted to focus our analyses on certificates actively used on the Internet.

We leveraged the weekly Internet-wide TLS measurements described in Section 2.5.2 to extract and generate a single dataset of X.509 certificates. Within this set, we identified all certificates nearing expiration and requested their revocation status from the included OCSP endpoint. Limiting our checks to expiring certificates allowed us to minimize the necessary OCSP requests, reduce the load on the OCSP responders, and collect the revocation data in a reasonable amount of time. To avoid disrupting their operation, we rate-limited our OCSP requests to $100\,\mathrm{req.\,min^{-1}}$ per CA, which we confirmed with Let's Encrypt, the CA we had to query the most. In addition to OCSP checks, we extracted all CRL distribution points from our certificate set and downloaded each CRL daily. These endpoints listed in the CCADB can be downloaded independently from the certificates. Therefore, we created a separate script to access the CCADB, extract the CRLs, and download each CRL daily.

Once our revocation databases were created, we matched the entries to specific certificates. This step was necessary because we wanted to analyze the revocations in the context of certificates and all three revocations distribution mechanisms only disclose serial numbers, which are unique only among a single CA. Matching OCSP responses to certificates was straightforward, as we needed the certificate to issue the request. Similarly, we matched CRLs to certificates based on the CRL distribution point included in the respective certificate and checked if the certificate's serial number was listed on the CRL. Matching CCADB CRLs was more complex, requiring us to find the entry for the issuing certificate listed in the CCADB and validate the signatures of potential issuing certificates by matching the Authority Key Identifier to the Subject Key Identifier. We then checked whether one of the issuing certificate's CCADB CRLs contained the certificate's serial number in question. Both identifiers are X.509 extensions indicating a suitable issuing certificate [38]; however, only checking the signature can guarantee the relation.

In conclusion, our certificate revocation collection pipeline enabled us to build a global database of revocations based on a comprehensive set of X.509 certificates. Although the

Table 2.13: Collected X.509 certificates from our weekly Internet measurements from March 1, 2022, to January 4, 2024, broken down into various categories (*cf.*, Sosnowski et al. [4]).

	Certificates	Fraction of Parent
Total (with expiry date)	1.20×10^9	
⌞ Leafs	1.19×10^9	98.1%
⌞ Valid	1.12×10^9	94.5%
⌞ with OCSP endpoint	1.12×10^9	100.0%
⌞ with CRL(s)	241.51×10^6	21.5%
⌞ with CCADB CRL(s)	1.12×10^9	100.0%
Valid Leafs with Revocation Information Obtained		
⌞ via OCSP	674.20×10^6	60.0%
⌞ via CRL	240.76×10^6	21.4%
⌞ via CCADB CRLs	1.12×10^9	100.0%

CCADB CRLs provided a convenient way to collect data independent of actual certificates, matching the collected revocations to individual certificates was more complex.

2.8.2 Comparing Certificate Revocation Sources

Based on the methodology outlined in the previous section, we gathered a large dataset of X.509 certificates used in the TLS Ecosystem and their revocations. This dataset allowed us to compare the effectiveness of different methods used by CAs to announce revocations.

Table 2.13 provides an overview of the certificates collected between March 1, 2022, and January 4, 2024. We gathered over 1.1 billion valid leaf certificates during this period. In the table, *Fraction of Parent* indicates the percentage in relation to the parent category; for example, 94.5% of all leaf certificates were found to be valid. A certificate was considered valid if our TLS scanner verified its validity at least once during the scan. Valid certificates have to contain an expiration date and we excluded any certificates without it because we partitioned our dataset based on certificate expiry.

While all the certificates we examined supported OCSP, only 22% offered a CRL endpoint. We were able to match all valid leaf certificates to one or multiple CCADB CRLs. Starting from July 21, 2022, we began downloading CRLs and performing OCSP requests daily based on the certificates from our TLS scans collected in the previous four months. This was inspired by Let's Encrypt's policy of restricting their certificates to a 90-day validity period [10]. We started collecting the CCADB CRLs on October 16, 2022.

Table 2.14: Observed revoked valid leaf certificates and the provided reasons (*cf.*, Sosnowski et al. [4]).

Revocation Reason	OCSP	CRLs	CCADB
None	61.91%	9.90%	34.38%
Cessation Of Operation	23.85%	43.45%	30.76%
Superseded	7.52%	40.35%	30.00%
Affiliation Changed	4.64%	5.13%	3.52%
Key Compromise	1.75%	1.13%	1.30%
Unspecified	0.31%	0.00%	0.00%
Privilege Withdrawn	0.03%	0.04%	0.04%
Certificate Hold	0.00%	0.00%	
Total	1.52×10^6	3.10×10^6	4.48×10^6

Despite our efforts, the data indicates that we could not successfully collect a revocation status for every certificate and method. We performed OCSP requests twice, once for certificates expiring in 30 days and again if they expire the next day. Our OCSP requester failed twice during the measurement period, and we could not collect the revocation data for some of the certificates before they expired. In some rare cases, the download of a CRL failed, or our library could not parse a response. As a result, we did not obtain the revocation data for all certificates via OCSP and certificate CRLs.

In conclusion, we analyzed a large dataset of certificate revocations collected from weekly Internet-wide measurements over almost two years. The analysis revealed that all certificates supported OCSP, while only 22% had CRLs embedded. However, we were able to match a CCADB CRL to all certificates. Our most successful method of collecting revocations was using the CCADB CRLs.

Collected Revocation Data

We could identify 4.5 million certificates revoked at some point during our study, which accounts for 0.37% of the total observed certificates. Table 2.14 displays the total number of revoked valid leaf certificates we identified with the respective revocation mechanism after the policy change on October 1, 2022. It is worth noting that we only performed OCSP requests for about 60% of the certificates.

We also investigated revocation reasons. CAs can provide reason codes for their revocations as defined in [38]. However, most revocations were announced without a reason code. According to the Mozilla root store policy [110], CAs should use the reasons *Key Compromise*, *Privilege Withdrawn*, *Cessation of Operation*, *Affiliation Changed*, and *Superseded*. The first two reasons may be considered more critical as they indicate potential misuse of a certificate, while the latter three reasons are seen as less harmful as they describe circumstances in the normal life cycle of certificates, such as discontinued

Table 2.15: Observed top issuers identified from the Issuer name and their revocations (*cf.*, Sosnowski et al. [4]).

Organization	Revocations			Certificates	
	OCSP	CRLs	CCADB	Total	with CRLs
Let's Encrypt	567.0×10^3		1.1×10^6	796.4×10^6	
Google Trust [...] LLC	232.9×10^3	429.5×10^3	444.2×10^3	93.5×10^6	100%
cPanel, Inc.	132	202	205	58.0×10^6	100%
Sectigo Limited	39.0×10^3	5.1×10^3	96.7×10^3	54.8×10^6	1%
GoDaddy.com, Inc.	490.5×10^3	2.3×10^6	2.3×10^6	36.1×10^6	100%
Cloudflare, Inc.				30.2×10^6	100%
DigiCert Inc	65.7×10^3	202.0×10^3	278.7×10^3	22.0×10^6	19%
ZeroSSL	30.5×10^3		47.7×10^3	11.9×10^6	
Amazon	3	13	13	8.3×10^6	100%
Microsoft [...]	337	517	292	2.3×10^6	100%

websites, changing subject names, or certificates being replaced by updated versions. Most revocations with a reason fell into the latter three categories. We only observed a single certificate with the reason code of *Certificate Hold* (the only temporary revocation method) via OCSP and the CRLs; however, not over the CCADB CRLs. The issuer was "NETLOCK Kft.", a minor issuer from which we collected only 1.45×10^3 certificates. It should be noted that the provided reasons may not be accurate as they are defined solely by the respective CA.

In summary, most certificates are revoked without providing a reason or for harmless reasons that are part of the normal life cycle of certificates. Nonetheless, 1% of the observed revocations indicate misuse and could be a potential source of threat intelligence in the future. Differences between the three revocation sources may be attributed to different issuing organization practices, as discussed in the next section.

Top Certificate-Issuing Organizations

Major organizations dominate the certificate ecosystem. We determined these organizations through the Issuer field embedded in the certificates, and the top 10 are listed in Table 2.15.

It is important to note that the names provided may not necessarily correspond to the actual CA issuing the certificate, as discussed by [97]: CA certificates are valuable, and private keys are sometimes passed on to other organizations. In these cases, the issuer's name might contain outdated information because a certificate cannot be updated after receiving its signature. Additionally, some CAs use white-labeled intermediate certificates to issue certificates for a client. For example, Cloudflare may appear as the organization, but the actual CA controlling the private key of this certificate is DigiCert.

We noticed no revocation for Cloudflare and only a few from Amazon after the policy change.

To conclude, our results showed that the revocation behavior depends highly on the organization. This is especially visible in the support for CRLs; some organizations support them only for a portion of issued certificates, and others not at all, explaining the low CRL coverage in general.

2.8.3 Conclusion

Section 2.8 examines three methods for distributing X.508 certificate revocations and their suitability for creating a comprehensive Internet-wide view of revocations: the OCSP, CRLs embedded in certificates, and the CCADB CRLs. To conduct this analysis, we propose a data collection pipeline that takes a set of certificates obtained through active Internet-wide measurements as input and regularly collects revocations via one of the three methods.

Our findings indicate that the CCADB CRLs provided the most extensive view of global certificate revocations, covering 100% of the 1.1×10^9 valid leaf certificates collected over almost two years and revealing 44% more revocations. In contrast, conventional CRLs only covered 22% of valid leaf certificates, and the OCSP was too costly to maintain an up-to-date database.

We investigated and confirmed the effectiveness of a new possibility for collecting and analyzing certificate revocations: the CRLs published in the CCADB. Initiated by root store policy changes from Mozilla [110] and Apple [23], CAs must provide these CAs for all intermediate certificates capable of issuing new certificates as of October 1, 2022. The ability to rapidly collect a full set of relevant certificate revocations on the Internet allows for a better understanding, modeling, and monitoring of the certificate ecosystem. A comprehensive revocation dataset can enhance measurement studies modeling certificate validity and contributes to developing and maintaining novel approaches for distributing revocation information.

2.9 Related Work

In this section, we discuss various research works that studied the TLS Ecosystem, used TLS metadata for security-related detection purposes, and employed graph models to analyze the TLS Ecosystem.

Several research works have conducted Internet-wide active measurements to enhance our understanding of the TLS Ecosystem. For instance, Amann et al. [20], Durumeric

et al. [62], Holz et al. [80], and Kotzias et al. [92] have performed such research. Durumeric et al. [61] analyzed Internet-wide IPv4 address space scans with a darknet and identified 10.8 million scans originating from more than 1.7 million hosts in January 2014, highlighting the widespread use of Internet scans. Moreover, VanderSloot et al. [163] discovered that IPv4 address space scans only captured one-third of their certificates, indicating the need for a combination of CT logs and SNI scans to obtain a comprehensive view of the TLS Ecosystem.

Numerous related works have demonstrated how mining the extensive data available in the TLS Ecosystem can unveil hidden relationships among modeled entities. For example, Roberts and Levin [132] revealed that publicly available certificates can expose information about company relationships and domain structures. The authors showed how domain names embedded inside certificates can reveal undisclosed business relationships. Even invalid certificates hold data-mining potential, Chung et al. [50] showed how the behavior of a few type of end-user devices and their invalid certificates can be used to uniquely track individual devices.

2.9.1 Active TLS Fingerprinting

As introduced in Section 2.6, Active TLS Fingerprinting uses metadata from the TLS handshakes to uncover undisclosed information. Several works [17, 18, 21, 83, 104] leveraged these metadata using passive approaches. In contrast, this thesis promotes active measurements, which allow interaction with any responsive server on a large scale without being restricted by the increased encryption that hinders passive approaches. Furthermore, it enables obtaining a comprehensive data set from a single vantage point.

A related approach to EFACTLS is JARM, developed by Althouse et al. [19]. JARM can be used to fingerprint and detect malicious C2 servers, and it was successfully applied by Papadogiannaki and Ioannidis [124]. Theofanous et al. [160] extended JARM to develop their own fingerprinting approach, which they describe as being inspired by JARM, EFACTLS, and DissecTLS. They use a hybrid fingerprinting method that sends the ten fixed CHs from JARM and then exhaustively removes cipher suites until the server stops responding. Additionally, they include features of the server certificate in the fingerprinting process. In contrast to this thesis, they used machine-learning techniques to classify C2 servers instead of the threshold-based classification. Their results for identifying C2 servers show a higher recall than we achieved with EFACTLS and DissecTLS but a lower precision.

2.9.2 Graph-based Analyses of the TLS Ecosystem

Graphs can provide an intuitive way to model real-world data, particularly when dealing with network data sources such as the Internet (*cf.*, Section 2.7.2). For instance, Fontugne's [69] work emphasizes the potential of graph modeling in combining multiple Internet data sources to create new knowledge. The *Internet Yellow Pages* project, described by the author, gathers and combines various information sources to form a graph of Internet resources, including AS information, IP prefixes, and domains.

Security-related Internet-wide graph modeling has been successfully applied to higher layers based on the web graph derived from the connections between web pages through hyperlinks. The Web Graph served as the basis for Google's PageRank [39], which globally ranks websites. Gyöngyi et al. [78] proposed *TrustRank* as a means of identifying spam by propagating a trust score from known benign web pages across the Web Graph. Similarly, Najafi et al. [112] introduced *MalRank*, which propagates a maliciousness score between connected nodes based on known CTI observed in proxy and DNS logs.

Graph modeling is a common technique to model passively collected network data, including TLS metadata. Carter et al. [44] developed PTP and used the score propagation technique on a bipartite graph between domains and IP addresses observed in web proxy logs to detect malicious domains based on blocklists. Mandlík et al. [99] described a machine-learning approach based on neural networks and graph inference to detect malicious domains modeling different network entities. Kazato et al. [89] utilized Graph Convolutional Networks (GCNs) on a graph based on IP addresses, domains, and URLs to detect malicious domains, enriching their graph model with DNS resolution and Registration Data Access Protocol (RDAP) information.

Several related works used network graph models and classic community detection algorithms to derive new knowledge. For example, Cangialosi et al. [42] modeled a bipartite graph between domains and email addresses obtained with WHOIS to find clusters of domains corresponding to distinct organizations. Blondel et al. [37] demonstrated the general applicability of community detection algorithms to a vast graph of 118 million nodes and one billion edges derived from a large telecommunication network and the Web Graph. Zhuang and Chang [174] modeled mutual contacts observed between hosts in a network as a graph to find communities of Peer-to-Peer (P2P) botnets.

Similarly to Section 2.7, Simeonovski et al. [142] used DNS, IP addresses, and hosting information to derive a property graph. They employed taint-style propagation techniques to understand the impact of dependencies between Internet services and their providers on attacks. They propagated scores only in explicit directions based on their domain knowledge.

2.10 Reproducibility

We highly value the creation of reproducible research and the publication of associated data and tools. Hence, we have open open-sourced the EFACTLS and DissecTLS fingerprinting approaches as well as the 10 general-purpose CHs and their generation code as part of the TUM goscanner [73]. Moreover, we open-sourced the scripts necessary to reproduce the graph creation pipeline for the ITEG and provide an already parsed graph created from a scan of the Tranco [96] toplist that can be directly imported and explored in Neo4J. Additionally, we have created websites explaining our measurement pipeline, the necessary script calls, and give details about the published data:

- *Active TLS Fingerprinting: Additional Material* [150] explains the measurement pipeline and provides access to the data used in the analysis of Section 2.6.8. The data is published over mediaTUM [151].
- *DissecTLS: Additional Material* [149] provides explanations and access to the scripts necessary to reproduce the analyses of Section 2.6.7.
- *ITEG: Additional Material* [144] gives access to further details about our results, the scripts necessary to create our ITEG, and an already parsed example graph.

2.11 Key Results

This chapter investigated how actively collected data from TLS-based systems can be modeled to uncover undisclosed security-related information. We examined two general methodologies: Active TLS Fingerprinting and the ITEG. To compare different approaches, we suggested Entropy for measuring the information collected through active measurements and defined a threshold-based classification technique. The main motivation of this chapter was to determine if we can create innovative CTI and demonstrate this with the detection of C2 servers.

We further refined Active TLS Fingerprinting by distinguishing between fixed-probe and exhaustive approaches. We proposed specific approaches for each category: EFACTLS and DissecTLS. Through large-scale active Internet measurements, we demonstrated the effectiveness of these approaches in fingerprinting C2 servers. Furthermore, we showed how our extended feature collection, improved CHs generation, and our exhaustive scanning could enhance existing related approaches.

With the ITEG, we illustrated how Internet-wide graph modeling of the TLS Ecosystem can be combined with threat propagation techniques to indicate malicious behavior. We demonstrated the usefulness of this approach by conducting monthly Internet-wide DNS

and TLS measurements, generating respective graphs, using three existing blocklists as input threat indication, applying a PTP algorithm to spread the indication to related nodes in the graph, and identifying new servers before they appeared on the respective blocklist.

Finally, we explored various techniques for distributing certificate revocation information and their potential to be incorporated into Internet-wide studies. The goal was to create a comprehensive dataset of global certificate revocations that could be utilized as CTI. Our findings suggested that the CRLs from the CCADB provided the most sophisticated basis to acquire a global view of certificate revocations, covering nearly all the revocations we gathered from the CRLs embedded in the certificate and when scanning the OCSP.

2.12 Author's Contributions

This chapter builds upon previous publications, expanding on prior findings and refining the individually published research into a comprehensive contribution. It incorporates joint work with collaborators Johannes Zirngibl, Patrick Sattler, Claas Grohnfeldt, Michele Russo, Daniele Sgandurra, Tim Betzer, Juliane Aulbach, Jonas Lang, and Georg Carle. This collaboration was conducted in the context of the conference publications *Active TLS Stack Fingerprinting: Characterizing TLS Server Deployments at Scale* [6], *DissecTLS: A Scalable Active Scanner for TLS Server Configurations, Capabilities, and TLS Fingerprinting* [5], *Propagating Threat Scores with a TLS Ecosystem Graph Model Derived by Active Measurements* [1], and *An Internet-Wide View on HTTPS Certificate Revocations: Observing the Revival of CRLs via Active TLS Scans* [4], as well as the published journal article *EFACTLS: Effective Active TLS Fingerprinting for Large-Scale Server Deployment Characterization* [8]. For all of these works, the author of this thesis holds first authorship and provided significant contributions to the publications. This includes major contributions to the conception, design, and analyses of the studies and writing of the published manuscript. Parts of the original works that were not so relevant to the research questions of this thesis have been omitted. The following paragraphs offer detailed information about the contributions of the author of this thesis, indicate which publications the specific sections of this chapter are primarily based on, and how the author extended the collaborative work for this thesis.

Sections 2.3 and 2.4 are based on the collaborative work [8]. In this thesis, the author explores the concept of using Entropy for active measurements in greater detail, and broadened the threshold-based classification concept to encompass all C2 detection studies discussed in this chapter.

Section 2.5 combines content from all five publications [1, 4, 5, 6, 8] and summarizes the Internet measurement pipelines used to create the datasets that served as the foundation for the analyses in these publications and this thesis.

One component of these pipelines is the TUM goscanner [73], which the author enhanced by improving the overall scanner architecture and implementing the EFACTLS and DissecTLS fingerprinting approaches The author contributed to designing and maintaining the top- and blocklist measurement pipeline detailed in Section 2.5.1. Aside from the enhancements made to the TUM goscanner, the author holds no significant contribution to the Internet-wide scanning pipeline described in Section 2.5.2.

Section 2.6 combines and expands upon content from three publications [5, 6, 8] that investigate Active TLS Fingerprinting. The first two publications form the foundation of the EFACTLS methodology, with the research initially published as a conference paper [6] and later as an extended journal article [8]. The third publication [5] discusses the DissecTLS methodology. The author contributed significantly to all three publications, participating in the initial conception and design of the research, conducting the presented analyses, and implementating both the EFACTLS and DissecTLS fingerprinting methodologies.

For this thesis, the three publications have been merged into a single section to provide a more comprehensive overview of Active TLS Fingerprinting. Compared to the original publication, a slightly different dataset for Internet measurements was utilized, described in Table 2.4. While the underlying data is the same as that published with [8] under [151], it was filtered differently for servers from which successful fingerprints could be collected using all three approaches: JARM, EFACTLS, and DissecTLS, allowing for direct comparisons. This change of the dataset allowed for recreating the fingerprinting study of C2 servers in Section 2.6.8 to additionally compare the performance of using JARM and DissecTLS fingerprints as classification input. The comparison is shown in the added Fig. 2.5. A new analysis has been added in Fig. 2.4b, which investigates the impact of the Status Request extension on DissecTLS fingerprints.

Section 2.7 is based on the joint publication [1]. The author led the research agenda, made significant contributions to the design of the ITEG model, the development and implementation of the graph processing pipeline, the PTP implementation, and the presented analyses. For this thesis, the author added a discussion on alternative graph models.

Section 2.8 is based on the content of the joint work [4]. Jonas Lang initially designed the revocation pipeline as part of his Bachelor's Thesis and Interdisciplinary Project, supervised by the publication's authors. The author of this thesis led the work, defined

the research goals, guided the research agenda, and contributed to the publication by refining the approach and implementation of the pipeline and conducting the presented analyses.

Finally, Section 2.9 combines and unifies the related work analyses from the five publications [1, 4, 5, 6, 8].

Chapter 3

Modeling TLS Characteristics for Performance Insights

This chapter investigates the performance of TLS-based systems in the context of novel cryptographic developments. The contents of this chapter are based on the published conference contribution [3]. Changes and new contributions compared to the original publication are described in Section 3.8.

3.1 Introduction

To preserve the security of our communication over the Internet, adopting new cryptographic algorithms may be necessary; however, such changes can significantly impact the performance of these communications. The TLS protocol secures the majority of our Internet traffic [94]. Thus, any performance changes to the protocol can significantly affect a substantial portion of Internet traffic and the overall user experience. It is essential to identify the factors that influence performance and assess the extent of their impact before implementing any changes on a broad scale.

This chapter examines methods for measuring and modeling TLS protocol characteristics to analyze performance implications of the negotiated algorithms. These methods are used to understand the performance effects of novel PQC algorithms on TLS-based systems and to identify new opportunities for performance optimization. While some performance implications of changes to the TLS protocol may be evident—such as the volume of data exchanged or the number of Central Processing Unit (CPU) cycles consumed—today's networks are complex, involving numerous interplaying technologies, hardware components, software libraries, and protocols. This complexity makes

it challenging to identify the potential consequences of a protocol change. Even minor adjustments can lead to unforeseen side effects when implemented in real-world environments.

In the remainder of this chapter, we investigate the following research questions:

RQ2.1: How can we model and measure TLS 1.3 handshakes for performance analyses in a local testbed? To analyze the performance implications of different cryptographic algorithms in the TLS protocol, we propose abstracting the handshake into two phases that can be measured without decrypting network traffic. We empirically evaluate these implications through testbed measurements. The testbed allowed us to collect performance data with minimal measurement bias and emulate various network conditions.

RQ2.2: What are the performance implications of post-quantum-safe TLS? It is known that PQC alters the CPU costs associated with the TLS handshake and increases the volume of exchanged data. However, to better understand the implications of these changes when deployed in actual networks, we applied our methodology to compare both pre-existing algorithms like ECDH and RSA with currently relevant post-quantum algorithms like Kyber, Dilithium, SPHINCS$^+$, and Falcon. We found that TLS using PQC is generally fast, but the large key sizes can be a bottleneck. Sometimes, selecting the ideal PQC algorithm involves a trade-off between computational and bandwidth costs.

RQ2.3: Do new optimization opportunities emerge when post-quantum algorithms are deployed in a TLS network stack? Our analyses revealed new optimization possibilities due to the significantly larger key sizes of PQC algorithms exchanged during the TLS handshake compared to current state-of-the-art methods. We found that both the mapping of TLS messages to TCP segments and the initial TCP CWND can be adjusted for improved performance.

3.2 Background

Quantum Computers (QCs) differ significantly from traditional computers and can efficiently solve mathematical problems fundamental to our current cryptographic algorithms. They are no longer seen merely as a concept within computational science and theoretical physics. Substantial research efforts and significant funding from corporations and governments are being directed toward developing practical QCs. Notable advancements include Google's announcement of achieving quantum supremacy [25] and IBM's release of a 433-qubit processor [55].

A practical QC would represent a major milestone in technological evolution, enabling solving computational problems currently intractable for conventional computers. Unfortunately, two of the few practical algorithms that can be efficiently solved on QCs are the mathematical problems forming the basis of today's public-key cryptography: the integer factorization and the discrete logarithm [139]. Thus, most current public-key algorithms, such as DH, ECDH, and the Digital Signature Algorithms (DSAs) RSA and Elliptic Curve DSA (ECDSA), would need to be replaced with alternatives that can withstand the computational power of QCs.

To address the potential threat posed by QCs, the National Institute of Standards and Technology (NIST) launched a competition in 2016 to identify and standardize new algorithms. These algorithms are based on mathematical problems resistant to QCs, commonly known as PQC [134]. By the end of 2022, after three evaluation rounds, NIST announced the early winners Kyber, Dilithium, Falcon, and SPHINCS$^+$ [16]. Three of these algorithms have already reached the standardization stage. Additionally, NIST announced a fourth round, which includes at least the three KA algorithms: HQC, Bike, and Classic McEliece.

Although existing QCs need to incorporate more qubits to crack cryptographic algorithms, the risk of *Store-Now-Decrypt-Later* attacks—where adversaries store encrypted data today to decrypt it later when powerful QCs become available—highlights the necessity for adopting quantum-safe approaches as soon as possible. Consequently, PQC is becoming increasingly relevant, and selecting the best-performing algorithms for practical applications like TLS is an important area of research. Performance is critical, as resources are often limited in processing time and power consumption, and low client latency can affect user experience, directly measurable in lost revenue [32].

3.3 Modeling TLS Handshakes for Performance Analyses

This chapter focuses on TLS 1.3, the most popular method for establishing secure connections between two endpoints (*cf.*, [168]). At least three aspects of the TLS handshake influence overall performance: CPU costs, the volume of transmitted data, and the number of Round Trip Times (RTTs) required to complete the handshake. With TLS 1.3, clients can pre-compute a key share based on the KA they expect the server to select, enabling 1-RTT handshakes. Therefore, our focus is on 1-RTT handshakes.

The CPU costs associated with TLS and the amount of transmitted data primarily depend on the cryptographic algorithms used for the KA and the DSA. TLS 1.3 remains

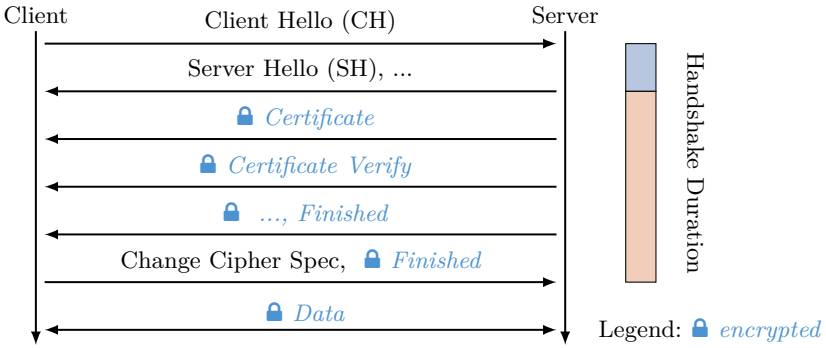


Figure 3.1: 1-RTT TLS 1.3 handshake, including the two phases where latency can be measured without decryption (*cf.*, Sosnowski et al. [3]).

independent of specific KAs or DSAs, as these are merely negotiated parameters of the handshake. When assessing the performance impact of using an alternative algorithm for TLS, one can predict changes in CPU costs and the volume of transmitted data. However, real-world systems are complex, and adjustments to these parameters can have unforeseen side effects. We propose to detect such influences by creating an abstract TLS handshake model, which can be used to collect and analyze performance data from real-world TLS-based systems.

Post-quantum-safe TLS differs from traditional TLS in specifying and selecting post-quantum algorithms during the handshake. The private-public key pair used to generate the server certificate determines the DSA. Fig. 3.1 illustrates a 1-RTT TLS 1.3 handshake, listing the TLS messages relevant to this chapter and highlighting the handshake duration we examined. Clients initiate the handshake with the CH, to which servers respond with the TLS Server Hello (SH). The handshake occurs over a symmetrically encrypted channel. The Certificate message is essential for the client to authenticate the server. The certificates included can employ pst-quantum signatures, which affect their size and computational expenses when verifying the signatures. Next, the Certificate Verify message transmits the handshake signature generated using the server certificate's private key, which PQC also influences. Finally, the server concludes the handshake with a Finished message, which contains a hash of all previous messages to ensure the integrity of the handshake.

Depending on the Maximum Transmission Unit (MTU), which is a network parameter that determines the maximum packet size, the messages from the SH to the server's Finished message can be sent in a single IP packet. The implementation's efficiency in combining these messages varies. Using the certificates received, the client verifies the signature and determines whether the server is trustworthy based on the employed PKI.

Table 3.1: Overview of currently relevant algorithms for TLS 1.3 and their current state of standardization.

Key Agreement (KA)	Digital Signature Algorithm (DSA)
CRYSTALS-Kyber (ML-KEM [114])	CRYSTALS-Dilithium (ML-DSA [113])
Bike	Falcon (ongoing)
HQC	SPHINCS$^+$ (SLH-DSA [115])
State-of-the-Art pre-quantum counterpart:	
ECDH	RSA, ECDSA

To complete the handshake, the client sends a dummy Change Cipher Spec and an Finished message, which again contains a hash over previous messages. In our measurements, both were consistently combined into the same IP packet. A passive observer can measure the first part of the handshake (from CH to SH) and the second part (from SH to the client's Finished), as these contain unencrypted data.

In summary, PQC alters the CPU costs and transmitted data volume of TLS handshakes. However, such changes can have unforeseen side effects in real-world systems. We propose modeling 1-RTT TLS 1.3 handshakes as two consecutive parts to investigate the performance impacts on actual traffic without decrypting the communication.

3.4 Post-Quantum Cryptography and its Influence on TLS

Asymmetric cryptography relies on mathematical problems that can be solved efficiently in one direction while reversing the operation is computationally expensive to the point of being practically impossible for large numbers. Regular computers can efficiently compute a product or exponentiation, but finding the inverse, such as integer factorization or discrete logarithms, is much more demanding. Shor [139] demonstrated that the security of today's asymmetric cryptography is at risk due to QCs being able to solve these problems efficiently.

TLS employs algorithms that are influenced by quantum computing in the initial KA, the handshake signature, and the certificate signatures forming an X.509 PKI. During the handshake, TLS switches to symmetric encryption, which is only partially affected by quantum cryptanalysis. Grover's [77] algorithm theoretically reduces the security level of symmetric encryption using QCs by half.

To understand the impact of QCs on TLS, we briefly give an overview of the current state of PQC. PQC is based on mathematical problems that are considered intractable for QCs. NIST has initiated a multi-round challenge for researchers to submit PQC

Table 3.2: Security levels and requirements according to [154] (*cf.*, Sosnowski et al. [3]).

Level	Comparable difficulty to a ...
1:	key search on a block cipher with a 128-bit key (e. g., AES128)
2:	collision search on a 256-bit hash function (e. g., SHA256)
3:	key search on a block cipher with a 192-bit key (e. g., AES192)
4:	collision search on a 384-bit hash function (e. g., SHA384)
5:	key search on a block cipher with a 256-bit key (e. g., AES256)

candidates or propose attacks. Recently, NIST announced [16] the standardization of four post-quantum finalists: Kyber, Dilithium, Falcon, and SPHINCS$^+$. Additionally, four more algorithms—Bike, Classic McEliece, HQC, and Sike—were promoted to a fourth round, although Sike was broken shortly after the announcement. Furthermore, Classic McEliece has key sizes that exceed the $2^{16} - 1$ bytes required by TLS 1.3, which would require changes to the TLS protocol. As a result, neither KA method is currently relevant for TLS. The current state of NIST standardization and references to the respective standards are summarized in Table 3.1. It is important to note that this field of research is rapidly evolving. Recently, NIST announced [153] 14 additional DSAs that are candidates for standardization, which could soon increase the number of algorithms relevant to TLS.

Each algorithm can be configured to meet various security or performance requirements, with the authors providing several pre-selected configurations for direct use in a TLS handshake based on defined security levels. According to NIST [154], an algorithm fulfills a certain security level if the computational complexity of breaking it via brute force is comparable to specific problems, as summarized in Table 3.2. Proving the security of cryptographic algorithms is challenging; typically, they are "proven over time", meaning they are assumed secure until vulnerabilities are discovered. The longer researchers search for vulnerabilities without finding any, the greater the confidence in the algorithm's security.

This situation creates two challenges for the adoption of the comparatively new PQC:

iv) Unknown vulnerabilities might exist, threatening the security of encrypted communications.

iv) Even if no vulnerabilities exist, persuading people that the new algorithms are secure enough to be adopted can be difficult.

A hybrid approach may mitigate these issues by combining traditional and post-quantum KA methods. This way, an attacker must compromise both methods before being able to

recover the shared secret. In this practice, the two KAs are performed independently, and the final shared secret is obtained by concatenating the individual secrets [155]. Similarly, two DSAs can be combined [123], requiring both to be broken to compromise the PKI. We refer to the resulting combination from either approach as a hybrid. These hybrids claim to be at least as secure as the current state-of-the-art while benefiting from the additional security provided by PQC. However, the complexity of combining algorithms can increase the costs associated with TLS. It is important to note that, similar to individual algorithms, there is no proof that hybrid approaches provide the claimed level of security or do not introduce new attack vectors.

To conclude, PQC impacts TLS by addressing the predicted vulnerabilities in current asymmetric algorithms against quantum computers by introducing new KA and cryptographic signatures algorithms. Currently, NIST is standardizing such algorithms. However, adopting new algorithms that have yet to be proven over time is risky. A hybrid approach combining traditional and post-quantum methods claims to enhance security but could also increase complexity and costs.

3.5 Measuring the Performance Impact of PQC on TLS 1.3

This section explores the performance of PQC in TLS 1.3. We assess how various traditional and post-quantum cryptographic algorithms affect TLS 1.3, covering the following topics:

v) a measurement methodology designed to minimize interference and ensure comparability;

v) a comparison of traditional and post-quantum algorithms used for the KA and DSA in TLS, analyzing end-to-end handshake latency and the volume of data transmitted;

v) insights into how the seemingly independent KAs and DSAs can impact each other; and

v) recommendations on the most suitable algorithms from a performance standpoint.

3.5.1 Measurement Methodology

We measured the performance of PQCs used in TLS through a series of sequential TLS handshakes conducted on a three-node setup in 60-second intervals. The purpose of this setup was to analyze protocol performance without deploying any utilities on the client and server that could interfere with the results.

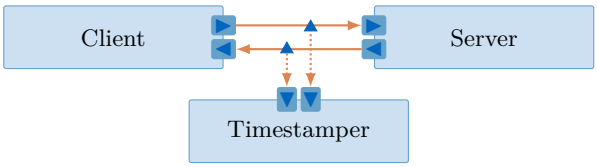


Figure 3.2: Measurement Setup (*cf.*, Sosnowski et al. [3])

In TLS 1.3, KA and DSA are selected independently. However, a TLS handshake requires both, and a combined measurement of concrete KA and DSA pairs is needed. To evaluate them individually, we fixed the KA or DSA to X25519 or RSA_{2048} for most analyses. We examined four general types of KAs: ECDH, HQC, Kyber, and Bike, each available in multiple variants, resulting in 23 KAs. Similarly, we investigated four general DSAs: RSA, Falcon, Dilithium, and SPHINCS$^+$, which resulted in 22 distinct DSAs. Notably, SPHINCS$^+$ offered 36 variants, and we observed latencies that range from a few milliseconds to several seconds. Thus, our analyses focus solely on the fastest SPHINCS$^+$ configuration, the simple Haraka signature optimized for signing speed, and selected subsets of the theoretical $|\text{KA} \times \text{DSA}|$ combinations. Additional SPHINCS$^+$ and KA-DSA combinations are available on our website, described in Section 3.6. Our measurements specifically concentrate on the 1-RTT TLS handshake, and we configured our setup to ensure that the 2-RTT fallback never occurred.

Measurement Setup

Figure 3.2 illustrates our measurement setup, which consists of three nodes: a client and a server, which are separate hosts connected directly via $10\,\text{Gbit}\,\text{s}^{-1}$ fiber links, and a third Timestamper host. The Timestamper tapped into the connection using passive optical fiber taps, allowing us to collect precise hardware timestamps for all packets exchanged between the client and the server with minimal impact on latency and jitter. All nodes utilized identical hardware: Intel Xeon D-1518 CPUs (4 cores, 2.2 GHz), dual-port Intel X552 Network Interface Cards (NICs), and 32 GB of Random-Access Memory (RAM). The client and server operated on Debian Bullseye (Linux kernel v5.10). We used a fork of the OpenSSL [28] library that supported PQC. The TLS handshakes were measured using the integrated OpenSSL client and server. The Timestamper ran Debian Buster (Linux kernel v4.19) and utilized MoonGen [64] to record hardware timestamps. Our measurement and analysis processes were automated via POS [71] to ensure the repeatability of results. We measured handshake performance over 60 seconds and observed between 1 000 and 30 000 handshakes, depending on the complexity of the algorithms under investigation. The reported latencies represent the median values recorded during each measurement period.

Black-Box Measurements

Only the TLS server and client were operational on their respective nodes, while the Timestamper monitored the connection. As outlined in Fig. 3.1, we measured two parts of the handshake: the time the server took to start serializing the SH after the client started serializing the CH and the time the client took to complete the handshake with a Finished message. Our TLS library always sent the Finished message and the unencrypted Change Cipher Spec message in the same packet, enabling us to timestamp these three events accurately without decrypting the traffic. The first part includes the computations related to KA on the server. We noted instances where the server held back the SH until it calculated the handshake signature. In these cases, the first part also depended on the DSA. We analyze these instances in Section 3.5.3. The second part relies on both KA and DSA due to the required key computation and signature verification on the client side. We did not account for any computations related to the CH on the client.

Optimized TLS Message Buffering

Our study identified inconsistent behavior within the OpenSSL library. For specific algorithms, the server occasionally buffered all computed TLS messages and sent them to the client in a batch or sent some messages earlier. This behavior was independent of the MTU and originated from an internal library buffer of 4 096 bytes. By default, the library calculated all messages and forwarded them only to the client after completing the Certificate Verify message. However, whenever a message exceeded the buffer size, its contents—most notably the SH—were flushed to the TCP stack and sent to the client. This behavior affected handshake latency, particularly for costly key decapsulation operations on the client side. Overall performance improved when clients began computing the secret key while servers continued processing the rest of the handshake. To enhance consistency, we modified the OpenSSL library to immediately send the SH and the Certificate message to the client as soon as they were computed. Unless otherwise specified, we used this optimized version of OpenSSL for our measurements and analyses. However, if TLS libraries adopt this behavior, we recommend they do so individually per algorithm, as the benefits of faster processing should outweigh the overhead of sending additional IP packets and the extra context switch required to flush the SH earlier.

3.5.2 Post-Quantum TLS under Ideal Network Conditions

In this analysis, we measured the latency and the amount of data exchanged between the client and the server during a TLS 1.3 handshake. Our black-box approach avoids using on-host analysis tools, resulting in unbiased measurements. The results for the

Table 3.3: Measured handshake latency, number of handshakes, and data usage for the investigated algorithms per NIST security level (*cf.*, Sosnowski et al. [3]).

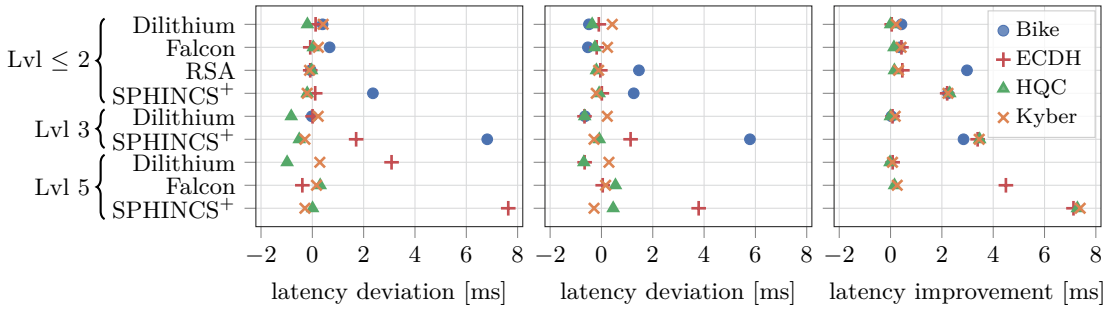
(a) KAs combined with `rsa:2048` as DSA

Lvl	KA	Handshake		Data Sent [B]	
		Latency Median [ms]	# Total	Client	Server
1	X25519	0.25, 1.48	22 315	689	1 455
	bikel1	0.24, 2.79	13 371	2 250	3 048
	hqc128	0.27, 1.48	21 922	2 958	6 060
	kyber512	0.20, 1.78	20 816	1 457	2 191
	kyber90s512	0.19, 1.79	20 545	1 457	2 191
	p256	0.33, 1.50	21 175	722	1 488
	p256_bikel1	0.42, 2.58	13 507	2 315	3 113
	p256_hqc128	0.52, 1.31	20 458	3 023	6 125
	p256_kyber512	0.51, 1.81	17 859	1 522	2 256
3	**bikel3**	0.42, 6.07	6 913	3 844	4 642
	hqc192	0.53, 1.40	19 467	5 387	10 761
	kyber768	0.25, 1.82	19 170	1 893	2 511
	kyber90s768	0.20, 1.78	20 747	1 893	2 511
	p384	3.09, 2.63	7 000	754	1 520
	p384_bikel3	3.23, 9.00	3 693	3 941	4 739
	p384_hqc192	3.39, 3.30	6 118	5 484	10 858
	p384_kyber768	3.17, 2.72	6 736	1 990	2 608
5	**hqc256**	0.72, 1.92	15 258	8 214	16 412
	kyber1024	0.25, 1.78	19 753	2 277	3 043
	kyber90s1024	0.22, 1.78	19 952	2 277	3 043
	p521	6.97, 5.30	3 382	790	1 556
	p521_hqc256	7.52, 9.38	2 397	8 347	16 545
	p521_kyber1024	7.06, 5.41	3 329	2 410	3 176

(b) DSAs combined with `X25519` as KA

Lvl	DSA	Handshake		Data Sent [B]	
		Latency Median [ms]	# Total	Client	Server
	rsa:1024	0.32, 0.68	29 970	689	1 066
	rsa:2048	0.25, 1.48	22 268	689	1 455
1	**falcon512**	0.36, 1.02	24 262	689	2 920
	rsa:3072	0.26, 3.41	12 625	689	1 839
	rsa:4096	0.25, 6.88	7 355	689	2 223
	sphincs128	0.28, 15.02	3 693	1 001	36 153
	p256_falcon512	0.39, 1.35	21 731	689	3 137
	p256_sphincs128	0.28, 15.48	3 545	1 001	36 372
2	**dilithium2**	0.39, 0.84	27 072	689	6 981
	dilithium2_aes	0.39, 0.78	27 527	689	6 981
	p256_dilithium2	0.39, 1.27	22 626	689	7 185
3	**dilithium3**	0.36, 0.94	26 076	741	9 471
	dilithium3_aes	0.38, 0.86	26 998	741	9 471
	sphincs192	0.27, 23.83	2 366	1 105	74 769
	p384_dilithium3	0.31, 3.86	11 639	741	9 823
	p384_sphincs192	0.27, 28.75	2 000	1 105	75 087
5	**dilithium5**	0.36, 1.10	24 133	793	12 923
	dilithium5_aes	0.36, 0.98	25 250	793	12 923
	falcon1024	0.38, 1.89	18 405	689	5 097
	sphincs256	0.27, 49.52	1 180	1 209	104 253
	p521_dilithium5	0.29, 7.55	6 763	793	13 330
	p521_falcon1024	0.35, 8.72	5 979	689	5 572
	p521_sphincs256	0.27, 60.78	966	1 209	104 679

Legend: **Post-Quantum** Hybrid CH → SH SH → Client Finished



(a) Default OpenSSL behavior (b) With Optimization (c) Resulting Improvement

Figure 3.3: Comparison of the handshake latency for KA-DSA combinations on the same level. The OpenSSL behavior of assembling TLS messages (default and optimized) influences the latency. A positive deviation means it was faster-than-predicted (*cf.*, Sosnowski et al. [3]).

KAs and DSAs we investigated are detailed in Tables 3.3a and 3.3b. Algorithms marked with an underscore (_) indicate hybrids.

The reported latency represents the median time required to complete the handshake, divided into two parts (see Fig. 3.1). Additionally, we present the volume of data the client or server exchanged in each handshake. Besides Bike and SPHINCS$^+$, PQC challenges the speed of ECDH or RSA at level one, while outperforming traditional algorithms on higher levels. We observed no significant overhead in using hybrid algorithms, moreover, on higher security levels the traditional algorithms became the bottleneck. Handshakes conducted with Dilithium, regardless of the security level, were faster than RSA_2048, which we consider sub-level one because its security resembles a 112-bit key length [30] rather than the 128-bit key length required for level one. We also observed minor performance improvements using variants of Kyber and Dilithium (indicated as `90s` or `aes`), which utilize AES instead of SHAKE for masking and sampling. Even the fastest variant of SPHINCS$^+$ listed in Table 3.3b demonstrated poor performance, with handshake latency and data usage—due to large signatures—up to 20 times higher than the others.

In summary, our findings indicate that post-quantum algorithms, except for Bike and SPHINCS$^+$, challenge traditional algorithms at security level one while significantly outperforming them at higher security levels. Furthermore, we noted almost no overhead when using hybrid algorithms.

3.5.3 Analysing the Independence of KA / SA

KAs and DSAs are selected independently in TLS 1.3, and we assumed they influence handshakes independently. However, this was only sometimes the case based on the following experiment.

If we assume that the KA and DSA independently impact the TLS handshake and the latency caused by each algorithm is deterministic, we should be able to predict the handshake latency of arbitrary combinations. Otherwise, they somehow influenced each other. Given two KAs $k1, k2$ and two DSAs $s1, s2$, we can measure the TLS handshake latency $M : KA \times SA \to \mathbb{R}$. If both are independent, then

$$M(k1, s1) + M(k2, s2) = M(k1, s2) + M(k2, s1). \tag{3.1}$$

Hence, we can calculate an expected value for arbitrary combinations with our baseline $E : KA \times SA \to \mathbb{R}$ with

$$E(k, s) = M(k, \mathrm{RSA}_{2048}) + M(\mathrm{X25519}, s) - M(\mathrm{X25519}, \mathrm{RSA}_{2048}). \tag{3.2}$$

We combined all KAs and DSAs (except hybrids and only RSA_{3072}) on their respective NIST security level and calculated the deviation $E(k, s) - M(k, s)$ for both default and optimized OpenSSL version—which sends SHs immediately after computation—in Figs. 3.3a and 3.3b, respectively.

While we expected small deviations, the results reveal that specific combinations significantly impacted each other. Dilithium-HQC combinations were slightly slower than expected, and Bike and ECDHs were faster, depending on the DSA. Though we cannot rule out additional factors, we found this was due to the TLS message buffering of OpenSSL (*cf.*, Section 3.5.1). It introduces a dependency between KA-DSA and causes deviations from the calculated expected latency. We found two explanations: first, algorithm combinations were slightly faster if their combined key sizes caused an early push of the SH, although one algorithm alone would not trigger the push. The outliers for the optimized OpenSSL version are smaller because this push was consistent (see Fig. 3.3b). Second, CPU-intensive KAs (i. e., Bike or ECDH above level one) benefited from the parallel processing enabled by the early SH, especially when the DSA was computationally heavy (e. g., for SPHINCS$^+$ and RSA_{3072}). In the case of Bike and RSA, the effect is only visible for the optimized version because they did not cause a push in the default version.

To conclude, the OpenSSL buffering messages introduces a dependency between KA-DSA; however, it is low compared to the whole handshake. We showed that the hand-

shake latency can be reduced when the SH is pushed to the client as soon as it is computed. Aligning this behavior had an interesting effect: most handshakes were faster, as shown in Fig. 3.3c. Combinations with SPHINCS$^+$ improved up to 7.4 ms*; indicating optimization potential libraries could utilize for some algorithms.*

3.5.4 Post-Quantum TLS in Constrained Environments

So far, we analyzed TLS and PQC performance under optimal network conditions characterized by a loss-free, low-latency connection with high bandwidth. To gain insights into realistic network conditions, we extend our analysis to examine constrained environments, which may experience packet loss, delays, and limited bandwidth, as encountered by embedded devices or in wireless communications.

Using the Linux netem tool [156], we emulated various scenarios including a 10% packet loss, 1 second RTT, a bandwidth of 1 Mbit s^{-1}, and two scenarios reflecting real-world settings: LTE-M over a distance of 15 km and a 5G setup. Our findings from these experiments are detailed in Tables 3.4a and 3.4b.

We made several notable observations:

iv) A high packet loss alone slowed down the handshakes, but it had the most negligible impact compared to the other parameters. However, the effect of packet loss is amplified by the large keys and in conjunction with a high delay reveals that it can cause the handshake to take several RTTs, which occured in the more realistic scenarios.

iv) Low bandwidth generally resulted in slower handshakes with an increased effect on algorithms that transferred more data (e. g., HQC, Dilithium, and SPHINCS$^+$);

iv) The latency of TLS increased approximately linearly with added delay.

iv) The two realistic scenarios primarily depended on the RTT, although the high packet loss occasionally caused the median latency to increase by several RTTs. It is important to note that we measured consecutive handshakes over a 60-second period, which led to significantly fewer samples in high-delay situations.

The 1-second RTT scenario revealed an interesting finding: All SPHINCS$^+$, HQC, and Dilithium algorithms at level 5 required multiple RTTs to complete the handshake. This effect was due to the TLS messages from the server exceeding the TCP CWND. Each TLS handshake took place after the TCP handshake, which meant the CWND was still at the configured minimum during the Slow Start phase (usually $10 \times MSS$). As shown in Table 3.3, PQC requires a more extensive data volume, resulting in occasional handshakes taking two to four RTTs. Additionally, all PQC algorithms involve large

Table 3.4: Median handshakes latency measured for different network scenarios. Loss, RTT, and bandwidth were emulated with netem [156]. The width of the bars are scaled depending on the Maximum latency per table and scenario (*cf.*, Sosnowski et al. [3]).

(a) KAs combined with `rsa:2048` as DSA

Lvl	DSA	No Emulation	High Loss (10%))	Low Bandwidth	High Delay	LTE-M	5G
1	X25519	1.8	2.0	14.1	1002.2	214.1	46.2
	bikel1	3.1	6.2	27.0	1006.6	227.0	50.6
	hqc128	1.8	2.0	51.3	1002.2	251.3	46.3
	kyber512	2.0	2.3	20.0	1002.5	220.0	46.4
	kyber90s512	2.0	2.2	20.0	1002.4	220.0	46.4
	p256	1.9	2.0	14.4	1002.2	314.7	46.2
	p256_bikel1	3.0	6.9	27.5	1007.0	227.5	51.1
	p256_hqc128	1.9	2.6	51.8	1002.8	251.9	46.8
	p256_kyber512	2.3	2.5	20.5	1002.6	220.6	46.7
3	**bikel3**	6.5	16.3	45.1	1016.5	508.8	60.5
	hqc192	1.9	3.0	89.2	1003.0	568.3	47.2
	kyber768	2.1	2.3	22.6	1002.4	222.6	46.5
	kyber90s768	2.0	2.3	22.6	1002.5	222.6	46.5
	p384	5.7	8.2	17.2	1008.4	245.2	52.3
	p384_bikel3	12.3	23.3	52.5	1023.4	258.0	67.4
	p384_hqc192	6.7	9.8	92.9	1010.0	507.4	54.0
	p384_kyber768	5.9	8.3	26.0	1008.5	226.1	52.5
5	**hqc256**	2.6	4.4	139.8	2004.8	706.9	92.8
	kyber1024	2.1	2.3	26.9	1002.5	226.9	46.5
	kyber90s1024	2.0	2.3	26.9	1002.5	367.6	46.5
	p521	12.3	17.4	21.2	1017.5	221.5	61.6
	p521_hqc256	16.9	20.8	158.1	2020.9	498.7	108.9
	p521_kyber1024	12.5	17.7	34.5	1017.8	234.6	61.8

(b) DSAs combined with `X25519` as KA

Lvl	DSA	No Emulation	High Loss (10%))	Low Bandwidth	High Delay	LTE-M	5G
	rsa:1024	1.0	1.4	11.0	1001.6	411.2	45.6
	rsa:2048	1.8	2.0	14.1	1002.2	214.1	46.2
1	**falcon512**	1.4	1.8	25.9	1001.9	226.0	46.0
	p256_falcon512	1.7	2.6	28.0	1002.8	228.1	46.8
	p256_sphincs128	15.8	16.0	297.8	2006.6	908.9	136.1
	rsa:3072	3.7	4.0	17.2	1004.2	475.4	48.2
	rsa:4096	7.2	7.5	20.4	1007.7	220.4	51.9
	sphincs128	15.3	15.5	294.7	2005.8	906.1	94.0
2	**dilithium2**	1.2	1.7	58.7	1001.9	258.7	45.8
	dilithium2_aes	1.2	1.6	58.7	1001.7	258.7	45.7
	p256_dilithium2	1.6	2.4	60.6	1002.7	440.7	46.6
3	**dilithium3**	1.3	1.9	78.9	1002.0	475.8	46.1
	dilithium3_aes	1.2	1.7	78.8	1001.8	278.8	45.9
	p384_dilithium3	4.2	9.0	85.3	1009.2	285.3	53.1
	p384_sphincs192	29.1	29.8	613.7	3008.6	1722.6	184.8
	sphincs192	24.1	24.6	607.7	3005.1	1473.5	181.6
5	**dilithium5**	1.5	2.3	106.5	2001.7	504.7	89.8
	dilithium5_aes	1.4	1.9	106.4	2001.6	480.4	89.7
	falcon1024	2.2	2.5	43.7	1002.8	445.8	46.7
	p521_dilithium5	7.9	18.0	117.0	2009.5	510.5	97.5
	p521_falcon1024	9.1	18.1	55.4	1018.1	255.5	62.1
	p521_sphincs256	61.1	61.6	857.3	4013.4	1999.6	233.5
	sphincs256	49.8	50.2	846.0	4005.6	1885.1	226.0

Legend: **Post-Quantum** Full handshake (CH → Client Finished) [1] 1 Mbit s^{-1} [2] 1 s RTT

[3] 10% loss, 200 ms RTT, and 1 Mbit s^{-1} bandwidth. Parameters measured by [57] over 15 km.

[4] 4% loss, 44 ms RTT, and 880 Mbit s^{-1}. Parameters measured by [172].

key sizes, and combining post-quantum KA and DSA results in an additive effect on the handshake size, potentially further increasing the number of RTTs required for the handshake.

In conclusion, the larger PQC keys significantly impact environments with reduced bandwidth, long RTTs, and packet loss. Due to slightly shorter key lengths, Kyber and Falcon outperformed other post-quantum algorithms in low-bandwidth settings. We anticipate that the initial CWND will become a critical tuning factor for TLS servers to maintain the ability to perform 1-RTT handshakes, particularly when combining post-quantum KAs and DSAs, both of which increase the handshake size.

3.5.5 Related Work

The research interest in post-quantum TLS is extensive, with numerous publications exploring various aspects, particularly performance.

In 2020, Paquin et al. [125] evaluated different experimental setups under both emulated and real-world network conditions. They concluded that handshake completion times primarily depend on the speed of cryptographic operations in optimal conditions and on data size under poor network conditions. In the same year, Sikeridis et al. [141] published their findings on post-quantum signatures and authentication, assessing cryptographic performance through Internet-wide measurements. Their research highlighted that key and certificate sizes significantly affect handshake time. Sikeridis et al. [140] further investigated the performance impact of PQC on TLS and Secure Shell Protocol (SSH) handshakes. They identified the initial TCP CWND as a bottleneck, arguing that the increased delays measured for post-quantum algorithms could be mitigated by increasing the initial CWND, allowing for competitive performance compared to traditional algorithms. We only evaluated PQC with TLS 1.3 over TCP, but Kempf et al. [90] showed that fast post-quantum TLS is also possible with QUIC. They also compared the performance of different QUIC libraries and evaluated the overhead of the symmetric encryption by implementing a NOOP cipher.

Paul et al. [127] explored the use of mixed algorithms within an X.509 PKI and reported promising results by combining the Extended Merkle Signature Scheme (XMSS) for root certificates with Dilithium for the rest of the certificate chain.

Focusing on low-power embedded systems, Bürstinghaus-Steinbach et al. [41] evaluated Kyber and SPHINCS$^+$ against their traditional counterparts, ECDH and ECDSA. They found that certain algorithms, particularly Kyber, outperformed traditional variants, suggesting that the deployment of PQC is feasible with minimal overhead, especially with potential improvements from hardware accelerators. Continuing this focus on

Table 3.5: Summary of related works on performance measurement of post-quantum TLS.

Investigated aspects	Sikeridis et al. [141]	Paul et al. [127]	Bürstinghaus-Steinbach et al. [41]	Paquin et al. [125]	Marchsreiter and Sepúlveda [101]	Tzinos et al. [161]	Zheng et al. [173]	Sikeridis et al. [140]	Kempf et al. [90]	this work
PQC KAs	✗	✗	○[1]	○[1]	○[1]	○[1]	○[1]	○[1]	✓	✓
PQC DSAs	✓	✓	○[2]	○[3]	✓	✗	○[3,4]	○[2,3]	✓	✓
Hybrid algorithms	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓
Constrained networks	✗	✗	✗	○[5]	✗	○[5]	✗	✗	✗	✓
Two handshake phases	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
KA and DSA dependency	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

Legend: ✓ Yes ✗ No ○ Partly [1]Kyber [2]SPHINCS$^+$ [3]Dilithium [4]Falcon [5]Only RTT and loss

embedded devices, Marchsreiter and Sepúlveda [101] analyzed hybrid post-quantum algorithms. They noted that while PQC can be more efficient, handshake times may still be negatively affected by the choice of hybrid algorithms, with performance on constrained devices potentially suffering due to large key sizes and slower connections.

In 2022, Tzinos et al. [161] evaluated various KA mechanisms in a local network environment, focusing on the speed of cryptographic operations. Similarly, Zheng et al. [173] focused on the cryptographic implementation of ML-KEM/Kyber and proposed an optimized version that achieved a higher rate of handshakes per second. They concluded that their findings align with the choices made by NIST, which suggests Kyber as a new standard. Measurements from Cloudflare [169] demonstrated the competitive performance of post-quantum algorithms. Additionally, Cloudflare has enabled hybrid KAs across nearly all domains they serve [54], but they currently only support Kyber-768 and ML-KEM, excluding DSAs.

Recently, concerns [34] have been raised regarding the security analysis of Kyber, particularly suggesting that the security level of Kyber-512 may fall below NIST's level 1 requirements. This implies that Kyber may need longer keys and increased computational times to achieve comparable security. If this is the case, analyses that conclude Kyber outperforms other post-quantum algorithms at the same security level should be reevaluated, and alternative algorithms may be preferable.

We present a summary of our findings regarding related works in Table 3.5. PQC continuously evolves, so studies can quickly become outdated. Most of the listed related works evaluated algorithms that are no longer relevant or did not cover new algorithms

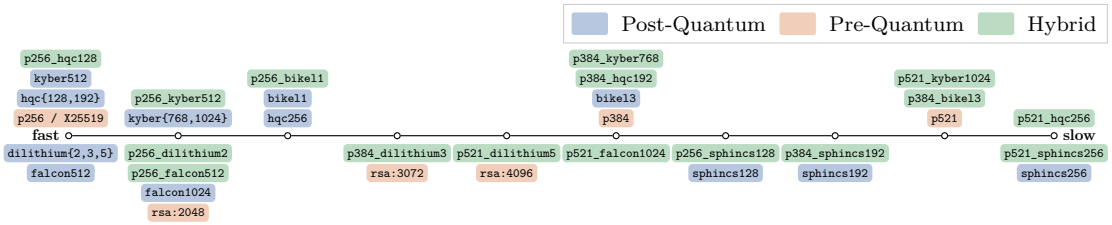


Figure 3.4: KAs (top) and DSAs (bottom) ranked depending on the logarithmic handshake latency we measured. Top and Bottom is ranked separately with the algorithms on the left being the fastest (*cf.*, Sosnowski et al. [3]).

that have emerged. NIST recently announced [153] an additional 14 DSAs under consideration for standardization, highlighting the rapidly evolving field and the potential for future research. Like us, more recent studies have included hybrid algorithms in their assessments. Nonetheless, few studies have examined constrained network scenarios, and most of the listed research relied on measurements using cloud servers. This absence of emulation may explain why only one study [140] identified the TCP CWND as a potential bottleneck. Furthermore, our research stands out because we modeled and measured the TLS handshake in two distinct phases and analyzed the potential interdependencies between KA and DSA. The modeling and analysis led us to discover the suboptimal handling of TLS messages in our library. Additionally, we employed a 3-node testbed measurement setup, which allowed us to obtain accurate measurements and minimize any potential biases in the results.

In summary, numerous studies have focused on PQC from a performance standpoint. PQC is a rapidly evolving field, and due to the ongoing development of new algorithms, it continues to be a significant area of research. Although the methodologies used for measurement and the algorithms explored differ among these studies, their key findings are largely consistent with our results.

3.5.6 Discussion

In this section, we discuss results and insights that can be generated with our methodology and derived from our analyses.

While we do not evaluate the security of post-quantum algorithms, we present precise measurements demonstrating their end-to-end performance. Our tables contain complex data, so we aimed to refine and simplify our results to put them into context, thereby making them more useful as recommendations. Figure 3.4 presents a ranking of PQC algorithms. We took the overall latency data from Tables 3.3a and 3.3b, applied a logarithmic transformation, linearly scaled the results to fit within the interval $[0, 10]$,

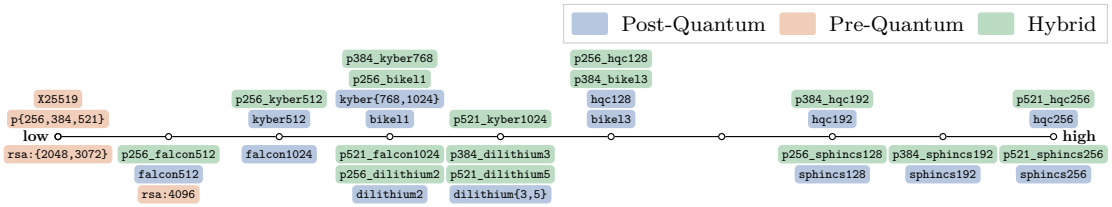


Figure 3.5: KAs (top) and DSAs (bottom) ranked depending on the data transmission volume. Top and Bottom is ranked separately with the algorithms on the left causing the least traffic volume (*cf.*, Sosnowski et al. [148]).

and rounded them to whole numbers. This process led to a ranking of the algorithms, with higher ranks displayed on the left.

Our results indicate that PQC challenges our state-of-the-art solutions, and some DSAs are even faster. Moreover, the hybrid algorithms showed no performance drawbacks at NIST level one. However, at higher levels, the post-quantum algorithms became faster than the hybrids, which are limited by the speed of the pre-quantum algorithm. It is important to note that latency should not always be the primary criterion; the computational capacity of clients can sometimes be more critical, especially for constrained devices.

Additionally, the transmitted data volume can become a more significant consideration in scenarios involving low bandwidth or high latency. Kyber and Falcon outperformed other post-quantum algorithms in our measurements with limited bandwidth due to their slightly smaller key sizes. Hence, Fig. 3.5 introduces a second dimension by ranking algorithms based on the amount of data transferred. The data transfer volume for PQC is significantly higher than that of pre-quantum algorithms, with considerable differences among post-quantum algorithms. In the case of the hybrid algorithm, the data transfer sizes are primarily influenced by the post-quantum component, with only a minimal impact from the pre-quantum algorithms.

Latency and data transmission volume are valuable metrics for ranking different algorithms for TLS. Nevertheless, there is another factor that we did not evaluate in this section but should be considered before deploying new cryptographic algorithms: the potential asymmetry between client and server. If the computational costs are significantly higher on the server side, an attacker could exploit this to overload the server, as discussed in [56]. Our findings in [3] demonstrated that the CPU costs can be up to six times higher on the server. Furthermore, significant size differences between client requests and server responses, such as a small request triggering a large response, could be used to overwhelm targets with spoofed requests, as indicated in [116]. Table 3.3b

shows that server replies can be up to 96 times larger than the initial client requests. For comparison, QUIC mandates a maximum amplification factor of 3 to mitigate such attacks [116].

In conclusion, we recommend transitioning to hybrid algorithms, as they offer the following advantages:

vi) Protection against threats from actors who might capture network traffic to extract sensitive data, particularly when powerful QCs become available.

vi) Confidence in their security because the traditional part of the algorithms has been proven over time.

vi) No significant performance drawbacks.

The performance disadvantage of SPHINCS$^+$ is evident, described as a "clear drawback" [26] by its authors. However, its underlying mathematical properties are well understood within the cryptography community. Therefore, SPHINCS$^+$ may be viewed as more secure from today's perspective. We did not evaluate the DSAs involved in CRLs or OCSP checks and leave the analysis of the impact of PQC on certificate revocation mechanism for future work.

3.5.7 Conclusion

Section 3.5 compares the performance of traditional and post-quantum algorithms for TLS. All measurements were conducted using a consistent methodology on identical hardware to ensure comparable results. We focused on measuring the initial handshake of TLS, which relies on asymmetric cryptography expected to be compromised by QCs in the future. Through black-box measurements, we examined various DSAs and KAs currently relevant in the NIST standardization competition to obtain precise handshake latencies and assess data transmission volumes. We also emulated network conditions like high loss, limited bandwidth, and increased delay to analyze performance in constrained environments.

Our findings allow for comparing the performance of all currently relevant traditional and post-quantum algorithms for TLS. We discovered that handshakes using PQC can exceed the initial TCP CWND on servers, leading to additional RTTs per handshake. Thus, the initial CWND could become an important tuning factor for post-quantum TLS. Our results indicate that HQC and Kyber perform on par with the current state-of-the-art algorithms, while Dilithium and Falcon are even faster. We observed no performance drawbacks when using hybrid algorithms; moreover, PQC outperformed any algorithm currently in use at NIST security levels three to five. Only SPHINCS$^+$

and Bike showed poor performance. PQC can be slower than the current state-of-the-art in low-bandwidth environments due to the increased data transmission volumes. Kyber and Falcon outperformed other post-quantum algorithms in these cases because of their comparatively smaller key sizes.

In conclusion, using post-quantum or hybrid algorithms does not result in significant performance drawbacks. Our findings can help optimize TLS libraries and aid in selecting the right algorithm for various applications. We hope that future developments in post-quantum cryptography will take our performance perspective into account so that these algorithms remain practical for TLS implementations. Ideally, they should provide performance improvements over existing state-of-the-art algorithms, encouraging their wider adoption.

3.6 Reproducibility

A central goal of our research is to promote reproducible experiments and publication of associated measurement data. To facilitate this, we created a website [148] explaining our measurements and providing additional analyses for our investigated algorithms. Our experiment and plotting scripts are available on GitHub [147], and we published our measurement data through mediaTUM [143].

For our experiments, we utilized three nodes equipped with optical splitters and hardware timestamping to minimize external effects. While this specific setup may not be broadly accessible, we have made our artifacts available to a wider community by adapting our scripts to run in a containerized 2-node environment, which is also accessible via GitHub [147]. This approach allows researchers to execute similar measurements on different hardware, with the understanding that results may vary due to differences in containers, virtualized networks, software timestamping, and underlying hardware. However, the presented trends and our conclusions should remain reproducible.

Additionally, we provide the raw PCAP files and CPU profiler results recorded on our infrastructure [143] to enable the exact reproduction of the presented evaluations. Our efforts at reproducibility were recognized, and we received the *Artifacts Available*, *Artifacts Evaluated*, and *Results Reproduced* v.1.1 badges. These awards indicate that independent reviewers confirmed the artifacts are publicly available in an archival repository, they completed an audit, and that the main results from our paper can be reproduced [24].

3.7 Key Results

This chapter examined how to measure and analyze the performance of TLS handshakes. We applied this methodology to compare traditional and post-quantum cryptographic algorithms in a local testbed under various emulated network conditions. We defined a model of the TLS handshake that breaks it down into two phases, allowing for black-box performance measurements without the need to decrypt TLS traffic. Our results indicate that, under ideal network conditions, post-quantum algorithms—except for Bike and SPHINCS$^+$—performed comparably to traditional algorithms at NIST security level one and outperformed them at higher security levels. Furthermore, we did not find significant disadvantages using hybrid algorithms, mostly identifying the pre-quantum algorithms as the performance bottleneck. Choosing an appropriate post-quantum algorithm can involve considering the trade-off between CPU costs and bandwidth. This trade-off is particularly evident in our emulated constrained network scenarios. Here, the large cryptographic keys used in post-quantum TLS handshakes impact the performance more than just increasing the serialization time. If the handshake messages exceed the initial TCP CWND, the handshake can take multiple RTTs to complete. Moreover, we observed that the large keys exacerbate the effect of packet loss on a connection. Additionally, we identified new performance tuning factors related to the larger key sizes: increasing the initial TCP CWND and immidiatly sending TLS messages after computation, the latter was not the default behavior of the investigated OpenSSL library.

3.8 Author's Contributions

This chapter is based on collaborative work with Florian Wiedner, Eric Hauser, Lion Steger, Dimitrios Schoinianakis, Sebastian Gallenmüller, and Georg Carle in the context of the conference publication *The Performance of Post-Quantum TLS 1.3* [3]. The chapter contains and extends content from the published contribution. The author of this thesis significantly contributed to the conception and design of the study, the analyses, and the published manuscript. The author investigated potential bottlenecks in TLS related to PQC, provided the TLS handshake model, conducted the presented analyses, and implemented the optimized behavior of the TLS library as identified in Section 3.5.3. For this thesis, the content of the original work has been refined to align with the research questions of this thesis. Sections of the original work that did not involve the author's significant contributions or were irrelevant to the research questions have been removed. Information on the NIST PQC standardization process was updated to reflect new developments, resulting in the added Table 3.1. The related work section

was extended and summarized in the added Table 3.5 to better differentiate the current state-of-the-art.

Chapter 4

Extending the Applicability of Web Caching

This chapter investigates extending the applicability of web caching to a broader range of TLS-based systems on the Internet. The content of this chapter is based on the published conference paper [2]. Changes and new contributions compared to the original publication are described in Section 4.9.

4.1 Introduction

Web caching is a known method to achieve improved availability and performance (e. g., [12, 13, 35, 43]), and caching dynamic data like REST API responses presents a known challenge [86]. Both is very important for web applications as an unavailable service or a low client latency directly impacts the user experience and can be measured in lost revenue [32]. Additionally, the achievable throughput defines the scalability and the number of clients a system can support. However, not all types of web communication can utilize caching. The increasing adoption of TLS by web applications, the growing use of CDNs, the separation of network requests for static code and dynamic application data in software development, and the adoption of standardized API formats like REST [67] or JSON:API [88] have changed how web applications are built (*cf.*, Section 4.2). TLS provides the means to ensure the integrity of an end-to-end communication which means application developers have fine-grained control over which parties in the network can influence the communication and, e. g., apply caching. These changes in web development and the fine-grained control which web proxies partake in the communication open up opportunities to enhance web application performance and availability.

If a web application separates requests for static code and dynamic data and utilizes an architecture that supports caching read requests, most of the application traffic can be cached effectively. However, the missing link to achieve a fully responsive application, even during outages of the server, is the caching of requests that change the application state. A new type of web proxy that acts as an information broker for the underlying data rather than working on the level of HTTP requests, could enable this link. CRDTs [138] offer a unique method for building distributed applications and have shown significant progress in recent years (e. g., [91, 131]). In this chapter, we investigate an innovative web caching method for distributing application data in a network of web proxies based on CRDTs, enabling origins to automatically update outdated cached content and proxies to respond directly to write requests. We call the method CWC.

There are two additional challenges to consider: first, any approach that works on a layer on top of HTTP has the risk of impacting the application logic. If this risk is too high or software developers cannot understand or control the behavior, they will not adopt the approach. The second challenge is that the approach can be easily integrated into existing systems and, e. g., REST APIs. Ideally, it should allow developrs to use it only for selected API endpoints that are performance critical. Our proposed solution with CWC is a dynamic protocol upgrade where origin server configures the caching logic applied by a proxy for each API endpoint as it is beeing accessed. The goal of CWC is to enable caching of both read and write API requests while providing a good-enough consitency to make the approach suitable for dynamic API data.

In the rest of this chapter, we investigate the following research questions:

RQ3.1: How can we process requests that change application state on Web proxies? With CWC is an innovative approach for caching dynamic application data and enables web proxies to independently process mutating requests and synchronize the received changes with the origin later in time. We additonally discuss requirements for enabling CWC for REST APIs on three different levels and suggest design considerations for an origin and CDN for its implementation.

RQ3.2: How can we integrate a novel web proxy technology into existing applications? With an on-demand protocol upgrade, CWC can be integrated seamlessly into established applications and existing REST APIs. We reason that with this approach application developers retain full control of their application logic and no data is lost.

RQ3.3: What types of API communication benefit from the proposed web caching approach? The highest benefit is achieved if application developers can express the API

communication logic in simple CRUD operations. Nevertheless, if this is not possible, the push-based semantic of CWC still provides performance and consistency benefits over TTL and invalidation-based approaches. Additionally, we experimentally compared different API caching approaches based on simple forwarding, TTL-based caching, invalidation-based caching, and CWC in a simulated CDN deployment. The results showed CWC can compete with the performance of a TTL-based and the consistency of an invalidation-based approach while being the only option for caching write operations.

4.2 Background

This chapter focuses on client-server applications, where the client provides a User Interface (UI), and the server manages the main application logic. This approach is commonly used in enterprise applications and typically involves three primary layers: *presentation*, *domain* (the actual logic), and *data source* [70]. Usually, the presentation is handled by a client-side UI, the domain by a backend server, and the data source by a database.

In modern applications, the interface from the presentation to the domain layer can be implemented with Web APIs, which are dedicated endpoints for retrieving the application data. This approach has the benefit of allowing the backend server to offer a streamlined API that deals only with the domain logic. The streamlining allows multiple different clients, such as iOS, Android, and Web, to utilize the same API, making development and maintenance more efficient. Developing such APIs for web applications is aided by the increasing popularity of Single-Page Applications (SPAs). Since the emergence of Flash and Java applets in 2000, followed by Asynchronous JavaScript and XML (AJAX) in 2005 and HTML5 in 2008, there has been a shift towards SPAs [68]. In this architectural pattern, the entire client application is loaded upfront into the browser, and user interactions are handled via JavaScript. SPAs offer the portability and cross-platform functionality of traditional web applications with the client-state management and responsiveness of native applications [68].

However, communication with backend servers becomes a bottleneck if users frequently wait for server responses. Ultimately, the physical distance those API calls must traverse imposes a hard limit on application performance. One solution is to bring the server closer to the client through a distributed system. Table 4.1 illustrates different levels of such a system. The HTTP caching feature of CDNs is a lightweight form of distribution, as certain requests can be handled directly by the CDN proxies. Nevertheless, a single backend server (referred to as *origin* in the context of CDNs) remains responsible for the domain logic, which can still be a bottleneck. A fully distributed system runs the

Table 4.1: Improvement of the system latency in the context of the distribution level of the application.

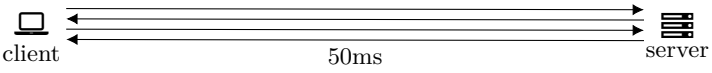
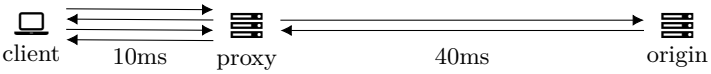
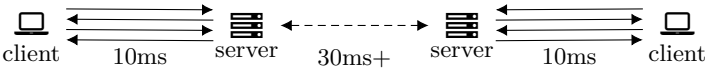
Distribution level	Example deployment. Arrows are exchanged messages with the respective one-way-delay	Latency
none		200ms
simple (caching)		120ms
full		40ms

Table 4.2: Comparing state-of-the-art caching strategies for Web APIs using a CDN.

Caching Strategy	Suitable for Dynamic API Data	Caches Reads	Caches Writes
None	✔	✖	✖
TTL	✖	✔	✖
Invalidation	✔	✔	✖
CRDT Web Caching	✔	✔	✔

application close to the clients, enabling fast responses. However, such a system is complex to develop, and operating multiple servers worldwide instead of a single origin can be costly.

Web application developers have currently two main options to improve the REST APIs using CDN caches: define a TTL or actively invalidate content. However, TTL-based caching is unsuited for the dynamic data exchanged via REST APIs, and neither can speed up write requests. Table 4.2 illustrates this relation.

Content Delivery Networks

CDNs are services a web application provider can utilize to improve their own service. CDNs operate proxies at numerous locations worldwide. When users access a service that uses a CDN, they connect to one of the CDN proxies closest to their location. The proxy then forwards the request to the origin server of the web application provider. This redirection through the CDN allows for various optimizations in communication, such as serving cached responses, preventing Distributed Denial of Service (DDoS) attacks, modifying the content, and redirecting it through more efficient paths on the Internet [121]. CDNs like Cloudflare, Fastly, and Akamai implement HTTP caching, although some extend the functionality.

HTTP Caching

This caching method was first defined in 1996 for HTTP/1.0 [118] and has been continuously improved with the latest update in 2022 [66]. It is designed for the end user's browser and intermediate shared caches, such as those provided by CDN proxies. The main idea is that a server can mark any HTTP responses with caching instructions using HTTP headers. The server can specify whether the response should not be cached, should be treated as private for each user, or can be served to any user requesting the same URL. Servers can specify a maximum TTL for how long a cache can serve the content before it becomes stale. However, no standardized way exists to invalidate a stored response before the TTL expires. Despite this, HTTP caching provides a limited method for caches to verify stored responses with the origin: a cache can inform the origin about a stored response (with the *ETag* or *If-Modified-Since* header) allowing the origin to send a new response or inform the cache that the stored version is valid. Nevertheless, this can only save parts of the retransmission, while the costs of contacting the origin remain.

Invalidation-based Caching

Modern CDNs go beyond the standard HTTP caching functionality by extending the "expiration-based caching model and additionally expose (non-standardized) interfaces for asynchronous cache invalidation" [74]. This includes tagging responses with a custom key and then invalidate all responses associated with that key. Fastly, for example, claims to purge all global caches in less than 200 ms using a bimodal multicast algorithm to propagate the purges [152]. Similarly, Cloudlflare claims to be able to invalidate cached content in less than 150 ms [93] Moreover, more advanced caching strategies can be built upon the invalidation concept. For instance, Stellate offers a GraphQL API caching service that operates on the Fastly network by automatically generating the appropriate cache invalidations [157].

Push-based Caching

The idea that web performance can be improved by servers actively pushing data to web caches was already discussed back in 1995 by [35]. In this approach, the origin server is responsible for keeping the caches up-to-date. This method is beneficial when large files, such as videos, are pushed to networks with high client demand. Typically, clients are aware of the cache locations because the origin informs them about the best locations to access the content. However, to our knowledge, there is currently no standardized way to push content to web caches or CDNs.

Conflict-free Replicated Data Types

CRDTs [138] are used in distributed computing to achieve eventual consistency. Unlike consensus protocols that rely on complex coordination, CRDTs achieve consistency by defining the data type. Each peer can modify a CRDT without coordinating with other peers. It might result in different peers having different states, but the logic of the CRDT ensures consistency when all changes have propagated to all peers. There are two types of CRDTs: state-based and operation-based. State-based CRDTs synchronize by merging the states of two peers, while operation-based CRDTs synchronize only the operations on the data. Successful synchronization between peers only requires a reliable communication, so changes eventually arrive at each peer.

A simple example of a CRDT is a distributed counter. If the data type of the counter allows arbitrary writes, conflicts can occur when two peers write simultaneously. However, if the data type of the counter is defined as a CRDT that allows only adding or subtracting a delta, then each peer can apply the modifications without conflicts. After synchronizing all deltas, the counter's value will be the same on all peers.

Developing new powerful CRDTs for actual web applications is an active field of research that has seen significant progress in recent years. Related works have defined CRDTs for arbitrary JavaScript Object Notation (JSON) data types [91, 131]. JSON CRDT libraries like Automerge [27] go beyond the basic functionalities of a CRDT. For instance, providing mechanisms to uniquely identify a CRDT and WebSocket [105] interfaces to request and synchronize it between peers or a server and its clients.

Representational State Transfer (REST) APIs

REST [67] is a popular design paradigm for APIs in a distributed client-server system, defined in 2000. It covers several principles and constraints for APIs, including statelessness, cacheability, a uniform interface, and support for multi-layered systems. Applications using CDNs are an example for a multi-layered system. The cacheability of REST APIs is defined only on an abstract layer, distinguishing between cacheable and non-cacheable responses. REST is a paradigm offering little about concrete technologies to achieve it. A more concrete format for REST APIs is JSON:API [88], which defines how resources and relations are represented, accessed, and mutated. JSON:API indicates how REST is implemented in practice: using HTTP to access API endpoints, identifying resources with URLs, and transmitting data via JSON. Building on top of HTTP meas REST mostly relies on HTTP caching to implement its cacheability. HTTP caching works well for REST APIs if the data changes infrequently, and it is acceptable to serve outdated information for a limited time. However, it can be problematic if immediate visibility of changes is crucial for the application. For instance, a book-

Figure 4.1: Example client server architecture utilizing CWC for a REST API transmitting JSON.

store lists books available for borrowing, most users may only browse the lists, making caching the list a performance gain. However, when a user borrows a book, they should immediately see the effect, and all other users should see the unavailability. If the API relies solely on HTTP caching, each user (including the borrower) must wait until the TTL expires to observe changes. In the meantime, other users might attempt to borrow the seemingly available book, leading to conflicts.

4.3 CRDT Web Caching (CWC)

We define *CWC* as an approach for distributing application data using CRDTs from a central origin to one or multiple trusted web proxies with the option for proxies to apply changes to the CRDTs directly. The constantly synchronized CRDTs enable the origin to push changes on the data source to web proxies to keep the cache states consistent. If the origin defines a set of allowed mutations, the web proxies can directly apply data changes to the CRDT via uncoordinated distributed writes, effectively enabling caching of simple write requests. The eventually-consistent and conflict-free nature of CRDTs allows web proxies to merge incoming write requests and asynchronously forward the changes to the origin. Hence, the application can remain responsive even when the origin is unavailable (e. g., due to a power outage). We propose CWC as an on-demand protocol upgrade, conducted, e. g., when a client accesses an API endpoint. Therefore, CWC combines pull-based and push-based caching elements, allowing clients to initiate replication while servers actively populate relevant caches with cache updates.

We propose implementing CWC for REST APIs by wrapping the response between origin and proxy in a synchronized JSON CRDT [91]. If proxies should be able to directly modify the CRDT, the contained JSON data must be structured according to a well-defined format, e. g., using JSON:API [88], and the web proxies provide an automatically generated REST interface to mutate the CRDT data with simple CRUD operations. This concept is illlustrated in Fig. 4.1. The figure shows a client accessing a REST API endpoint. The request is forwarded by the proxy to the origin. In a normal setup, the origin would return directly a JSON response; however, in case of CWC, the JSON response is wrapped inside a synchronied JSON CRDT and the proxy will extract

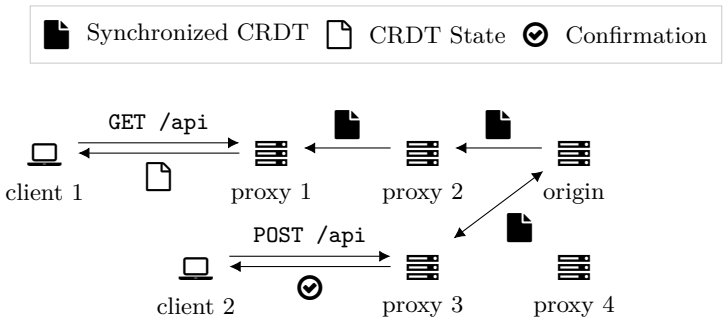


Figure 4.2: Example deployment utilizing CWC (*cf.*, Sosnowski et al. [2]).

the current state of the CRDT and return a JSON response to the client. Hence, the whole approach is transparent to the client.

An example deployment illustrating CWC with multiple proxies is shown in Fig. 4.2. It contains a client making a `GET` request to a REST API endpoint. As the API endpoint is distributed as a CRDT, proxy 1 can directly return the current state of the CRDT to the client. Proxy 1 does not replicate the CRDT directly from the origin but via proxy 2. For read-only APIs, changes to the CRDT propagate only from the origin to the proxies. However, when client 2 requests a mutation with the `POST` request, proxy 3 can immediately apply it on the CRDT, provide a confirmation, and synchronize the changes back to the origin later. Since no client accesses the API over proxy 4, it holds no replica of the CRDT.

4.3.1 On-demand Protocol Upgrade

CWC should integrate well into existing applications. This means, it should only be applied for selected endpoints, integrate seamlessly into existing REST APIs, and the origin should remain in control over the logic applied by the proxy. Hence, we propose CWC as an on-demand protocol upgrade, conducted, e. g., when a client accesses an API endpoint. During the upgrade process, the origin provides metadata informing proxies about the structure of the API, access permissions, allowed mutations, and other details. Any CRDT replica can be removed from a web proxy once all changes have been successfully forwarded to the origin.

An example for an upgrade to a JSON CRDT is shown in Fig. 4.3. The process starts with a client accessing a REST API endpoint that is expected to return a JSON object. A web proxy receives the request and forwards it to the origin according to standard HTTP proxy rules. The web proxy appends its support for CRDTs in an HTTP header, indicating its support for Automerge [27] by adding `Proxy-Upgrade:`

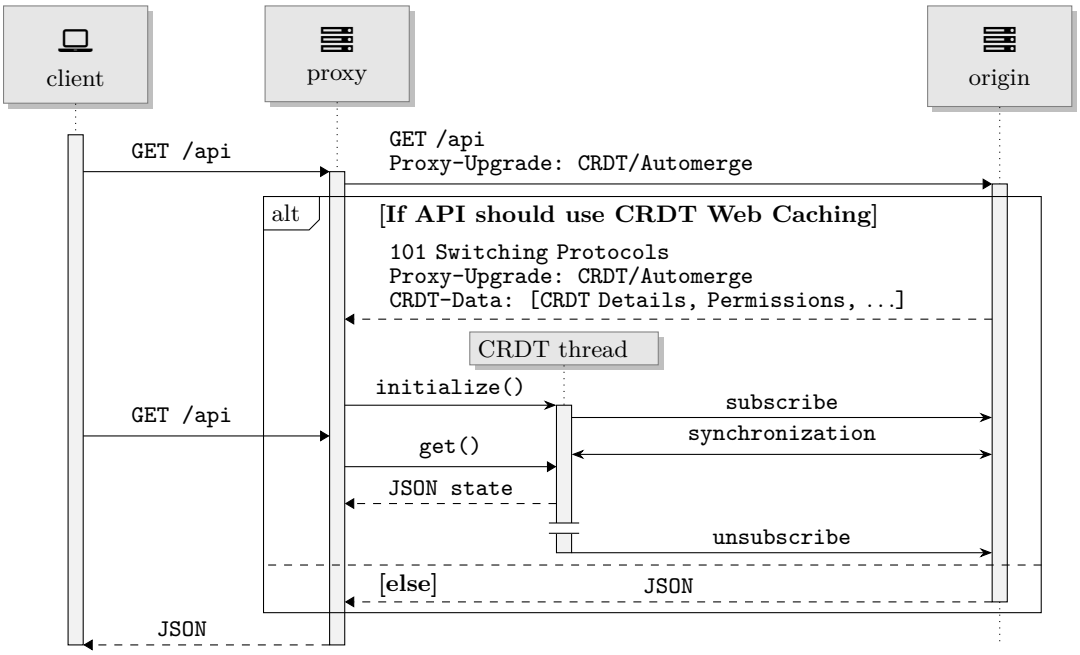


Figure 4.3: Sequence diagram of an example API call utilizing CWC. Future requests can start at the second `GET` request, skipping the initial round-trip to the origin (*cf.*, Sosnowski et al. [2]).

`CRDT/Automerge`. The origin server then decides whether the endpoint should be upgraded. If not, it responds with a regular HTTP response, and the web proxy acts as a shared HTTP cache. However, if the endpoint should be upgraded to a CRDT, the origin responds with a `101 Switching Protocols` response and the necessary details to access the CRDT. The origin informs the web proxy about the permission model, such as a required user authentication, endpoint access-control, and CRDT mutability. Then, the proxy can initialize its local replica of the CRDT and establish synchronization. The local replica will be continuously synchronized in a background thread until the proxy decides to evict it. Once the CRDT instance is fully replicated on the proxy, the current state can be returned to the client as a JSON object. If a client requests the same API endpoint again, the proxy can directly respond with the state of the local replica, skipping the initial round-trip to the origin.

4.3.2 Requirements

CWC for REST APIs can support different levels: basic, advanced, and mutable. The basic level is for read-only data, supports various CRDT definitions (e. g., not limited to JSON CRDTs), and has the following requirements:

R1 A web proxy stores a mapping of REST API endpoints (i. e., URL patterns) to local CRDT replicas. If the origin upgrades an API request to a CRDT, the proxy creates a local replica of the CRDT and maps the observed endpoint to it. Multiple endpoints can map to the same CRDT. A background task on the proxy ensures constant synchronization of all local CRDT with the origin.

R2 Received requests to an unknown endpoint are forwarded to the origin. Different HTTP methods for the same URL are treated as seperate endpoints. Any read-request (`HTTP GET`) to a known endpoint previously upgraded to a CRDT is answered with the current state of the respective CRDT.

R3 Web proxies periodically forward requests to a known CRDT endpoint to the origin to repeat the upgrade process and adjust for changes in the API metadata. If the connection between the origin and the web proxy breaks, the web proxy immediately repeats the upgrade.

R4 The origin informs web proxies about existing permissions during the upgrade, e. g., which users are allowed to access the data. Web proxies enforce the permissions on all client requests.

Advanced CWC enables a web proxy to provide clients with additional functionality for read-only API endpoints. In addition to requirements **R1** to **R4**, it requires:

R5 The origin structures the data of the CRDT according to a standardized format that the web proxy understands, e. g., based on linked data with a well-defined format like JSON:API [88].

R6 The origin informs the web proxy about the structure of the API, different access patterns, and whether only subsets of the CRDT should be served to clients for some of the patterns. The web proxy will store the patterns in the local mapping. This enables web proxies to directly answer a group of API endpoints, e. g., each API call that returns a single object of a collection.

Mutable CWC enables proxies to directly change the data and requires, in addition to **R1**-**R6**:

R7 The origin flags a CRDT as mutable and informs the web proxy which HTTP methods (`GET`, `POST`, `DELETE`, etc.) and which operations are allowed. The origin upgrades all allowed methods to the same CRDT. The web proxy stores the corresponding methods in the local mapping.

R8 A web proxy flags local CRDT replicas as *dirty* if it applies a mutation. The flag is removed when all changes are successfully synchronized with the origin. No dirty replicas are purged from the local cache of a proxy.

4.3.3 Correctness

In order to ensure the safe use of CWC, we must verify that

C1 web proxies only apply CWC to endpoints intended by the application logic,

C2 no information is lost before reaching the origin, and

C3 no conflicts occur when using mutable CWC.

Requirement **R2** states that all requests to new endpoints are initially sent to the origin. This means that the origin's developers have complete control over how to handle every request, ensuring **C1**. Although proxies can respond to unknown endpoints due to the access patterns of advanced CWC, the application developers have specifically defined these patterns in the upgrade. Even if errors in the announced patterns occur and a new version of the origin is deployed, the web proxies will immediately adjust their behavior due to **R3**. CWC assumes that the data source always lies within the origin, and web proxies must only maintain a view. For read-only APIs (basic and advanced), **C2** is guaranteed because all information comes directly from the origin. If the web proxies can modify the CRDT, proxies will not purge their local version containing information unknown to the origin due to **R8**, thus ensuring **C2** for mutable APIs as well. Even if the connection between a web proxy and the origin fails due to network issues or server downtimes, the conflict-free nature of CRDTs guarantees that once synchronization is resumed, the diverged states can be successfully merged to inform the origin about received mutations. The same applies when multiple mutation are applied by the proxies simultaneously, then the CRDTs guarantee per definition that no conflicts occur, guaranteeing **C3**. Problems could occur when proxies apply mutations not allowed by the origin; however, we argue this is an implementation detail and the responsibility of the CDN to offer developers a powerful syntax to specify allowed mutations and to enforce a secure CRDT access. In summary, the correctness of CWC depends on the correctness of the origin-provided configuration; however, CWC provided developers with the necessary control to ensure it.

4.3.4 CDN and Origin Design Considerations

To effectively implement CWC in a CDN, we recommended the following considerations:

iv) HTTP Compatibility. web proxies should behave like regular HTTP proxies and only indicate their ability to switch to CWC with the appropriate upgrade header.

iv) Stateless Upgrade. All necessary information for configuring CWC on a proxy should be provided during the upgrade to avoid needing the proxy to hold any prior state about the origin.

iv) Transparent Proxy. Responses should appear as though they came directly from the origin, allowing the origin to seamlessly mix regular HTTP and upgraded CRDT APIs without the client developer needing to be aware of it.

iv) Access Control. The CDN should enforce access restrictions provided by the origin. An approach like OAuth 2.0 [79] and JSON Web Tokens (JWTs) [87] could enable decentralized client authentication, along with approaches such as:

  iv) Scopes. The origin defines an aspect of the client request as scope, which could be a parameter in the access token—such as a role. Requests with the same scope and to the same endpoint have access to the same CRDT. Requests with a different scope will be treated as a new endpoint, and the origin can decide whether access is denied or the API should be upgraded to the same or a different CRDT.

  iv) Permission Lists. The origin explicitly defines which users can read or modify which JSON attributes, e. g., with a permission description language like [128].

  iv) Invariants. The origin defines a list of invariants. The invariants are a list of condition expressions on the data that always have to be true, similarly to SQL constraints [165].

  iv) Custom Edge Functions. If neither approach is sufficient, developers can deploy a custom function on the web proxy that handles access control.

Recommendations for an origin server looking to make the best use of CWC include:

iv) HTTP per Default. The origin should only upgrade performance-critical APIs and keep the rest as simple HTTP endpoints.

iv) Authenticated CDN. The origin should always authenticate web proxies to prevent unauthorized access to the CRDT, for example, using mutual TLS authentication with a client certificate.

iv) Consider Eventual Consistency. Mutable CRDTs should persist between application restarts to allow receiving synchronization messages before the restart.

iv) CRDTs as Data View. The CRDT of an API should be treated as a view on the data source, with the actual data possibly being stored in a database. Changes to

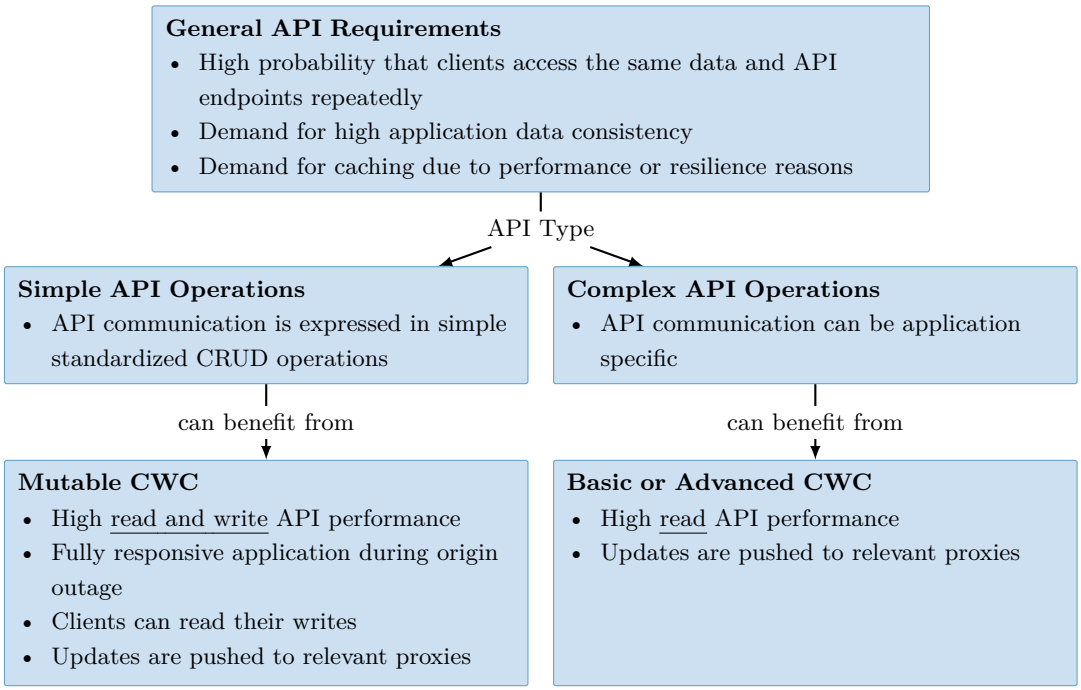


Figure 4.4: Main types of web application comunication that can benefit from CWC.

.

the data source have to be applied to the CRDT, and incoming mutations should be validated before applied in the database.

iv) Protocol Flexibility. If the origin generates the same API as the CDN for accessing the CRDTs, clients can communicate directly with the origin, enabling local testing and independence from a CDN, and allow the CDN to upgrade only frequently used endpoints.

4.3.5 Web Applications that Benefit from CWC

CWC promises improved availability and performance for web applications. However, not every application can take advantage of these enhancements. The following discussion covers two main types of applications communication that can utilize CWC, as illustrated in Fig. 4.4.

Like other web caching methods, CWC's potential can only be realized when there is a high chance that a request can be served with cached content. This is true when clients frequently access the same API endpoints and the same data, allowing the initial caching costs to be amortized by future requests. Compared to TTL or invalidation-

based caching, CWC causes higher initial caching costs due to CRDT synchronization. Therefore, it should only be implemented when the benefits outweigh these costs.

The first type of applications that can benefit from CWC has an API communication logic that can be expressed through simple CRUD operations. These applications can utilize the full potential of mutable CWC. Although the API operations may be simple, this does not imply that the overall application logic has to be simple, as discussed in Section 4.6. Such applications can benefit from increased read and write performance. Additionally, if there is an outage at the origin server, the application can remain fully responsive by serving all requests from cached content. When CWC is used for both read and write API endpoints, clients can even read their own write operations during the outage.

The second type of application has a communication logic too complex to be automatically applied to a CRDT. In these cases, applications can still apply CWC and gain improved consistency and performance through automatic cache updates enabled by synchronized CRDTs. CWC combines push- and pull-based caching, granting CDNs control over where content should be cached and when it can be evicted (the pull aspect), while also allowing applications to update caches actively (the push aspect). If applications utilize advanced CWC rather than basic CWC, they can further enhance the chances of content being served directly from the web proxy. However, if the benefits of the push-based approach (i.e., saving an RTT to the origin for cache updates) do not outweigh the additional implementation and synchronization costs of CWC, then TTL or invalidation-based caching may be a more suitable choice. Moreover, related push-based caching approaches (e. g., proposed by Ninan et al. [119]) could provide similar benefits for this type of application communication.

In conclusion, CWC comes with higher initial caching costs than TTL or invalidation-based caching, and not all web applications should utilize it. Application developers should evaluate whether the potential benefits justify the additional programming and synchronization costs.

4.4 Comparing Web Caching Strategies with a Simulated CDN

To evaluate the performance implications of applying CWC to REST APIs, we simulated a simple CDN deployment with two example applications: a flight booking service and a discussion forum. These applications were chosen to reflect the two communication types that can benefit from CWC, as discussed in Section 4.3.5. The flight scenario

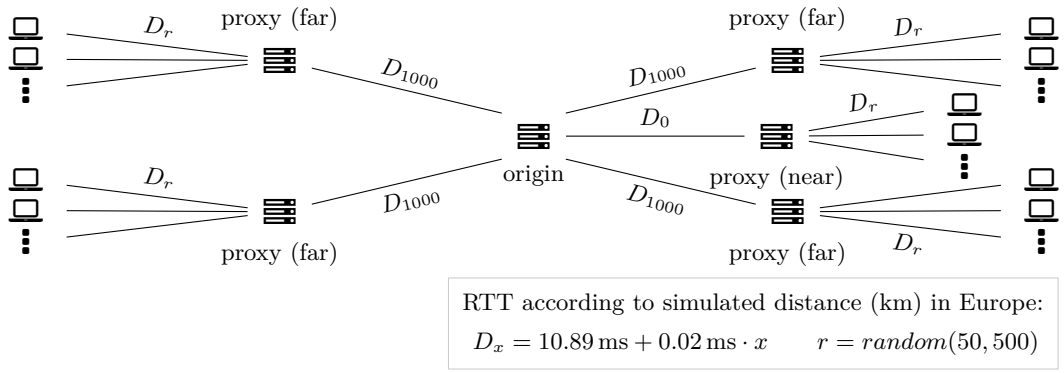


Figure 4.5: Simulated CDN deployment used to evaluate API caching strategies (*cf.*, Sosnowski et al. [2]).

represents write-heavy applications with complex logic that only the origin can handle. In contrast, the forum scenario involves many reads, and only simple CRUD operations. We tested the four caching strategies using nocache, TTLs, invalidations, and CWC.

4.4.1 Experiment Model and Setup

For our study, we modeled a simplified CDN deployment with Mininet [106], a tool capable of simulating complex network typologies on a single machine. The simulation ran on a single server equipped with two *AMD EPYC 7601 32-Core* processors (128 logical cores) and 1 TB of RAM and we pinned the origin, web proxies, and clients to separate cores to minimize a CPU scheduling bias. The deployment comprises multiple load generators, a few proxies, and a single origin spread evenly over a large area, as illustrated in Fig. 4.5. The load generators simulate clients; however, they create higher loads than real-world clients because they constantly make requests without the idle times caused by a human user.

For the experiment, we assumed unlimited storage on all devices; hence, there were no cache evictions due to full storage. For the origin, we used an in-memory dictionary as the data source to simulate fast computation. If we can demonstrate the benefits of using a caching approach for such a basic origin server, real-world applications with longer processing times (due to complex logic, dedicated databases, etc.) should benefit even more.

We simulated the physical distance covered by network connections with artificial delays. The load generators send their requests to the nearest proxy, which either responds to the requests or forwards them to the origin. For the distances to be realistic, we modeled them after the European Union, approximately 4 000 km from west to east and north to

south. Martinez et al. [102] measured network delays worldwide depending on distance and created a regression model to estimate the RTT on different continents based on distance. For Europe, they measured the function D_x from Fig. 4.5, allowing us to simulate physical distances in our experiments. The origin is located at the center to best serve all clients. A single proxy is also located at the center, and four more are evenly distributed around the area, each 1 000 km away from the origin. Up to 100 load generators connect evenly to the five proxies, each being randomly located between 50 km and 500 km from the proxy. The resulting latencies are close to real-world values: [164] measured a median latency of 14 ms targeting CDNs and 34 ms targeting data centers in Europe.

In the experiments, we started with a single load generator and then added a new one every minute up to 100. The load generators continuously made client requests according to a specified scenario.

4.4.2 Evaluated API Caching Strategies

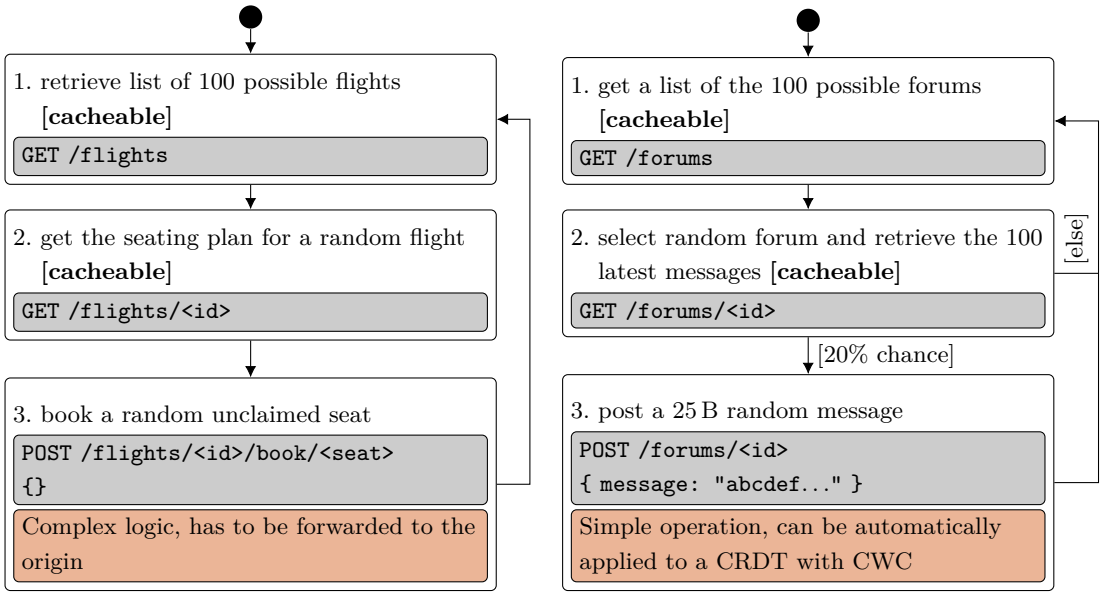
In this work, we compare four different caching strategies:

Nocache The proxies did not cache any responses and only acted as a relay between client and origin.

TTL The APIs utilized an expiration-based strategy where responses to read requests were cached for a defined period. For this study, we choose a TTL of 5 min as representative value for TTL-based caching. Real-world applications could use lower or higher TTL values depending on their concrete requirements.

Invalidation Responses to read requests were cached with a TTL longer than the experiment. When the origin updates the data source, affected cached responses are invalidated. We implemented this functionality by tagging each response of the origin with one or more keys. Whenever the origin returned a response after changing data, the keys to be invalidated were attached. A proxy observing invalidation keys purged associated local entries and forwarded the list to the other proxies.

CWC The APIs used advanced and mutable CWC according to Section 4.3, implemented with Automerge [27]. Our proxies automatically derived REST APIs with the metadata provided during the upgrade. The origin forwarded changes on data objects to all affected CRDTs, effectively updating cached objects on the proxies through Automerge's inherent synchronization protocol.



(a) Write-heavy flight scenario where clients book a seat on a random flight.

(b) Read-heavy forum scenario where clients read messages, and some post new messages.

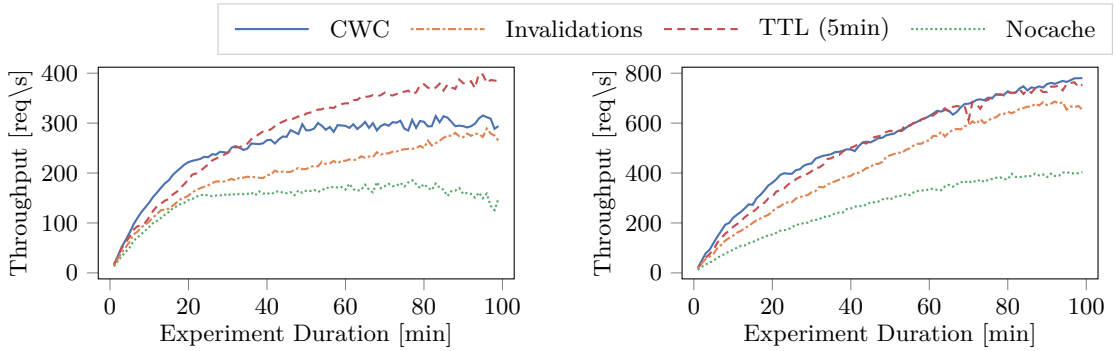
Figure 4.6: The example scenarios we used to compare the API caching strategies. Their behavior is described as state machines (*cf.*, Sosnowski et al. [2]).

4.4.3 Example Scenarios

The experiments are initiated load generators acting as clients according to the state machines in Fig. 4.6. As soon as a client finished the last action of the scenario, it immediately repeated the state machine, as if it was a new client.

In the flight scenario, clients attempt to book a seat on a flight by first requesting a list of 100 possible flights. The client then randomly selects one flight and requests a list of 100 bookable seats for that flight. Each seat has an attribute indicating its availability. If an API caching strategy is being evaluated, both lists are cached. Subsequently, a client attempts to book a random available seat. The booking transaction is assumed to be complex and handled solely by the origin. If a flight is fully booked, the origin removes it from the list of possible flights and creates a new empty one. While caching can enhance performance, it is important for a client to have up-to-date information to avoid attempting to book an already taken seat. The scenario is write-heavy due to each client attempting to book a seat.

In the forum scenario, clients initially requests the list of 100 possible forums, then selects a random forum and retrieve the latest 100 messages. Both read requests are cached if an API caching strategy is evaluated. Then, there is a 20% chance that a client posts a new random message. Mutable CWC is used for the post because adding



(a) Flight Scenario

(b) Forum Scenario

Figure 4.7: Successful requests per second over time. Every 1 min a new load generator was started, increasing the system load. (*cf.*, Sosnowski et al. [2]).

a new entry to an existing list is a simple CRUD operation. The main challenge of this scenario is to enable real-time communication between clients, ensuring they always see the latest messages.

4.4.4 Evaluated Metrics

Both latency and throughput are important for client-server applications. In our experiment, we measured throughput as the number of HTTP requests served per second and latency as the time it takes for the client to send an API request and receive a response. To better compare the strategies, we aggregated each request's latency per time frame and differentiated between read and write requests. Incorporating web proxies into a client-server deployment can improve performance by increasing throughput and reducing latency, but it can lead to consistency issues. To analyze these effects, we need additional metrics such as the *staleness ratio*, *cache-hit ratio*, and the *client-observable inconsistency window*. The staleness ratio indicates the portion of requests answered with outdated data, which can occur even with no caching if content is updated during transmission. The cache-hit ratio reveals the portion of requests that a web proxy could directly serve from its cache. Both are widely used metrics and have been used already in 1997 by [29]. A newer metric, the client-observable inconsistency window, measures the "time between the commit timestamp and the latest possible read of the previous version for systems that do not expose dirty reads" [33].

4.4.5 Application Performance: Throughput and Latency

A general metric that reveals the scalability of a system is throughput. A higher value means more clients can be served at the same time. We measured the number of HTTP requests completed per second for both scenarios in Fig. 4.7. The results reveal

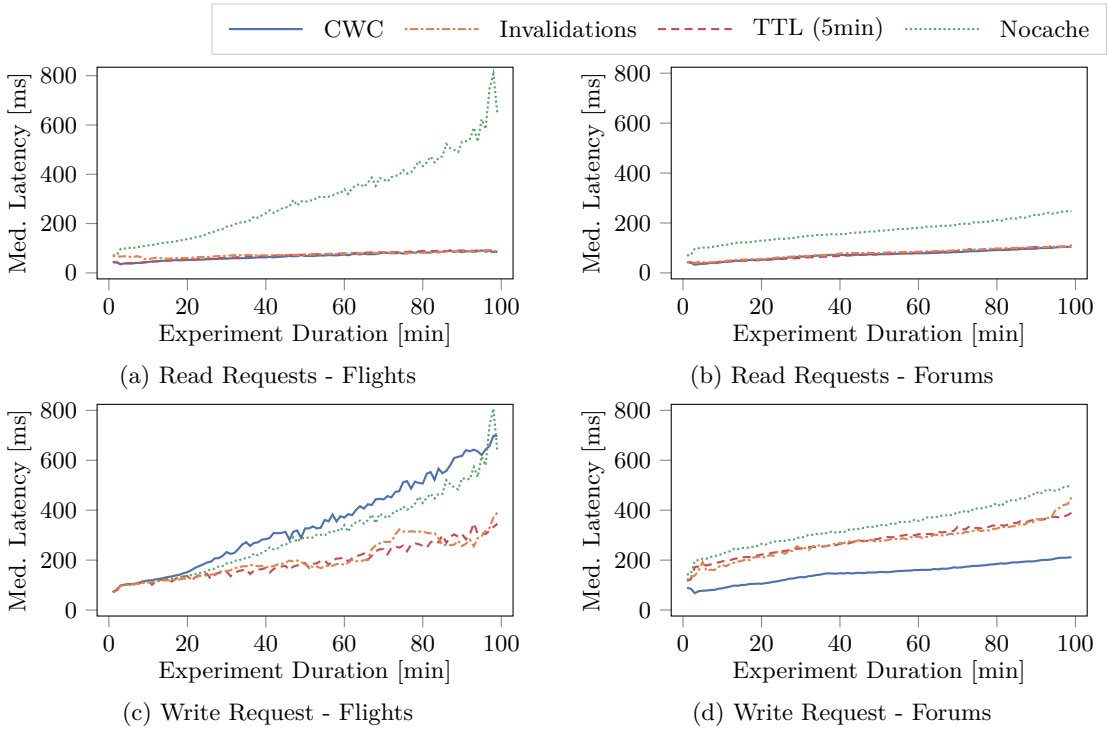


Figure 4.8: Median latency over time and with increasing system load (*cf.*, Sosnowski et al. [2]).

that CWC and TTL-based approaches provide the best throughput, followed by an invalidation-based approach. Interestingly, for a few concurrent clients, CWC outperformed the throughput of TTL-based caching by 19% for 20 concurrent clients in the flight scenario, as the content was actively pushed to proxies, resulting in faster request handling. However, the CRDT synchronization on the origin became a bottleneck under higher system load, allowing TTL-based caching to surpass CWC. Despite this, CWC's throughput never dropped below that of invalidation-based caching. Performance is poorest without caching. Generally, the flight scenario has a lower throughput than the forum scenario because it is more write-heavy. CWC challenges the throughput possible with TTL-based approaches while providing the advantage of fast cache consistency.

The latency of read and write requests over time offers an additional perspective on the application's performance. Fig. 4.8 displays the median read and write latency per minute. It is evident that all three API caching approaches result in similar performance enhancements for read requests compared to no caching, and the increasing load over time has minimal impact. In the absence of caching, the origin becomes the performance bottleneck, leading to a noticeable degradation in latency, particularly in the write-heavy flight scenario. A very different picture is revealed when analyzing the me-

Table 4.3: Summary of the simulation results (*cf.*, Sosnowski et al. [2]).

Scenario	Caching Strategy	req/s[1]	mean[2] read lat.	mean[2] write lat.	mean inc. window[3]	staleness ratio	cache-hit ratio
Flights	CRDT	252.3	91ms	404ms	701ms	6.5%	97.5%
	Invalid.	200.7	107ms	215ms	1 194ms	60.8%	79.5%
	TTL	281.1	79ms	213ms	5min	98.3%	99.3%
	Nocache	148.6	335ms	331ms	3ms	6.0%	0.0%
Forums	CRDT	527.3	95ms	190ms	1 949ms	26.0%	100.0%
	Invalid.	437.7	106ms	326ms	3 685ms	16.8%	81.2%
	TTL	512.1	85ms	306ms	5min	49.3%	99.6%
	Nocache	270.0	185ms	371ms	1ms	0.6%	0.0%

Note: Abbreviated are [1]requests per second, [2]read/write latency, and [3]client-observable inconsistency window.

dian write latency. In the flight scenario, the origin handles every write; consequently, performance is directly dependent on the origin load. Here, the invalidation-based and TTL-based approaches excel as they reduce the number of read requests without significantly impacting the write latency. CWC showed the poorest write performance due to the additional CRDT synchronization costs. However, the forum scenario demonstrated the biggest advantage of CWC, enabling web proxies to directly process and respond to simple write requests, resulting in significantly lower latency than any other approach.

4.4.6 Cache Effectiveness

We previously discussed the benefits of caching REST APIs, but we did not evaluate the quality of the cached data. To address this, we look at additional caching metrics in Table 4.3. It is important to note that the different application behaviors mean that the absolute measured numbers should only be compared within the same scenario.

The table reveals that TTL-based caching offered one of the best performance results with its 99% cache-hit ratio. However, the 5-minute inconsistency window and 98% staleness ratio make this option impractical for fast-changing APIs requiring up-to-date information. Advanced CWC offered the best performance after TTL-based caching, with a 10% lower throughput and 15% higher read latency. Despite this, it still resulted in a 26% higher throughput and 15% lower read latency than invalidation-based caching. However, CWC had around twice the write latency of TTL and invalidation-based caching in the flight scenario due to CRDT synchronization overhead. In the forum scenario, mutable CWC outperformed all other caching strategies because it was the only option that accelerated write requests. Compared to the second-best strategy, TTL-based caching, mutable CWC could reduce the mean write latency by 38%. However, the eventually consistent nature of the mutable CRDTs led to a higher staleness ratio (26%)

compared to invalidation-based caching (17%) and no caching (1%). Interestingly, even without caching some responses were stale due to content changed during transmission.

4.4.7 Conclusion

CWC influences how REST API data is distributed and cached in a network of web proxies from a CDN. To compare the approach with state-of-the-art CDN caching strategies we simulated a simple CDN deployment and two example scenarios with Mininet. We compared simple forwarding, TTL-based caching, invalidation-based caching, and CWC. Our results show that while TTL-based approaches can achieve better performance, they are not suited for the dynamic nature of REST APIs due to large inconsistency windows. Invalidation-based strategies where the origin actively purges outdated content from proxies can enable REST API caching. However, our approach outperformed this strategy as new content is automatically pushed to relevant proxies. While CWC demonstrated low read latency, it can come at the cost of higher write latency under high system load due to the overhead of the CRDT synchronization. However, mutable CWC outperformed every other strategy due to the acceleration of write requests. To conclude, CWC allows developers to significantly increase REST API availability and performance above the current state-of-the-art.

4.5 Related Work

Ever since the development of the World Wide Web, there have been numerous efforts to enhance its performance. Especially web caching of static content (documents, texts, videos, audio, etc.) is a well-elaborated topic.

In 1995, Abrams et al. [13] discussed the general potential of web proxy caching covering HTTP. Bestavros [35] highlighted the performance benefits of servers pushing data to proxies closer to the client based on local popularity and emphasized the need for actively invalidating cached content. Chankhunthod et al. [47] suggested using hierarchical caches that mirror the Internet topology. In 1997, Baentsch et al. [29] refined the concept and demonstrated the performance advantages of replication over caching. Iyengar and Challenger [86] explored caching dynamic data on proxies and suggested an invalidation-based approach. Cao and Liu [43] compared approaches such as TTL, “polling-every-time”, and invalidation, concluding that an invalidation-based protocol provides the best cache consistency. However, our results indicate that replication-based caching still outperforms invalidations.

Over time, the Internet landscape has evolved, with caching primarily carried out by end devices (e. g., a browser) or CDNs specifically configured by application developers.

Ninan et al. [119] argued that existing cache consistency mechanisms are unsuitable for CDNs. They propose a concept of cooperative leases where the origin notifies the CDN of any changes and sends an invalidation or updated object. More recently, Abolhassani et al. [12] showed that improving caching strategies on web proxies remains relevant, interpreting push- and pull-based caching approaches as a cost optimization problem and proposing a combined approach. Wingerath et al. [171] developed a different approach to handling dynamic data, suggesting to push application-specific caching logic to the client. Several related works have explored different strategies to pre-fetch content and populate a cache before a user accesses it. For instance, Wan et al. [167] proposed grouping users based on their navigational patterns and pre-fetching web requests according to a common user profile. With the increasing diversity of CDNs, new opportunities have emerged as well. Hou and Chen [81] discussed the additional dimension of monetary costs arising from the availability of different cloud providers and suggested a cost-aware strategy optimized for cloud storage caching.

REST, defined by Fielding [67], was designed to build upon the existing caching infrastructure; hence, can leverage the extensive research and technological advancements in the field. To the best of our knowledge, CRDTs have not been considered for web caching yet nor has an approach similar to CWC been proposed.

4.6 Discussion

This work proposes adding new data-aware functionality to existing web proxies and we want to discuss some aspects.

Intended Scenarios

We do not think all REST API endpoints should adopt CWC because it adds complexity to the origin code and its maintainability. However, it can greatly benefit performance-critical endpoints. The basic and advanced level of CWC can be a valuable tool for developers to enhance the performance and scalability of read-only API endpoints where the chance of stale content should be low. If stale content is not a problem, TTL-based caching could be easier to realize. The applicability of mutable CWC is more limited as web proxies cannot be aware of the application-specific semantic of concrete REST APIs. Hence, web proxies can only perform basic mutations on the data, such as the CRUD operations (e. g., inserting an entry into a list or updating an attribute). If these mutations are sufficient, mutable CWC can be a powerful tool to increase API performance.

Complex Logic with Mutable CWC

In addition to simple CRUD logic, mutable CWC can be utilized to accelerate complex application logic. A possible design pattern would be to store an unfinished state in the CRDT and do the final processing asynchronously on the origin. For instance, when users place orders with an enhanced CRDT API, the origin can process and confirm them later. This way, CWC is used like a message queue with web proxies acting as message brokers. Another pattern involves storing intermediate states in the CRDT API to create fast and reactive clients. For example, the currently selected chairs are synchronized with a fast CRDT API in a cinema booking application to prevent conflicting bookings. However, the final reservation is still handled only by the origin.

Transparent CRDT Operations

CRDTs were primarily designed for clients, e. g., to enable collaborative editing [91]. In case of CWC, web proxies could provide clients with direct access to the CRDTs synchronized for an API endpoint, but this adds new challenges such as security concerns. Web application clients are typically untrusted, and ensuring the integrity of changes made to CRDTs can be complex. While it is possible to define permissions on CRDTs [128], this approach increases system complexity and debugging challenges. they do not need a web proxy to enable a responsive application. To minimize changes to existing systems and ensure the focus on REST API caching challenges, we have kept the CRDT mechanism transparent to the client. It is subject to future work how web proxies can grant clients direct access to a CWC CRDTs and whether CWC is still needed in such a setting.

Limitations and Future Work

CWC is an approach that operates at the application layer (e. g., HTTP) and interacts with the specific logic of a web application. The effectiveness of CWC largely depends on how well a particular application can leverage the proposed caching behavior. We explored two types of application communication that could benefit from CWC and implemented simplified example scenarios to represent each type. However, the simulated CDN and the scenarios we evaluated are considerably simpler than real-world applications or CDN deployments. As a result, the measured outcomes only suggest how the four caching strategies could interact with a specific web application. We aimed to keep the evaluation straightforward so that readers can easily understand the findings and their implications and apply them to their own situations. Future work could focus on assessing CWC in a more realistic setting.

Another aspect we want to highlight is the selection of evaluation metrics. Our analysis concentrated on cache-specific metrics without examining the impact of different

caching strategies on resource costs, such as CPU, memory, or network bandwidth. For instance, Fig. 4.7a indicates that CWC outperformed all other approaches for up to 20 parallel load generators; however, this advantage diminished as system load increased. CWC likely encountered a resource bottleneck during our simulation. While manually monitoring CPU and memory usage on our experimental server, we observed high loads on the origin process and the virtualized switches from Mininet, suggesting that both may have faced a CPU bottleneck. Future research could involve a more thorough investigation of these aspects.

Lastly, we did not evaluate potential security concerns related to adopting CWC. Since CWC enables web proxies to manage application data independently, this could open new attack vectors.

4.7 Reproducibility

To support the reproducibility of our research, we created a website [145] that provides access to published data, scripts, and code. The repository contains the experiment scripts used to compare the web caching strategies with a simulated CDN, the CWC code, and the analysis scripts used to generate the plots and tables from this chapter. We published our measurement data through Zenodo [146].

4.8 Key Results

This chapter introduces CWC as a new method for distributing REST API data to web proxies. It enables web caching for additional types of communication, in particular simple API requests that change application state. We outline the requirements for achieving CWC on three levels: basic, advanced, and mutable. We propose design considerations for CDNs and origins to implement CWC and argue about the correctness of the approach. A custom on-demand protocol upgrade allows to include CWC seamlessly into established applications and existing REST APIs and ensures the origin server remains in control over the application logic.

We showed that CWC significantly benefits web applications availability and performance in scenarios where the API communication can be expressed in simple CRUD operations and where it is vital to provide up-to-date information. In scearios requiring more complex logic to process mutating requests, CWC provides increased performance and consistency compared to TTL and invalidation-based caching due to the inherent push-based caching semantic.

This chapters demonstrates how web application availability and performance can be enhanced by expanding the functionality of web proxies and transitioning from simple HTTP request handling to more data-aware approaches. We argue that the existing semantics of JSON CRDTs and standards like JSON:API are sufficient to enable advanced data-aware proxies—the foundation of CWC for REST APIs.

4.9 Author's Contributions

The contents of this chapter are based on joint work with Richard von Seck, Florian Wiedner, and Georg Carle in the context of the conference publication *CRDT Web Caching: Enabling Distributed Writes and Fast Cache Consistency for REST APIs* [2]. The chapter contains and extends content from the published work. The author of this thesis led the research and made significant contributions to the conception and design of the study, elaborated the CWC approach, conducted the analyses, and contributed the majority of the published manuscript. Additionally, the author designed and developed a proof-of-concept implementation of CWC and the Mininet simulation to compare the four caching strategies.

For this thesis, the author extended the background in Section 4.2 to explain the various concepts and considerations that formed the basis of the design of CWC. This includes the added Table 4.1, explaining the relation between system latency and distribution level. To better differentiate related caching concepts, Table 4.2 was added. In this thesis, the CWC concept is explained in more detail and the author added Fig. 4.1. Furthermore, the author added an argumentation about the correctness of the approach, design considerations for both a CDN and an origin that wants to utilize CWC, and a general overview of the two main types of web application communication that can benefit from CWC to better contextualize the results obtained from evaluating the two web application scenarios. These additions resulted in the added Sections 4.3.3 to 4.3.5 and Fig. 4.4. Finally, the discussion in Section 4.6 was extended to cover limitations and future work regarding the evaluation of this chapter and potential approaches for realizing more complex application logic with simple CRUD operations, suggesting that with a more CWC-aware API design it is possible for more complex API logic to benefit from CWC as well.

Chapter 5

Conclusion

This chapter summarizes the key findings of this thesis regarding the challenges of enhancing the security, performance, and availability of TLS-based systems on the Internet. Since TLS accounts for a significant portion of Internet traffic, addressing these challenges for TLS-based systems can significantly improve the Internet's overall security, performance, and availability. The Internet has become essential to modern society, so addressing these challenges is a valuable research objective. However, the Internet is also constantly evolving, so improvements in these areas are an ongoing effort and a source of considerable research potential. In concluding this thesis, we take the opportunity to discuss several open research questions that are worthy of investigation in the future.

5.1 Key Findings

In the following paragraphs, we present the key findings of this thesis, structured around the main research objectives outlined in Section 1.2.

RO1: Deriving Cyber Threat Intelligence by modeling and measuring TLS server characteristics. We presented two main methodologies to enhance Internet security by deriving security-relevant information from TLS-based systems using data obtained from active measurements: Active TLS Fingerprinting and a graph-based threat propagation using the ITEG. We demonstrated that both concepts can generate valuable CTI by showing their effectiveness in detecting C2 servers using a threshold-based classification technique. We further refined Active TLS Fingerprinting into two categories, fixed-probe and exhaustive fingerprinting, and proposed a concrete approach for each

category, EFACTLS and DissecTLS, respectively. In order to acquire sufficient information for effective classification, we proposed a heuristic entropy optimization of fixed scanning probes. Using this technique, we obtained an effective combination of scanning probes for EFACTLS. For exhaustive scanners, we suggested a model-driven design for probes and proposed a concrete approach as part of DissecTLS. Our work on the ITEG demonstrated how combining Internet-wide graph modeling with a threat propagation technique can reveal malicious activity. Based on Internet-wide TLS and DNS measurements, we utilized existing C2 server blocklists as algorithm input to identify new servers before they were added to the blocklist. Lastly, we compared how existing methods for distributing certificate revocation information can be used to create a comprehensive dataset of global certificate revocations. Our findings showed that a dataset formed by merging Internet scans and the CCADB CRLs provided the most extensive view of certificate revocations.

RO2: Modeling TLS protocol characteristics to identify performance implications of negotiated algorithms. To understand the performance implications of adopting new cryptographic developments in TLS-based systems, we proposed an approach for measuring and modeling TLS handshakes based on a local testbed under various emulated network conditions. We compared traditional and PQC algorithms for TLS. We demonstrated that, except for Bike and SPHINCS$^+$, PQC algorithms performed similarly to traditional algorithms at NIST security level one and better at higher levels. Hybrid algorithms showed no significant performance drawbacks, while the pre-quantum part of the algorithm was mainly the performance bottleneck. However, the generally larger key sizes of PQC pose a challenge in cases of limited bandwidth, slowing down the handshake and worsening the effects of packet loss. Some algorithm choices involved a trade-off between CPU costs and bandwidth under such conditions. Additionally, we could show new performance optimizations due to the larger keys. We indicated how adjusting the initial TCP CWND and immediately sending TLS messages after their computation—which was not the default in the investigated OpenSSL library—can be tuned to enhance performance.

RO3: Extending the applicability of web caching to a broader range of TLS-based systems. We introduced CWC as a novel approach to improve the availability and performance of TLS-based web applications. This method allows origin servers to push cache updates to web proxies and web proxies to immediately and independently process mutating requests and only later synchronize resulting changes with the origin. We proposed implementing CWC for REST APIs based on an on-demand protocol upgrade, enabling seamless integration into existing APIs while keeping the origin server in control

of the applied caching logic. We outlined the requirements and design considerations for CWC. We argued that the most valuable performance and availability benefits can only be achieved if the web application expresses API communication through standardized CRUD messages. In other scenarios, minor performance advantages can be achieved due to the push-based cache semantic of CWC. Using a simulated CDN and example web API scenarios, we compared CWC with state-of-the-art API caching approaches relying on a TTL, active invalidation, or no caching. Our results demonstrated that CWC matched the performance of TTL-based caching and provided the consistency of invalidation-based caching while being the only option capable of accelerating state-changing client requests.

5.2 Future Work

Our results demonstrated that data obtained through active Internet measurements can be effectively utilized for security-related use cases. Such use cases include directly searching for malicious activity and enhancing other approaches, such as passive flow classification. This thesis concentrated on TLS and some DNS and HTTP characteristics to reveal hidden relationships among malicious servers. These protocols are complex, and we likely did not identify every aspect that could uncover such relationships. Additional opportunities may arise from future protocol developments that are worth investigating.

Advanced machine learning techniques could be a promising alternative for automatically learning relevant features and improving classification performance. We also found that combining data from different sources can significantly enhance detection capabilities. Additional parameters could be actively measured and modeled as threat indicators, including TCP, QUIC, BGP, and HTTP metadata, and combined with our Active TLS Fingerprinting approaches. Zirngibl et al. [7] already showed that it is possible to fingerprint QUIC libraries on servers, as QUIC is also using TLS, combining the approaches offers an interesting opportunity to acquire a more comprehensive view on TLS-based servers. This data would not only benefit fingerprinting approaches but could also be incorporated into the ITEG model. We demonstrated how a global view of revoked certificates can be efficiently obtained. While many revocations were published without a reason code, some indicated malicious activity, which could be a valuable source of threat indicators that can be integrated with the ITEG to identify additional threats.

Furthermore, our ITEG model is not limited to security applications; it has the potential to help address various other research questions as well. For example, the relation-

ships within the ITEG could indicate business connections between companies based on shared resources. Although we have observed such relationships multiple times through manual inspection, an interesting research opportunity would be to automatically detect these cases. Exploring machine learning with graph neural networks could be a promising direction for this research.

Our research on PQC indicated that cryptographic algorithms should be evaluated in real network settings to fully comprehend their performance implications, as changing parameters can lead to unforeseen side effects. Extending our analyses to a broader range of network environments could open up potential areas for future research, especially concerning new network developments and protocols, e. g., examining them in 6G networks as soon as they are available.

Another consideration is that our analyses of post-quantum algorithms used in TLS only covered the currently relevant PQC algorithms at the time of writing. This field is continually evolving, and NIST has already announced [153] several additional DSAs under consideration for standardization. Our methodology can serve as a foundation for future research analyzing these algorithms. We recommend using a local testbed similar to our methodology that provides a stable baseline for comparing algorithms and isolating root causes. However, such analyses could extend our approach by additionally considering performance implications for real-world applications. For instance, the impact of a slow handshake may become negligible for long-lasting connections or may be overshadowed by the poor performance of web application servers.

Web caching is an established and extensively researched topic, yet improvements remain a relevant area for investigation. Our approach to finding an innovative contribution in this well-explored field was to introduce a new perspective by integrating software engineering trends for web applications with caching logic. This led to the concept of a more data-aware web proxy, capable of enhancing the performance and availability of proxied content beyond the current state-of-the-art. CWC is one method through which such a proxy could benefit web applications; however, the idea of a more data-aware network infrastructure opens up numerous possibilities for additional research questions and future innovations. While the general concept is not new, we emphasize that leveraging existing semantics and widely adopted standards, such as JSON:API or CRDTs, may be sufficient for developing advanced data-aware infrastructure.

Chapter A

Appendix

A.1 List of Acronyms

AJAX Asynchronous JavaScript and XML.
ALPN Application Layer Protocol Negotiation.
API Application Programming Interface.
AS Autonomous System.

BGP Border Gateway Protocol.

C2 Command and Control.
CA Certificate Authority.
CCADB Common CA Database.
CDN Content Delivery Network.
CH TLS Client Hello.
CPU Central Processing Unit.
CRDT Conflict-free Replicated Data Type.
CRL Certificate Revocation List.
CRUD Create Read Update Delete.
CT Certificate Transparency.
CTI Cyber Threat Intelligence.
CWC CRDT Web Caching.
CWND Congestion Window.
CZDS Centralized Zone Data Service.

DAG Directed Acyclic Graph.
DDoS Distributed Denial of Service.
DH Diffie–Hellman.
DNS Domain Name System.
DSA Digital Signature Algorithm.

ECDH Elliptic Curve DH.
ECDSA Elliptic Curve DSA.
EMS Extended Master Secret.
ETM Encrypt Then Mac.

FP false positive.

GCN Graph Convolutional Network.
GSB Google Safe Browsing.

HTTP Hypertext Transfer Protocol.
HTTPS Hypertext Transfer Protocol Secure.

IANA Internet Assigned Numbers Authority.

ID Identifier.
IDS Intrusion Detection System.
IP Internet Protocol.
ITEG Internet-wide TLS Ecosystem Graph.

JSON JavaScript Object Notation.
JWT JSON Web Token.

KA Key Agreement.

MITM Man-in-the-Middle.
MTU Maximum Transmission Unit.

NIC Network Interface Card.
NIST National Institute of Standards and Technology.

OCSP Online Certificate Status Protocol.

P2P Peer-to-Peer.
PKI Public Key Infrastructure.
PP true observations.
PQC Post-Quantum Cryptography.
PTP Probabilistic Threat Propagation.

QC Quantum Computer.

RAM Random-Access Memory.
RDAP Registration Data Access Protocol.
RDF Resource Description Framework.
REST Representational State Transfer.
RO Research Objective.
RQ Research Question.
RSA Rivest–Shamir–Adleman.
RTT Round Trip Time.

SAN Subject Alternative Name.
SH TLS Server Hello.
SNI Server Name Indication.
SPA Single-Page Application.
SSH Secure Shell Protocol.

TCP Transmission Control Protocol.
TLS Transport Layer Security.
TP true positive.
TTL Time to Live.

UI User Interface.
URL Uniform Resource Locator.

VT VirusTotal.

XMSS Extended Merkle Signature Scheme.

A.2 Glossary

Active TLS Fingerprinting The process of actively interacting with a TLS server to reveal undisclosed information.

DissecTLS Exhaustive Active TLS Fingerprinting approach, published by Sosnowski et al. [5].

EFACTLS Fixed-probe Active TLS Fingerprinting approach based on 10 optimized CHs, published by Sosnowski et al. [8].

JARM Fixed-probe Active TLS Fingerprinting approach, published by Althouse et al. [19].

JSON:API A concrete specification for building REST APIs; *i.a.*, defines the format for objects, relations, read-, and write-requests based on JSON. Available under [88].

Server Behavior The totality of the capabilities, the interpretation, and the configuration of TLS on a server, which influence the outcome of the TLS handshake.

A.3 List of Figures

A.4 List of Tables

Bibliography

Conference Publications with Author's Contribution

[1] Markus Sosnowski, Patrick Sattler, Johannes Zirngibl, Tim Betzer, and Georg Carle. "Propagating Threat Scores with a TLS Ecosystem Graph Model Derived by Active Measurements". In: *Proc. Network Traffic Measurement and Analysis Conference (TMA)*. Dresden, Germany, 2024. DOI: 10.23919/TMA62044.2024.10559063.

[2] Markus Sosnowski, Richard von Seck, Florian Wiedner, and Georg Carle. "CRDT Web Caching: Enabling Distributed Writes and Fast Cache Consistency for REST APIs". In: *Proc. Int. Conference on Network and Service Management (CNSM)*. Prague, Czech Republic, 2024. DOI: 10.23919/CNSM62983.2024.10814315.

[3] Markus Sosnowski, Florian Wiedner, Eric Hauser, Lion Steger, Dimitrios Schoinianakis, Sebastian Gallenmüller, and Georg Carle. "The Performance of Post-Quantum TLS 1.3". In: *Proc. ACM Int. Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. Paris, France, 2023. DOI: 10.1145/3624354.3630585.

[4] Markus Sosnowski, Johannes Zirngibl, Patrick Sattler, Juliane Aulbach, Jonas Lang, and Georg Carle. "An Internet-Wide View on HTTPS Certificate Revocations: Observing the Revival of CRLs via Active TLS Scans". In: *Proc. IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. 2024. DOI: 10.1109/EuroSPW61312.2024.00038.

[5] Markus Sosnowski, Johannes Zirngibl, Patrick Sattler, and Georg Carle. "DissecTLS: A Scalable Active Scanner for TLS Server Configurations, Capabilities, and TLS Fingerprinting". In: *Proc. Passive and Active Measurement Conference (PAM)*. 2023. DOI: 10.1007/978-3-031-28486-1_6.

[6] Markus Sosnowski, Johannes Zirngibl, Patrick Sattler, Georg Carle, Claas Grohnfeldt, Michele Russo, and Daniele Sgandurra. "Active TLS Stack Fingerprinting:

Characterizing TLS Server Deployments at Scale". In: *Proc. Network Traffic Measurement and Analysis Conference (TMA)*. ★ **Best Paper Award**. Enschede, Netherlands, 2022.

[7] Johannes Zirngibl, Florian Gebauer, Patrick Sattler, Markus Sosnowski, and Georg Carle. "QUIC Hunter: Finding QUIC Deployments and Identifying Server Libraries Across the Internet". In: *Proc. Passive and Active Measurement Conference (PAM)*. 2024. DOI: 10.1007/978-3-031-56252-5_13.

Journal Publications with Author's Contribution

[8] Markus Sosnowski, Johannes Zirngibl, Patrick Sattler, Georg Carle, Claas Grohnfeldt, Michele Russo, and Daniele Sgandurra. "EFACTLS: Effective Active TLS Fingerprinting for Large-Scale Server Deployment Characterization". In: *IEEE Transactions on Network and Service Management* (2024). DOI: 10.1109/TNSM.2024.3364526.

References

[9] Donald E. Eastlake 3rd. *Transport Layer Security (TLS) Extensions: Extension Definitions*. RFC 6066. 2011. DOI: 10.17487/RFC6066.

[10] Josh Aas. *Why ninety-day lifetimes for certificates?* 2015. URL: https://letsencrypt.org/2015/11/09/why-90-days.html (visited on Sept. 4, 2024).

[11] Tareq Abedrabbo, Nicki Watt, Dominic Fox, and Aleksa Vukotic. *Neo4j in Action*. Manning, 2014.

[12] Bahman Abolhassani, John Tadrous, Atilla Eryilmaz, and Serdar Yüksel. "Optimal Push and Pull-Based Edge Caching For Dynamic Content". In: *IEEE/ACM Transactions on Networking* (2024). DOI: 10.1109/TNET.2024.3352029.

[13] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, and Edward A. Fox. "Caching Proxies: Limitations and Potentials". In: *Proc. ACM Int. Conference on World Wide Web (WWW)*. Boston, MA, USA, 1995. DOI: 10.1145/3592626.3592635.

[14] abuse.ch. *Feodo Tracker*. URL: https://feodotracker.abuse.ch/ (visited on July 5, 2024).

[15] abuse.ch. *SSLBL - Detecting malicious SSL connections*. URL: https://sslbl.abuse.ch/ (visited on July 5, 2024).

[16] Gorjan Alagic, Daniel Apon, David Cooper, et al. *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*. Tech.

rep. National Institute of Standards and Technology, 2022. DOI: 10.6028/NIST.IR.8413-upd1.

[17] John Althouse. *JA4+ Network Fingerprinting*. 26, 2023. URL: https://blog.foxio.io/ja4+-network-fingerprinting (visited on Oct. 14, 2024).

[18] John Althouse, Jeff Atkinson, and Josh Atkins. *TLS Fingerprinting with JA3 and JA3S*. 15, 2019. URL: https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967 (visited on July 5, 2024).

[19] John Althouse, Andrew Smart, Randy Nunnally Jr., and Mike Brady. *Easily Identify Malicious Servers on the Internet with JARM*. 17, 2020. URL: https://engineering.salesforce.com/easily-identify-malicious-servers-on-the-internet-with-jarm-e095edac525a (visited on July 5, 2024).

[20] Johanna Amann, Oliver Gasser, Quirin Scheitle, Lexi Brent, Georg Carle, and Ralph Holz. "Mission accomplished? HTTPS security after diginotar". In: *Proc. ACM Internet Measurement Conference (IMC)*. London, United Kingdom, 2017. DOI: 10.1145/3131365.3131401.

[21] Blake Anderson and David A. McGrew. *Accurate TLS Fingerprinting using Destination Context and Knowledge Bases*. 2020. DOI: 10.48550/arXiv.2009.01939.

[22] Apache Software Foundation. *Apache Spark™ - Unified engine for large-scale data analytics*. 2018. URL: https://spark.apache.org/ (visited on July 16, 2024).

[23] Apple. *Apple Root Certificate Program*. 2023. URL: https://www.apple.com/certificateauthority/ca_program.html (visited on Jan. 13, 2025).

[24] *Artifact Review and Badging Version 1.1*. 2020. URL: https://www.acm.org/publications/policies/artifact-review-and-badging-current (visited on Nov. 5, 2024).

[25] Frank Arute, Kunal Arya, Ryan Babbush, et al. "Quantum Supremacy using a Programmable Superconducting Processor". In: *Nature* (2019). DOI: 10.1038/s41586-019-1666-5.

[26] Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, et al. *SPHINCS$^+$ Submission to the NIST post-quantum project, v.3.1*. 2022.

[27] Automerge contributors. *Automerge CRDT - Build local-first software*. 2024. URL: https://automerge.org/ (visited on Dec. 16, 2024).

[28] Michael Baentsch et al. *OQS-OpenSSL_1_1_1*. URL: https://github.com/open-quantum-safe/openssl (visited on July 17, 2024).

[29] Michael Baentsch, Lothar Baum, Georg Molter, Steffen Rothkugel, and Peter Sturm. "Enhancing the Web's Infrastructure: From Caching to Replication". In: *IEEE Internet Computing* (1997). DOI: 10.1109/4236.601083.

[30] Elaine Barker. *Recommendation for Key Management: Part 1 - General*. Tech. rep. National Institute of Standards and Technology, 2020. DOI: 10.6028/NIST.SP.800-57pt1r5.

[31] Richard Barnes, Jacob Hoffman-Andrews, Daniel McCarney, and James Kasten. *Automatic Certificate Management Environment (ACME)*. RFC 8555. 2019. DOI: 10.17487/RFC8555.

[32] Marcus Basalla, Johannes Schneider, Martin Luksik, Roope Jaakonmäki, and Jan Vom Brocke. "On Latency of E-Commerce Platforms". In: *Journal of Organizational Computing and Electronic Commerce* (2021). DOI: 10.1080/10919392.2021.1882240.

[33] David Bermbach. "Benchmarking Eventually Consistent Distributed Storage Systems". PhD thesis. KIT Scientific Publishing, 2014. DOI: 10.5445/KSP/1000039389.

[34] Daniel J. Bernstein. *The inability to count correctly: Debunking NIST's calculation of the Kyber-512 security level*. 2023. URL: https://blog.cr.yp.to/20231003-countcorrectly.html (visited on Jan. 13, 2025).

[35] Azer Bestavros. *Demand-based Document Dissemination for the World-Wide Web*. Tech. rep. Boston University, 1995.

[36] Birk Blechschmidt and Quirin Scheitle. *MassDNS*. URL: https://github.com/blechschmidt/massdns (visited on July 5, 2024).

[37] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. "Fast unfolding of communities in large networks". In: *Journal of Statistical Mechanics: Theory and Experiment* (2008). DOI: 10.1088/1742-5468/2008/10/P10008.

[38] Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and David Cooper. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280. 2008. DOI: 10.17487/RFC5280.

[39] Sergey Brin and Lawrence Page. "The anatomy of a large-scale hypertextual Web search engine". In: *Computer Networks and ISDN Systems* (1998). DOI: 10.1016/S0169-7552(98)00110-X.

[40] Daniel R. L. Brown. *SEC 2: Recommended Elliptic Curve Domain Parameters*. 27, 2010. URL: http://www.secg.org/sec2-v2.pdf.

[41] Kevin Bürstinghaus-Steinbach, Christoph Krauß, Ruben Niederhagen, and Michael Schneider. "Post-Quantum TLS on Embedded Systems: Integrating and Evaluating Kyber and SPHINCS+ with mbed TLS". In: *Proc. ACM Asia Conference on Computer and Communications Security (ASIA CCS)*. Taipei, Taiwan, 2020. DOI: 10.1145/3320269.3384725.

[42] Frank Cangialosi, Taejoong Chung, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. "Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem". In: *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Vienna, Austria, 2016. DOI: 10.1145/2976749.2978301.

[43] Pei Cao and Chengjie Liu. "Maintaining Strong Cache Consistency in the World Wide Web". In: *IEEE Transactions on Computers* (1998). DOI: 10.1109/12.675713.

[44] Kevin M. Carter, Nwokedi Idika, and William W. Streilein. "Probabilistic threat propagation for malicious activity detection". In: *Proc. IEEE Int. Conference on Acoustics, Speech and Signal Processing*. Vancouver, BC, Canada, 2013. DOI: 10.1109/ICASSP.2013.6638196.

[45] Censys.io. *Host Definitions - Censys*. URL: https://search.censys.io/search/definitions (visited on July 5, 2024).

[46] Chia-ling Chan, Romain Fontugne, Kenjiro Cho, and Shigeki Goto. "Monitoring TLS Adoption using Backbone and Edge Traffic". In: *Proc. IEEE Int. Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. Honolulu, HI, USA, 2018. DOI: 10.1109/INFCOMW.2018.8406957.

[47] Anawat Chankhunthod, Peter Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. "A Hierarchical Internet Object Cache". In: *Proc. USENIX Annual Technical Conference (USENIX ATC)*. San Diego, CA, USA, 1996.

[48] Chromium. *HSTS Preload List*. URL: https://hstspreload.org/ (visited on July 16, 2024).

[49] Chronicle Security. *VirusTotal*. URL: https://www.virustotal.com (visited on July 16, 2024).

[50] Taejoong Chung, Yabing Liu, David Choffnes, Dave Levin, Bruce MacDowell Maggs, Alan Mislove, and Christo Wilson. "Measuring and Applying Invalid SSL Certificates: The Silent Majority". In: *Proc. ACM Internet Measurement Conference (IMC)*. Santa Monica, California, USA, 2016. DOI: 10.1145/2987443.2987454.

[51] Taejoong Chung, Jay Lok, Balakrishnan Chandrasekaran, et al. "Is the Web Ready for OCSP Must-Staple?" In: *Proc. ACM Internet Measurement Conference (IMC)*. Boston, MA, USA, 2018. DOI: 10.1145/3278532.3278543.

[52] Cisco. *Umbrella Popularity List*. URL: https://s3-us-west-1.amazonaws.com/umbrella-static/index.html (visited on July 16, 2024).

[53] Cloudflare. *Cloudflare Radar: Domain Rankings*. URL: https://radar.cloudflare.com/domains (visited on July 16, 2024).

[54] Cloudflare Research. *Cloudflare Research: Post-Quantum Key Agreement*. 2023. URL: https://pq.cloudflareresearch.com/ (visited on Oct. 24, 2024).

[55] Hugh Collins and Chris Nay. *IBM Unveils 400 Qubit-Plus Quantum Processor and Next-Generation IBM Quantum System Two*. URL: https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two (visited on Oct. 24, 2024).

[56] Scott A. Crosby and Dan S. Wallach. "Denial of Service via Algorithmic Complexity Attacks". In: *Proc. USENIX Security Symposium (USENIX Security)*. Washington, D.C., 2003.

[57] Samir Dawaliby, Abbas Bradai, and Yannis Pousset. "In depth performance evaluation of LTE-M for M2M communications". In: *Proc. IEEE Int. Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. New York, NY, USA, 2016. DOI: 10.1109/WiMOB.2016.7763264.

[58] Alban Diquet. *SSLyze*. URL: https://github.com/nabla-c0d3/sslyze (visited on July 5, 2024).

[59] David Dittrich and Erin Kenneally. "The Menlo Report: Ethical principles guiding information and communication technology research". In: *US Department of Homeland Security* (2012).

[60] *Domain names - concepts and facilities*. RFC 1034. 1987. DOI: 10.17487/RFC1034.

[61] Zakir Durumeric, Michael Bailey, and J. Alex Halderman. "An Internet-Wide View of Internet-Wide Scanning". In: *Proc. USENIX Security Symposium (USENIX Security)*. San Diego, CA, 2014.

[62] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. "Analysis of the HTTPS certificate ecosystem". In: *Proc. ACM Internet Measurement Conference (IMC)*. Barcelona, Spain, 2013. DOI: 10.1145/2504730.2504755.

[63] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. "ZMap: Fast Internet-wide Scanning and Its Security Applications". In: *Proc. USENIX Security Symposium (USENIX Security)*. Washington, D.C., USA, 2013.

[64] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. "MoonGen: A Scriptable High-Speed Packet Generator". In: *Proc. ACM Internet Measurement Conference (IMC)*. Tokyo, Japan, 2015. DOI: 10.1145/2815675.2815692.

[65] N. Etemadi. "An elementary proof of the strong law of large numbers". In: *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete* (1981). DOI: 10.1007/BF01013465.

[66] Roy T. Fielding, Mark Nottingham, and Julian Reschke. *HTTP Caching*. RFC 9111. 2022. DOI: 10.17487/RFC9111.

[67] Roy Thomas Fielding. “Architectural styles and the design of network-based software architectures”. PhD thesis. University of California, 2000.

[68] Gil Fink and Ido Flatow. “Introducing Single Page Applications”. In: *Pro Single Page Application Development*. 2014. DOI: 10.1007/978-1-4302-6674-7_1.

[69] Romain Fontugne. *Understanding the Japanese Internet with the Internet Yellow Pages*. 3, 2023. URL: https://blog.apnic.net/2023/09/06/understanding-the-japanese-internet-with-the-internet-yellow-pages/ (visited on Oct. 22, 2024).

[70] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Signature Series (Fowler). Pearson Education, 2012.

[71] Sebastian Gallenmüller, Dominik Scholz, Henning Stubbe, and Georg Carle. “The pos Framework: A Methodology and Toolchain for Reproducible Network Experiments”. In: *Proc. ACM Int. Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. Virtual Event, 2021. DOI: 10.1145/3485983.3494841.

[72] Oliver Gasser, Quirin Scheitle, Pawel Foremski, Qasim Lone, Maciej Korczyński, Stephen D. Strowes, Luuk Hendriks, and Georg Carle. “Clusters in the Expanse: Understanding and Unbiasing IPv6 Hitlists”. In: *Proc. ACM Internet Measurement Conference (IMC)*. Boston, MA, USA, 2018. DOI: 10.1145/3278532.3278564.

[73] Oliver Gasser, Markus Sosnowski, Patrick Sattler, and Johannes Zirngibl. *TUM goscanner*. DOI: 10.5281/zenodo.11243061. URL: https://github.com/tumi8/goscanner.

[74] Felix Gessert. “Low Latency for Cloud Data Management”. PhD thesis. University of Hamburg, 2018.

[75] Google. *Chrome User Experience Report*. URL: https://developer.chrome.com/docs/crux (visited on July 16, 2024).

[76] Google. *Safe Browsing - Making the world’s information safely accessible*. URL: https://safebrowsing.google.com (visited on July 16, 2024).

[77] Lov K. Grover. “A Fast Quantum Mechanical Algorithm for Database Search”. In: *Proc. Annual ACM Symposium on Theory of Computing (STOC)*. Philadelphia, PA, USA, 1996. DOI: 10.1145/237814.237866.

[78] Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. “Combating Web Spam with Trustrank”. In: *Proc. Int. Conference on Very Large Data Bases (VLDB)*. Toronto, Canada, 2004.

[79] Dick Hardt. *The OAuth 2.0 Authorization Framework*. RFC 6749. 2012. DOI: 10.17487/RFC6749.

[80] Ralph Holz, Jens Hiller, Johanna Amann, Abbas Razaghpanah, Thomas Jost, Narseo Vallina-Rodriguez, and Oliver Hohlfeld. “Tracking the deployment of TLS 1.3 on the web: a story of experimentation and centralization”. In: *ACM SIGCOMM Computer Communication Review* (2020). DOI: 10.1145/3411740.3411742.

[81] Binbing Hou and Feng Chen. “GDS-LC: A Latency- and Cost-Aware Client Caching Scheme for Cloud Storage”. In: *ACM Transactions on Storage (TOS)* (2017). DOI: 10.1145/3149374.

[82] Timothy Hunter and Joseph Bradley. *GraphFrames: DataFrame-based Graphs*. URL: https://github.com/graphframes/graphframes (visited on July 16, 2024).

[83] Martin Husák, Milan Cermák, Tomá Jirsík, and Pavel Celeda. “Network-Based HTTPS Client Identification Using SSL/TLS Fingerprinting”. In: *Proc. Int. Conference on Availability, Reliability and Security (ARES)*. Toulouse, France, 2015. DOI: 10.1109/ARES.2015.35.

[84] IANA. *Transport Layer Security (TLS) Extensions*. URL: https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml (visited on July 5, 2024).

[85] IANA. *Transport Layer Security (TLS) Parameters*. URL: https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml (visited on July 5, 2024).

[86] Arun Iyengar and Jim Challenger. “Improving Web Server Performance by Caching Dynamic Data”. In: *Proc. USENIX Symposium on Internet Technologies and Systems (USITS)*. 1997.

[87] Michael B. Jones, John Bradley, and Nat Sakimura. *JSON Web Token (JWT)*. RFC 7519. 2015. DOI: 10.17487/RFC7519.

[88] Yehuda Katz, Dan Gebhardt, Gabe Sullice, Jeldrik Hanschke, Tyler Kellen, Steve Klabnik, and Ethan Resnick. *JSON:API*. 2022. URL: https://jsonapi.org/format/1.1/ (visited on Dec. 16, 2024).

[89] Yuta Kazato, Yoshihide Nakagawa, and Yuichi Nakatani. “Improving Maliciousness Estimation of Indicator of Compromise Using Graph Convolutional Networks”. In: *Proc. IEEE Annual Consumer Communications & Networking Conference (CCNC)*. Las Vegas, NV, USA, 2020. DOI: 10.1109/CCNC46108.2020.9045113.

[90] Marcel Kempf, Nikolas Gauder, Benedikt Jaeger, Johannes Zirngibl, and Georg Carle. “A Quantum of QUIC: Dissecting Cryptography with Post-Quantum Insights”. In: *Proc. IFIP Networking Conference (IFIP Networking)*. 2024. DOI: 10.23919/IFIPNetworking62109.2024.10619916.

[91] Martin Kleppmann and Alastair R. Beresford. "A Conflict-Free Replicated JSON Datatype". In: *IEEE Transactions on Parallel and Distributed Systems* (2017). DOI: 10.1109/TPDS.2017.2697382.

[92] Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G. Paterson, Narseo Vallina-Rodriguez, and Juan Caballero. "Coming of Age: A Longitudinal Study of TLS Deployment". In: *Proc. ACM Internet Measurement Conference (IMC)*. Boston, MA, USA, 2018. DOI: 10.1145/3278532.3278568.

[93] Alex Krivit, Tim Kornhammar, and Connor Harwood. *Instant Purge: invalidating cached content in under 150ms*. 2024. URL: https://blog.cloudflare.com/instant-purge/ (visited on Dec. 19, 2024).

[94] Craig Labovitz. *Internet Traffic 2009-2019*. 10, 2019. URL: https://nanog.org/news-stories/nanog-tv/working-home/internet-traffic-2009-2019/ (visited on July 5, 2024).

[95] James Larisch, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. "CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers". In: *Proc. IEEE Symposium on Security and Privacy (SP)*. San Jose, CA, USA, 2017. DOI: 10.1109/SP.2017.17.

[96] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation". In: *Proc. Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, USA, 2019. DOI: 10.14722/ndss.2019.23386.

[97] Zane Ma, Joshua Mason, Manos Antonakakis, Zakir Durumeric, and Michael Bailey. "What's in a name? Exploring CA certificate control". In: *Proc. USENIX Security Symposium (USENIX Security)*. 2021.

[98] Majestic. *The Majestic Million*. URL: https://majestic.com/reports/majestic-million/ (visited on July 16, 2024).

[99] Šimon Mandlík, Tomáš Pevnỳ, Václav Šmídl, and Lukáš Bajer. "Malicious Internet Entity Detection Using Local Graph Inference". In: *IEEE Transactions on Information Forensics and Security* (2024). DOI: 10.1109/TIFS.2024.3360867.

[100] Frank Manola and Eric Miller. "RDF Primer". In: *W3C recommendation* (2004). URL: https://www.w3.org/TR/rdf-primer/ (visited on July 16, 2024).

[101] Dominik Marchsreiter and Johanna Sepúlveda. "Hybrid Post-Quantum Enhanced TLS 1.3 on Embedded Devices". In: *Proc. Euromicro Conference on Digital System Design (DSD)*. Maspalomas, Spain, 2022. DOI: 10.1109/DSD57027.2022.00127.

[102] Gonzalo Martinez, Jose Alberto Hernandez, Pedro Reviriego, and Paul Reinheimer. “Round Trip Time (RTT) Delay in the Internet: Analysis and Trends”. In: *IEEE Network* (2023). DOI: 10.1109/MNET004.2300008.

[103] Wilfried Mayer and Martin Schmiedecker. “Turning Active TLS Scanning to Eleven”. In: *Proc. ICT Systems Security and Privacy Protection*. Rome, Italy, 2017. DOI: 10.1007/978-3-319-58469-0_1.

[104] David McGrew, Brandon Enright, Blake Anderson, et al. *Mercury: network metadata capture and analysis*. URL: https://github.com/cisco/mercury (visited on Oct. 14, 2024).

[105] Alexey Melnikov and Ian Fette. *The WebSocket Protocol*. RFC 6455. 2011. DOI: 10.17487/RFC6455.

[106] Mininet Project Contributors. *Mininet: An Instant Virtual Network on your Laptop (or other PC)*. 2022. URL: https://mininet.org/ (visited on Dec. 16, 2024).

[107] Bodo Moeller and Adam Langley. *TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks*. RFC 7507. 2015. DOI: 10.17487/RFC7507.

[108] Richard Moore. *TLS Prober - An SSL/TLS Server Fingerprinting Tool*. URL: https://github.com/WestpointLtd/tls_prober (visited on July 5, 2024).

[109] Mozilla. *Common CA Database*. URL: https://www.ccadb.org/ (visited on Sept. 6, 2024).

[110] Mozilla. *Mozilla Root Store Policy, Version 2.8*. URL: https://github.com/mozilla/pkipolicy/blob/2.8/rootstore/policy.md (visited on Aug. 29, 2024).

[111] Mozilla. *SSL Configuration Generator*. URL: https://ssl-config.mozilla.org (visited on July 12, 2024).

[112] Pejman Najafi, Alexander Mühle, Wenzel Pünter, Feng Cheng, and Christoph Meinel. “MalRank: A Measure of Maliciousness in SIEM-Based Knowledge Graphs”. In: *Proc. ACM Annual Computer Security Applications Conference (ACSAC)*. San Juan, Puerto Rico, USA, 2019. DOI: 10.1145/3359789.3359791.

[113] National Institute of Standards and Technology. “Module-Lattice-Based Digital Signature Standard”. In: *Federal Information Processing Standards Publication (FIPS)* (2024). DOI: 10.6028/NIST.FIPS.204.

[114] National Institute of Standards and Technology. “Module-Lattice-Based Key-Encapsulation Mechanism Standard”. In: *Federal Information Processing Standards Publication (FIPS)* (2024). DOI: 10.6028/NIST.FIPS.203.

[115] National Institute of Standards and Technology. “Stateless Hash-Based Digital Signature Standard”. In: *Federal Information Processing Standards Publication (FIPS)* (2024). DOI: 10.6028/NIST.FIPS.205.

[116] Marcin Nawrocki, Pouyan Fotouhi Tehrani, Raphael Hiesgen, Jonas Mücke, Thomas C. Schmidt, and Matthias Wählisch. “On the interplay between TLS certificates and QUIC performance”. In: *Proc. ACM Int. Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. Rome, Italy, 2022. DOI: 10.1145/3555050.3569123.

[117] Neo4j, Inc. *Neo4j Graph Database & Analytics | Graph Database Management System*. URL: https://neo4j.com/ (visited on July 16, 2024).

[118] Henrik Nielsen, Roy T. Fielding, and Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.0*. RFC 1945. 1996. DOI: 10.17487/RFC1945.

[119] Anoop George Ninan, Purushottam Kulkarni, Prashant Shenoy, Krithi Ramamritham, and Renu Tewari. “Scalable Consistency Maintenance in Content Distribution Networks Using Cooperative Leases”. In: *IEEE Transactions on Knowledge and Data Engineering* (2003). DOI: 10.1109/TKDE.2003.1209001.

[120] NLnet Labs. *Unbound*. URL: https://www.nlnetlabs.nl/projects/unbound (visited on July 5, 2024).

[121] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. “The Akamai Network: A Platform for High-Performance Internet Applications”. In: *SIGOPS Operating Systems Review* (2010). DOI: 10.1145/1842733.1842736.

[122] OpenPhish. *OpenPhish - Phishing Intelligence*. URL: https://openphish.com/ (visited on July 16, 2024).

[123] Mike Ounsworth, John Gray, Massimiliano Pala, Jan Klaußner, and Scott Fluhrer. *Composite ML-DSA For use in X.509 Public Key Infrastructure and CMS*. Internet-Draft. Work in Progress. Internet Engineering Task Force, 2024. URL: https://datatracker.ietf.org/doc/draft-ietf-lamps-pq-composite-sigs/03/.

[124] Eva Papadogiannaki and Sotiris Ioannidis. “Pump Up the JARM: Studying the Evolution of Botnets Using Active TLS Fingerprinting”. In: *Proc. IEEE Symposium on Computers and Communications (ISCC)*. Paris, France, 2023. DOI: 10.1109/ISCC58397.2023.10218210.

[125] Christian Paquin, Douglas Stebila, and Goutam Tamvada. “Benchmarking Post-quantum Cryptography in TLS”. In: *Proc. Int. Conference on Post-Quantum Cryptography (PQCrypto)*. Paris, France, 2020. DOI: 10.1007/978-3-030-44223-1_5.

[126] Craig Partridge and Mark Allman. “Ethical considerations in network measurement papers”. In: *Communications of the ACM* (2016). DOI: 10.1145/2896816.

[127] Sebastian Paul, Yulia Kuzovkova, Norman Lahr, and Ruben Niederhagen. “Mixed Certificate Chains for the Transition to Post-Quantum Authentication

in TLS 1.3". In: *Proc. ACM Asia Conference on Computer and Communications Security (ASIA CCS)*. Nagasaki, Japan, 2022. DOI: 10.1145/3488932.3497755.

[128] Thierry Renaux, Sam Van den Vonder, and Wolfgang De Meuter. "Secure RDTs: Enforcing Access Control Policies for Offline Available JSON Data". In: (2023). DOI: 10.1145/3622802.

[129] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. 2018. DOI: 10.17487/RFC8446.

[130] Eric Rescorla and Tim Dierks. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. 2008. DOI: 10.17487/RFC5246.

[131] Arik Rinberg, Tomer Solomon, Roee Shlomo, Guy Khazma, Gal Lushi, Idit Keidar, and Paula Ta-Shma. "DSON: JSON CRDT Using Delta-Mutations For Document Stores". In: *Proc. Int. Conference on Very Large Data Bases (VLDB)*. Sydney, Australia, 2022. DOI: 10.14778/3510397.3510403.

[132] Richard Roberts and Dave Levin. "When Certificate Transparency Is Too Transparent: Analyzing Information Leakage in HTTPS Domain Names". In: *Proc. ACM Workshop on Privacy in the Electronic Society (WPES)*. London, United Kingdom, 2019. DOI: 10.1145/3338498.3358655.

[133] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases*. O'Reilly Media, 2013.

[134] Kent Rochford. *Request for Comments on Post-Quantum Cryptography Requirements and Evaluation Criteria*. 2016. URL: https://www.federalregister.gov/d/2016-18150 (visited on July 17, 2024).

[135] Marcin J. Schroeder. "An Alternative to Entropy in the Measurement of Information". In: *Entropy* (2004). DOI: 10.3390/e6050388.

[136] Sean Gallagher. *Nearly half of malware now use TLS to conceal communications*. 21, 2021. URL: https://news.sophos.com/en-us/2021/04/21/nearly-half-of-malware-now-use-tls-to-conceal-communications/ (visited on July 5, 2024).

[137] Claude Elwood Shannon. "A Mathematical Theory of Communication". In: *The Bell System Technical Journal* (1948). DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[138] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. "Conflict-Free Replicated Data Types". In: *Proc. Int. Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. Grenoble, France, 2011. DOI: 10.1007/978-3-642-24550-3_29.

[139] Peter Williston Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *Proc. IEEE Annual Symposium on Foundations of*

Computer Science (SFCS). Santa Fe, NM, USA, 1994. DOI: 10.1109/SFCS.1994.365700.

[140] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. "Assessing the Overhead of Post-Quantum Cryptography in TLS 1.3 and SSH". In: *Proc. ACM Int. Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. Barcelona, Spain, 2020. DOI: 10.1145/3386367.3431305.

[141] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. "Post-Quantum Authentication in TLS 1.3: A Performance Study". In: *Proc. Network and Distributed Systems Security (NDSS) Symposium*. San Diego, CA, USA, 2020. DOI: 10.14722/ndss.2020.24203.

[142] Milivoj Simeonovski, Giancarlo Pellegrino, Christian Rossow, and Michael Backes. "Who Controls the Internet? Analyzing Global Threats using Property Graph Traversals". In: *Proc. ACM Int. Conference on World Wide Web (WWW)*. Perth, Australia, 2017. DOI: 10.1145/3038912.3052587.

[143] Markus Sosnowski, Wiedner Florian, Eric Hauser, Lion Steger, Dimitrios Schoinianakis, Sebastian Gallenmüller, and Georg Carle. *The Performance of Post-Quantum TLS 1.3: Raw Data*. Dataset. Technical University of Munich, 2023. DOI: 10.14459/2023mp1725057.

[144] Markus Sosnowski, Patrick Sattler, Johannes Zirngibl, Tim Betzer, and Georg Carle. *ITEG: Additional Material*. URL: https://tumi8.github.io/iteg/ (visited on July 16, 2024).

[145] Markus Sosnowski, Richard von Seck, Florian Wiedner, and Georg Carle. *CRDT Web Caching: Additional Material*. 2024. URL: https://tumi8.github.io/crdt-web-caching/ (visited on Jan. 13, 2025).

[146] Markus Sosnowski, Richard von Seck, Florian Wiedner, and Georg Carle. *CRDT Web Caching Measurement Data*. Dataset. Zenodo, 2024. DOI: 10.5281/zenodo.13982759.

[147] Markus Sosnowski, Florian Wiedner, Eric Hauser, Lion Steger, Dimitrios Schoinianakis, Sebastian Gallenmüller, and Georg Carle. *PQS TLS Measurements repository*. 2023. DOI: 10.5281/zenodo.10054882. URL: https://github.com/tumi8/pqs-tls-measurements.

[148] Markus Sosnowski, Florian Wiedner, Eric Hauser, Lion Steger, Dimitrios Schoinianakis, Sebastian Gallenmüller, and Georg Carle. *The Performance of Post-Quantum TLS 1.3 Website*. URL: https://tumi8.github.io/pqs-tls-measurements (visited on July 17, 2024).

[149] Markus Sosnowski, Johannes Zirngibl, Patrick Sattler, and Georg Carle. *DissecTLS: Additional Material*. URL: https://tumi8.github.io/dissectls/ (visited on July 10, 2024).

[150] Markus Sosnowski, Johannes Zirngibl, Patrick Sattler, Georg Carle, Claas Grohnfeldt, Michele Russo, and Daniele Sgandurra. *Active TLS Fingerprinting: Additional Material*. URL: https://tumi8.github.io/active-tls-fingerprinting/ (visited on July 5, 2024).

[151] Markus Sosnowski, Johannes Zirngibl, Patrick Sattler, Georg Carle, Claas Grohnfeldt, Michele Russo, and Daniele Sgandurra. *EFACTLS Measurement Data*. Dataset. Technical University of Munich, 2024. DOI: 10.14459/2024mp1733820.

[152] Bruce Spang. *Building a Fast and Reliable Purging System*. 2014. URL: https://www.fastly.com/blog/building-fast-and-reliable-purging-system (visited on Dec. 16, 2024).

[153] National Institute of Standards and Technology. *NIST Announces 14 Candidates to Advance to the Second Round of the Additional Digital Signatures for the Post-Quantum Cryptography Standardization Process*. 2024. URL: https://csrc.nist.gov/news/2024/pqc-digital-signature-second-round-announcement (visited on Nov. 7, 2024).

[154] National Institute of Standards and Technology. *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. 2016. URL: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf (visited on July 17, 2024).

[155] Douglas Stebila, Scott Fluhrer, and Shay Gueron. *Hybrid key exchange in TLS 1.3*. Internet-Draft. Work in Progress. Internet Engineering Task Force, 2024. 24 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/11/.

[156] Stephen Hemminger. *tc-netem(8) — Linux manual page*. URL: https://man7.org/linux/man-pages/man8/tc-netem.8.html (visited on July 17, 2024).

[157] Max Stoiber. *The Hardest Part of Caching: Understanding What to Cache and When to Invalidate*. 2024. URL: https://stellate.co/blog/the-hardest-part-of-caching (visited on Dec. 16, 2024).

[158] Nick Sullivan. *High-reliability OCSP stapling and why it matters*. 2017. URL: https://blog.cloudflare.com/high-reliability-ocsp-stapling/ (visited on July 5, 2024).

[159] The Tcpdump Group. *TCPDUMP & LIBPCAP*. 2025. URL: https://www.tcpdump.org (visited on Jan. 13, 2025).

[160] Andreas Theofanous, Eva Papadogiannaki, Alexander Shevtsov, and Sotiris Ioannidis. "Fingerprinting the Shadows: Unmasking Malicious Servers with Machine Learning-Powered TLS Analysis". In: *Proc. ACM Int. Conference on World Wide Web (WWW)*. Singapore, 2024. DOI: 10.1145/3589334.3645719.

[161] Iraklis Tzinos, Konstantinos Limniotis, and Nicholas Kolokotronis. “Evaluating the performance of post-quantum secure algorithms in the TLS protocol”. In: *Journal of Surveillance, Security and Safety* (2022). DOI: 10.20517/jsss.2022.15.

[162] Jelle van Haaster, Rickey Gevers, and Martijn Sprengers. *Cyber Guerilla*. Syngress, 2016.

[163] Benjamin VanderSloot, Johanna Amann, Matthew Bernhard, Zakir Durumeric, Michael Bailey, and J. Alex Halderman. “Towards a Complete View of the Certificate Ecosystem”. In: *Proc. ACM Internet Measurement Conference (IMC)*. Santa Monica, CA, USA, 2016. DOI: 10.1145/2987443.2987462.

[164] Opeoluwa Victor Babasanmi and Josiah Chavula. “Measuring Cloud Latency in Africa”. In: *Proc. Int. Conference on Cloud Networking (CloudNet)*. 2022. DOI: 10.1109/CloudNet55617.2022.9978896.

[165] W3 Schools. *SQL Constraints*. 2025. URL: https://www.w3schools.com/sql/sql_constraints.asp (visited on Jan. 3, 2025).

[166] Gerry Wan, Liz Izhikevich, David Adrian, Katsunari Yoshioka, Ralph Holz, Christian Rossow, and Zakir Durumeric. “On the Origin of Scanning: The Impact of Location on Internet-Wide Scans”. In: *Proc. ACM Internet Measurement Conference (IMC)*. Virtual Event, 2020. DOI: 10.1145/3419394.3424214.

[167] Miao Wan, Arne Jönsson, and Cong Wang. “Web user clustering and Web prefetching using Random Indexing with weight functions”. In: *Knowledge and Information Systems* (2011). DOI: 10.1007/s10115-011-0453-x.

[168] David Warburton. *The 2021 TLS Telemetry Report*. URL: https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report (visited on July 17, 2024).

[169] Bas Westerbaan and Cefan Daniel Rubin. *Defending against future threats: Cloudflare goes post-quantum*. 2022. URL: https://blog.cloudflare.com/post-quantum-for-all/ (visited on Oct. 24, 2024).

[170] Dirk Wetter. *Testing TLS/SSL encryption*. URL: https://testssl.sh (visited on July 5, 2024).

[171] Wolfram Wingerath, Felix Gessert, Erik Witt, Hannes Kuhlmann, Florian Bücklers, Benjamin Wollmer, and Norbert Ritter. “Speed Kit: A Polyglot & GDPR-Compliant Approach For Caching Personalized Content”. In: *Proc. IEEE Int. Conference on Data Engineering (ICDE)*. Dallas, TX, USA, 2020. DOI: 10.1109/ICDE48307.2020.00142.

[172] Dongzhu Xu, Anfu Zhou, Xinyu Zhang, et al. “Understanding Operational 5G: A First Measurement Study on Its Coverage, Performance and Energy Consump-

tion". In: *Proc. ACM SIGCOMM*. Virtual Event, 2020. DOI: 10.1145/3387514.3405882.

[173] Jieyu Zheng, Haoliang Zhu, Yifan Dong, Zhenyu Song, Zhenhao Zhang, Yafang Yang, and Yunlei Zhao. "Faster Post-quantum TLS 1.3 Based on ML-KEM: Implementation and Assessment". In: *Proc. European Symposium on Research in Computer Security (ESORICS)*. Bydgoszcz, Poland, 2024. DOI: 10.1007/978-3-031-70890-9_7.

[174] Di Zhuang and J. Morris Chang. "Enhanced PeerHunter: Detecting Peer-to-Peer Botnets Through Network-Flow Level Community Behavior Analysis". In: *IEEE Transactions on Information Forensics and Security* (2019). DOI: 10.1109/TIFS.2018.2881657.

[175] Johannes Zirngibl, Steffen Deusch, Patrick Sattler, Juliane Aulbach, Georg Carle, and Mattijs Jonker. "Domain Parking: Largely Present, Rarely Considered!" In: *Proc. Network Traffic Measurement and Analysis Conference (TMA)*. 2022.