PQTLS-AD: Post-Quantum TLS Accelerated with DNS

Sangwon Lim
Seoul National University
Seoul, Republic of Korea
sangwonlim@snu.ac.kr

Hyeonmin Lee
University of Virginia
Charlottesville, VA, United States
frv9vh@virginia.edu

Gyeongheon Jeong
Seoul National University
Seoul, Republic of Korea
tiop7@snu.ac.kr

Ted "Taekyoung" Kwon
Seoul National University
Seoul, Republic of Korea
tkkwon@snu.ac.kr

Abstract—**Transport Layer Security (TLS) is expected to transition to Post-Quantum Cryptography (PQC) to mitigate the threats posed by quantum computing. While PQC ensures long-term security, it significantly increases the TLS handshake latency due to the larger key and signature sizes. Various solutions have been proposed to address this issue; however, they suffer from limitations, such as the necessity of pre-fetching certificates or dependency on prior connections. In this paper, we propose PQTLS-AD, a novel method that leverages the Domain Name System (DNS) to efficiently distribute PQC certificates. By offloading certificate transmission to DNS, PQTLS-AD reduces handshake data overhead, effectively lowering handshake latency. We have developed a PQTLS-AD prototype and conducted extensive experiments. Our results show that PQTLS-AD significantly reduces handshake latency, cutting it by more than one Round-Trip Time (RTT) compared to standard PQTLS.**

Index Terms—**Transport Layer Security, TLS, Post-Quantum Cryptography, Post-Quantum TLS, Performance, Latency**

I. Introduction

With growing concerns about security and privacy in online communications, Transport Layer Security (TLS) has become the de facto standard to set up a secure connection between a client and a server [8]. Meanwhile, advances in quantum computing pose a significant threat to current Public Key Cryptography (PKC) standards such as DH, ECDH, RSA, ECDSA, and EdDSA, which have been used by TLS [4]. To counteract this threat, NIST has embarked on the standardization of quantum-secure cryptographic alternatives for key exchange and digital signatures. In August 2024, NIST released FIPS 203 (Module-Lattice-Based Key-Encapsulation Mechanism) [15] based on CRYSTALS-KYBER, FIPS 204 (Module-Lattice-Based Digital Signature Standard) [14] based on CRYSTALS-DILITHIUM, and FIPS 205 (Stateless Hash-Based Digital Signature Standard) [16] based on SPHINCS+.

Many researchers have examined the Post-Quantum Cryptography (PQC) standards to find potential issues, such as computational requirements and data size, when introducing the proposed PQC in TLS. Henceforth, the TLS protocol with the PQC algorithms is referred to as PQTLS (like [24]). In terms of computational overhead, [2] revealed that while processing delays of PQTLS increased slightly when operating as a TLS server on low-end embedded devices but performs well as a client. On modern desktop computers, no noticeable delays were observed when operating as a server or a client [31], [33]. However, since PQC algorithms use much longer keys than traditional PKC ones, the size of the data transmitted during a PQTLS handshake increases substantially, which motivates this work.

Multiple studies indicate that when a PQTLS handshake is performed following a TCP handshake, multiple Round-Trip Times (RTTs) are required to transmit the increased data, due to the constraints imposed by the initial value of the TCP Congestion Window (CW) [2], [26], [29]. Network delays, which can span multiple RTTs, can reach hundreds of milliseconds depending on the geographical distances between clients and servers [8], [13]. In any case, long delays can significantly impact delay-sensitive users [1], [10], [28], [34].

Many studies [2], [12], [19]–[21], [27], [27], [31]–[33] have investigated the issue of increased setup delays caused by the additional data overhead in PQTLS handshakes, mainly due to the volume of the PQC certificate chain. To address this problem, several methods have been proposed, such as caching PQC certificates or prefetching them via alternative channels [6], [7], [22], [25]. However, these approaches have limitations, such as not being applicable during the initial connection or requiring additional channels.

This paper introduces Post-Quantum TLS Accelerated with DNS (PQTLS-AD), a method leveraging the Domain Name System (DNS) to overcome existing limitations. PQTLS-AD enables clients to establish a PQTLS session in a single RTT, even for first-time connections, without requiring additional channels. In contrast, PQTLS typically requires at least two RTTs to transfer the large PQC certificate sizes.

II. Background and Related Work

Domain Name System (DNS). DNS is a globally distributed database that maps domain names to their associated information using DNS records; for instance, an `A` record contains the IPv4 address of a domain name. These records are provided by authoritative name servers and are distributed through DNS resolvers, which are typically located close to clients. Furthermore, since DNS can be used to deliver various information of the server to the client before sending any packets to the server, there have been attempts to leverage it for multiple purposes; TLS Encrypted Client Hello [23] enhances privacy, ZTLS [9] reduces initial connection delay, and DANE [5] augments PKI by publishing certificates via DNS. Accordingly, new DNS

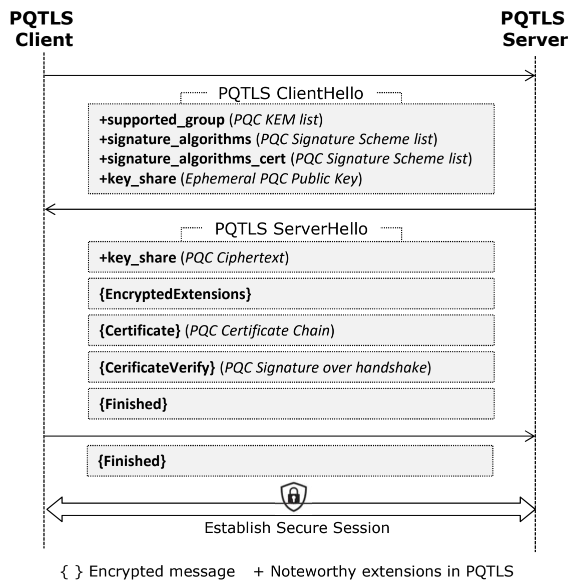

PQTLS Client
PQTLS Server
PQTLS ClientHello
+supported_group (PQC KEM list)
+signature_algorithms (PQC Signature Scheme list)
+signature_algorithms_cert (PQC Signature Scheme list)
+key_share (Ephemeral PQC Public Key)
PQTLS ServerHello
+key_share (PQC Ciphertext)
{EncryptedExtensions}
{Certificate} (PQC Certificate Chain)
{CerificateVerify} (PQC Signature over handshake)
{Finished}
{Finished}
Establish Secure Session
{ } Encrypted message + Noteworthy extensions in PQTLS

Fig. 1: A Post-Quantum TLS Handshake is illustrated.

record types have been proposed, such as the TLSA record [5], which stores the server's certificate or public key.

Post-Quantum TLS (PQTLS). Integrating the PQC schemes into TLS affects several stages of the handshake procedure [26]. Figure 1 illustrates the changes in detail. On the client side, the 'supported_group', which indicates the group for the key exchange supported by the client, is changed from the existing ECDHE or DHE to a PQC Scheme such as FIPS 203. Similarly, the 'signature_algorithms' and 'signature_algorithms_cert' supported by the client are also changed to PQC Schemes. Lastly, a 'key_share' (i.e., an ephemeral PQC public key) of a chosen PQC scheme, (e.g., FIPS 203), is transmitted. On the server side, a PQC ciphertext generated with the PQC public key is transmitted as a 'key_share'. The *Certificate* message contains a certificate chain consisting of certificates that use one of the PQC signature schemes presented by the client. *CertificateVerify* is also generated using one of the post-quantum digital signature schemes presented by the client. The rest of the protocol remains the same as the original TLS protocol.

PQTLS and TCP Congenstion control. PQTLS faces challenges due to the significant increase in the size of key exchange messages and certificate chains, which, when combined with TCP congestion control, leads to performance issues. A common characteristic of PQC algorithms is their reliance on larger key sizes compared to traditional, pre-quantum PKC algorithms [21], [26], [27]. Table I compares the sizes of keys, ciphertexts, and signatures of pre- and post-quantum PKC algorithms. [1]

Related Work. Various approaches have been proposed to address the increased handshake time caused by larger data sizes in PQTLS. Sikeridis et al. [26] suggested increasing the

[1]Due to space constraints, PQC is shown only for security category 3 [17].

TABLE I: Sizes (in bytes) of keys, ciphertexts, and signatures of pre- and post-quantum public key algorithms

Key Agreement Algorithm	Public Key	Ciphertext
ECDH NIST P-256	32	32
DHE 2048	256	256
ML-KEM-768	1184	1088
Digital Signature Algorithm	**Public Key**	**Signature**
ECDSA NIST P-256	32	32
RSA 2048	256	256
ML-DSA-65	1952	3309
SLH-DSA-SHA2/SHAKE-192s	48	16224
SLH-DSA-SHA2/SHAKE-192f	48	35664

TCP initial CW, but this can raise loss rates in congested or low-bandwidth networks and impact all TCP applications. Other studies, such as [24], propose omitting the validation of the server signature using KEM, though certificates are still transmitted, which remain the dominant data component.

[6], [7], [22], [25] explore how to reduce the size of the certificate chains to be delivered by omitting some of them. These methods involve (i) leveraging pre-established certificate dictionaries or sets of intermediate CAs (ICAs), or (ii) caching ICAs to avoid sending redundant certificates during the handshake. Despite their potential advantages, these approaches have limitations: (i) they require some certificates to be prefetched by browsers through an alternative channel, and (ii) they cannot be applied to initial connections.

III. PQTLS-AD Handshake Design

We focus on two goals: reducing PQTLS handshake latency and ensuring backward compatibility.

Reducing PQTLS handshake latency. To connect to a PQTLS server, a PQTLS client typically retrieves the server's IP address from the DNS and completes a TCP handshake before initiating the PQTLS handshake. Our design optimizes this procedure by *concurrently* retrieving the server's certificate chain (i.e., PQTLSAD records described in subsection III-B) from the DNS while fetching the A record (*introducing concurrency* tactic [3]). This allows the client to perform the pre-TLS handshake processes, such as the A record and the PQC certificate retrieval, within the same timeframe as before. As a result, it eliminates (or alleviates) the need to fetch PQC certificates from the server during the PQTLS handshake, achieving a 1-RTT PQTLS handshake.

Ensuring backward compatibility. A PQTLS-AD client can figure out a server's PQTLS-AD support by checking for the presence of PQTLSAD records (associated with the server's domain). If the server does not support PQTLS-AD [2], the client falls back to the standard PQTLS. Similarly, a PQTLS-AD server can verify a client's PQTLS-AD support by examining the *PQTLSAD* extension in *ClientHello*. This approach allows PQTLS-AD to coexist with standard PQTLS.

[2]If the PQTLSAD records are not fetched until the TCP 3-way handshake is finished, the client concludes that the server does not support PQTLS-AD.

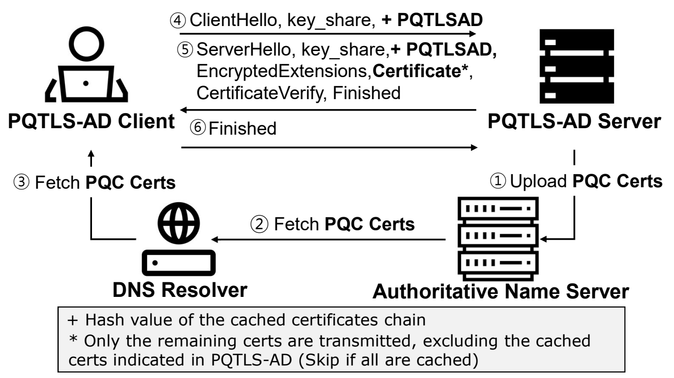

④ ClientHello, key_share, + PQTLSAD
⑤ ServerHello, key_share,+ PQTLSAD, EncryptedExtensions,Certificate*, CertificateVerify, Finished
⑥ Finished
PQTLS-AD Client
PQTLS-AD Server
③ Fetch PQC Certs
① Upload PQC Certs
② Fetch PQC Certs
DNS Resolver
Authoritative Name Server
+ Hash value of the cached certificates chain
* Only the remaining certs are transmitted, excluding the cached certs indicated in PQTLS-AD (Skip if all are cached)

Fig. 2: An overview of PQTLS-AD operations is shown. The PQTLS-AD server omits sending certificates (which have been delivered via DNS) during its handshake.

A. Operational flows

Based on this design choice, the operations of PQTLS-AD are illustrated in Figure 2.

① A PQTLS-AD server uploads some or all of its PQC certificate chain to its authoritative name server as a `PQTLSAD` record.

② A DNS resolver fetches (and caches) the PQC certificates (if a client requests the `PQTLSAD` record) from the authoritative server.

③ A PQTLS-AD client retrieves the PQC certificates from the DNS resolver, in parallel with obtaining the `A` record (i.e., IP addresses).

④ The client generates a hash of the retrieved PQC certificates, and sends the hash in the *PQTLSAD* extension of *ClientHello*. Note that we propose a new *PQTLSAD* extension for *ClientHello* and *ServerHello* (see subsection III-B).

⑤ If the received *ClientHello* contains the *PQTLSAD* extension, the server verifies whether the received hash matches its current certificates. If they match, the server generates the same *PQTLSAD* extension as it received, which is included in *ServerHello*. The *Certificate* sent by the server contains only the PQC certificate(s) absent in the *ClientHello*'s *PQTLSAD* extension.

⑥ The client finalizes the handshake by sending a *Finished* message to the server if no exceptions occur.

B. PQTLS-AD Details

As shown in Figure 2, PQTLS-AD extends both DNS and PQTLS. To facilitate this, we introduce new components: the `PQTLSAD` record (for DNS) and the *PQTLSAD* extension (for TLS).

- **PQTLSAD record (DNS)**: In PQTLS-AD, a server's PQC certificates are delivered via DNS. To enable this, we propose a DNS record type, the `PQTLSAD` record, which stores all or part of a certificate chain (e.g., depending on the server's policy).
- **PQTLSAD extension (TLS)**: We define a new TLS extension for *ClientHello* and *ServerHello*, with 'extension_type' set to *PQTLSAD* (64) and 'extension_data' containing the hash of PQC certificates, which follows the format below; the hash value is computed using SHA-256.[3]

{extension_type=*PQTLSAD*; extension_data=*hash of PQC certificates*}

We next describe the behaviors of PQTLS-AD servers and clients.

PQTLS-AD server. A server performs the following operations.

(i) *Publishing PQTLSAD (DNS)*: to support PQTLS-AD, a server is required to publish some or all of its PQC certificates using a `PQTLSAD` record. The server operator first decides whether to upload the entire certificate chain or only its subset, starting from the root CA certificate. As long as the total amount of data sent by the server (i.e., *ServerHello*, *Certificate*, and *CertificateVerify*) remains within the TCP initial CW size (14,600 bytes)—-for example, when using an ML-DSA-65 certificate derived from DILITHIUM3—-the server operator can choose to upload only a subset of its certificate chain. Specifically, the certificates of the root and intermediate CAs can be published via DNS, while the end-entity certificate of the server is sent directly during the handshake. In such cases, the DNS record only needs to be updated when the root or intermediate CA's certificates change, which usually occurs infrequently. However, if the size of the server's data including only the end-entity certificate exceeds the above CW size, uploading the entire chain to DNS is recommended, which may require relatively frequent updates considering the server's certificate reissuance cycle.

(ii) *Validating ClientHello (TLS)*: when the server receives a handshake request from the client, it determines its operating mode based on the presence of the *PQTLSAD* extension in *ClientHello*. If the extension is included, the server operates in PQTLS-AD mode; otherwise, it falls back to standard PQTLS mode. In PQTLS-AD mode, the server verifies the hash data (referred to as 'extension data' henceforth) in the *PQTLSAD* extension of the *ClientHello* message. This is done by checking whether the hash matches one of the subsets of the currently used certificate chain.

(iii) *Generating ServerHello (TLS)*: if a match is found, the server first generates *ServerHello*, which includes the *PQTLSAD* extension that matches the one included in *ClientHello*. Then, the server generates the *Certificate* message, including only the certificates not present in the *ClientHello*'s *PQTLSAD* extension since the client does not need to receive duplicates. If no match is found, the server generates the hash of its entire certificate chain, includes it in the *PQTLSAD* extension, sends it with *ServerHello*, and transmits the entire certificate chain via *Certificate*, as in standard TLS. This

[3]SHA-256 is used for the same reason as in RFC 7924, as it is already part of the TLS 1.3 cipher suite.

process explicitly informs the client that the cached certificate chain is invalid, ensuring the use of the correct certificates.

PQTLS-AD client. A client performs two operations.

(i) *Validating* `PQTLSAD` *(DNS, TLS)*: the client first checks for the presence of a valid `PQTLSAD` record for the target domain. If the `PQTLSAD` record is not obtained by the time the TCP handshake is completed [4], or if the published record contains certificates with a scheme that the client does not support [5], the client proceeds with a standard TLS handshake following the TCP handshake. If a `PQTLSAD` record exists and is valid, the client initiates a PQTLS-AD handshake with the server over the TCP connection. Here, validating a `PQTLSAD` record involves verifying the certificate chain it contains. This process includes checking the signature, expiration dates, name chaining, path length constraints, and name constraints of all certificates. Following this step, any additional validation is left to the client implementer. [6]

(ii) *Sending ClientHello (TLS)*: during the handshake, the client generates the hash of the certificate chain retrieved from the `PQTLSAD` record, includes it in the *`PQTLSAD`* extension, and sends it to the server via *ClientHello*.

C. Security Considerations

PQTLS-AD differs from existing PQTLS by transmitting (a subset or all) PQC certificates via DNS and omitting those certificates during the handshake, exchanging only their hash value instead.

DNS cache poisoning. Since PQC certificates can be validated through the certificate chain (from the root CA to end-entity), they inherently guarantee authenticity and integrity. Thus, if an attacker manipulates certificates via DNS poisoning, the client can immediately detect it through certificate chain verification.

TLS handshake message manipulation. In our mechanism, certificates retrieved from DNS are omitted from the *Certificate* message sent by the server. As a result, they are also excluded from the handshake transcript where the TLS *Finished* message ensures integrity. However, the client and server exchange the hash of the omitted certificates. This hash is included in the handshake transcript, effectively ensuring the integrity of the omitted certificates. Server authenticity remains guaranteed through *CertificateVerify*, as in standard TLS.

Denial of Service (DoS) attack. A potential attack on PQTLS-AD occurs when a PQTLS-AD server renews its certificates[7]. Typically, when renewing certificates, a server issues new certificates before the previous ones fully expire. Attackers can exploit this transition period by poisoning the DNS cache with *deprecated but still unexpired* certificates, causing a mismatch between the `PQTLSAD` records retrieved by the client and the currently used certificates of the server. This mismatch could render the server inaccessible. However, in such cases, the client can seamlessly fall back to standard TLS by using the certificates provided in the *Certificate* message (see subsection III-B), ensuring no performance degradation compared to standard TLS.

IV. Evaluation

A. Prototype Implementation and Experimental Setup

PQTLS-AD client and server. We implemented the PQTLS-AD client and server using, which is extended with Open Quantum Safe for OpenSSL [18], [30]. Both the client and server run on Ubuntu.[8] The client is located in the AWS Ohio region, while the server is located in Seoul, South Korea, with an average RTT of 247 ms between them. For fast and reliable DNS queries, the client is configured with EDNS0 and keeps the TCP connection open with the DNS resolver.

Authoritative name server and DNS resolver. We deploy an authoritative name server using BIND9 to serve DNS records for our test domain. The name server is co-located with the PQTLS-AD server. As the `PQTLSAD` record is a proposed one and is not officially supported, we slightly abuse the `TLSA` record [5] to store PQC certificates for experimental purposes. For DNS resolution, we utilize a local DNS resolver provided by Amazon, also located in the AWS Ohio region, with a query latency of about 1 ms for `A` record lookups.

B. TLS and DNS Latency Analysis

For experimental purposes, we generate two PQC certificate chains, each of length 2, employing CRYSTALS-DILITHIUM3 and SPHINCS+, respectively. Each certificate chain comprises a self-signed root certificate and an end-entity certificate, the latter being signed by the corresponding root certificate. The root certificates are uploaded as `PQTLSAD` records to our authoritative name server. Consequently, only the end-entity certificates are delivered by the server during the TLS handshake. We utilize CRYSTALS-KYBER768, a PQC scheme, as the key agreement algorithm in all scenarios. For comparative purposes, we additionally conduct experiments with a certificate chain of length two using SHA256ECDSA. In this setup, we measure the TLS handshake latency and DNS query latency of PQTLS and PQTLS-AD, and calculate the median values over 1,000 runs.

TLS handshake latency. As shown in Figure 3a, our experimental results demonstrate that PQTLS-AD effectively reduces handshake latency compared to PQTLS with the same certificate chain. Specifically, with DILITHIUM3 certificates, PQTLS-AD achieves a 1-RTT handshake, reducing 1-RTT compared to PQTLS. With SPHINCS+ certificates, PQTLS-AD reduces nearly 2-RTTs compared to PQTLS.

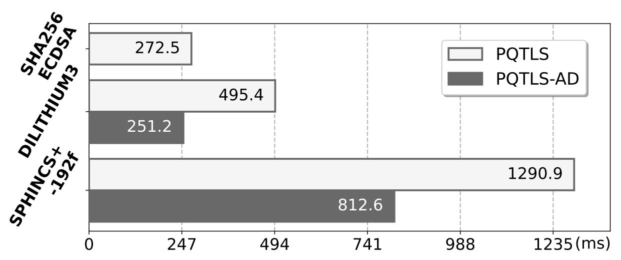
[4]This task runs in a separate thread, typically finishing before the other thread responsible for obtaining the IP address and establishing a TCP session.

[5]Popular browsers support digital signature algorithms endorsed by leading Certificate Authorities (CAs), so PQTLS-AD clients rarely receive certificates using unsupported schemes.

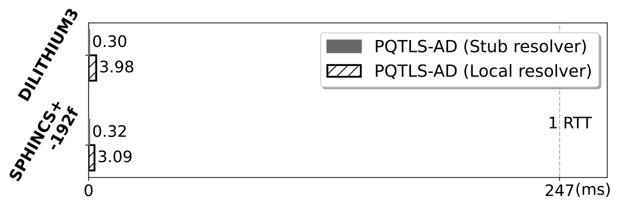
[6]As noted in [11], additional validation considerations–such as performance trade-offs–are outside the scope of PQTLS-AD.

[7]We do not consider attacks that render DNS unusable, as existing TLS clients would also be unable to receive `A` records and establish a connection.

[8]The client operates on Ubuntu 20.04.3 LTS hosted on an AWS EC2 t2.micro instance (Intel® Xeon® CPU E5-2686 v4 @ 2.30GHz with 1 GB RAM). The server runs on Ubuntu 20.04.6 LTS installed on a desktop with an Intel® Core™ i5-7500 CPU @ 3.40GHz and 8 GB RAM.



(a) We plot the handshake latencies of PQTLS and PQTLS-AD. The x-axis grid lines are spaced in 1-RTT (247 ms) intervals to highlight how many RTTs are taken for a handshake between the PQTLS/PQTLS-AD client and server.



(b) `PQTLSAD` record query times are plotted depending on DNS resolver types, with a dashed vertical line marking the 1-RTT threshold.

Fig. 3: TLS and DNS latency comparison.

DNS query latency. Next, we examine the DNS latencies for fetching PQTLS-AD records from the client side. We consider two resolver types that can be used by the client: a *stub* resolver (within the OS) and a *local* DNS resolver. In our experiments, `PQTLSAD` records are assumed to be cached in both resolvers, and we measure the time it takes for the client to fetch these cached records. As shown in Figure 3b, the stub resolver has a query latency of around 0.3 ms for both DILITHIUM3 and SPHINCS+ certificates. Even when the client retrieves DNS records from the local resolver, the query delay remains below 4 ms, which is minimal compared to the TCP handshake time—similar to the 1-RTT (i.e., the dashed vertical line in Figure 3b). Since the DNS queries run concurrently with the TCP handshake, they do not affect the overall time. Therefore, the impact of resolver types on the PQTLS-AD handshake delay is negligible.

C. Transmitted Data Size and Handshake Latency

To understand how our design achieves this reduction, we examine the size of the transmitted data during the handshake.

First, in the experiments, the size of *ClientHello* is 1,462 bytes for PQTLS and 1,474 bytes for PQTLS-AD that includes the *`PQTLSAD`* extension. Since both sizes are below the TCP initial CW size of 14,600 bytes, they can be transmitted within 1-RTT. Therefore, we focus on the TLS handshake messages sent by the server. Table II shows the sizes of the server's three handshake messages (*ServerHello*, *Certificate*, and *CertificateVerify*), along with their total. For SHA256ECDSA, the total size of 2,369 bytes allows the handshake to complete within 1-RTT since it does not exceed 14,600 bytes. Next, in the case of DILITHIUM3 with PQTLS,

TABLE II: Sizes of the server's TLS handshake messages.

DSA	Protocol	TLS handshake message size (bytes)			
		ServerHello	*Certificate*	*CertVerify*	Total
SHA256ECDSA	PQTLS	1,174	1,120	75	***2,369***
DILITHIUM3	PQTLS	1,174	10,934	3,297	***15,405***
	PQTLS-AD	1,186	5,433	3,297	***9,916***
SPHINCS+-192f	PQTLS	1,174	71,829	35,668	***108,671***
	PQTLS-AD	1,186	35,880	35,668	***72,734***

the total size is 15,405 bytes, with *Certificate* (of the server) accounting for 10,934 bytes. Since this exceeds 14,600 bytes, the handshake requires at least two RTTs. In contrast, using DILITHIUM3 with PQTLS-AD allows the server to omit the root CA certificate from *Certificate*, reducing its size to 5,433 bytes. This lowers the total data size to 9,916 bytes, enabling a 1-RTT handshake, which aligns with the 1-RTT gap between PQTLS and PQTLS-AD for DILITHIUM3 in Figure 3a.

For SPHINCS+ with PQTLS, the total data size is 108,671 bytes. According to TCP slow start, the server can initially send up to 14,600 bytes (initial CW). After receiving ACKs (after one RTT), the CW doubles to 29,200 bytes. Following another RTT, the CW increases to 58,400 bytes. Note that, at this point, 102,200 bytes can be transmitted. Thus, even with the assumption of no packet losses, at least 4 RTTs are required to complete the transmission of the server's handshake messages. With PQTLS-AD, the total size is reduced to 72,734 bytes by omitting the CA certificate (35,949 bytes) from *Certificate*. This allows the server to finish transmitting its handshake data over three RTTs, which explains the performance gap in the case of SPHINCS+ in Figure 3a.

D. Summary

Our experimental results indicate that the data sent by the server in PQTLS may exceed the TCP initial CW, leading to an increased handshake delay. This delay may extend across multiple RTTs due to TCP slow start, depending on the data size. However, with PQTLS-AD, certificates can be disseminated via DNS, notably reducing the data to be transmitted during the TLS handshake. Additionally, the latency introduced by delivering certificates through DNS is negligible compared to the handshake latency. In this way, PQTLS-AD effectively reduces the overall handshake time by *more than one RTT* compared to PQTLS.

V. Conclusion

Post-quantum cryptography (PQC) has significantly larger public key and signature sizes compared to traditional public key cryptography (PKC). When the TLS protocol adopts PQC, its handshake latency may be substantially increased, potentially degrading the user experience for delay-sensitive applications. To mitigate this issue, we proposed PQTLS-AD, which reduces the handshake latency by disseminating the server's PQC certificate chain via DNS. Our prototype-based experiments demonstrate that PQTLS-AD effectively decreases the TLS handshake delay by reducing the amount of the certificate chain that must be transmitted during the handshake. Moreover, PQTLS-AD is designed in a backward-compatible fashion, enabling seamless integration with existing TLS servers.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2023-00220985). This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) [NO.RS-2021-II211343, Artificial Intelligence Graduate School Program (Seoul National University)].

References

[1] I. N. Bozkurt, A. Aguirre, B. Chandrasekaran, P. B. Godfrey, G. Laughlin, B. Maggs, and A. Singla. Why is the internet so slow?! In M. A. Kaafar, S. Uhlig, and J. Amann, editors, *Passive and Active Measurement (PAM)*, pages 173–187, Cham, 2017. Springer International Publishing.

[2] K. Bürstinghaus-Steinbach, C. Krauß, R. Niederhagen, and M. Schneider. Post-quantum tls on embedded systems: Integrating and evaluating kyber and sphincs+ with mbed tls. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '20, page 841–852, New York, NY, USA, 2020. Association for Computing Machinery.

[3] D. Garlan, F. Bachmann, J. Ivers, J. Stafford, L. Bass, P. Clements, and P. Merson. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, 2nd edition, 2010.

[4] R. A. Grimes. *How Can Quantum Computing Break Today's Cryptography?*, pages 59–83. Wiley Data and Cybersecurity, 2020.

[5] P. E. Hoffman and J. Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698, Aug. 2012.

[6] P. Kampanakis, C. Bytheway, B. Westerbaan, and M. Thomson. Suppressing CA Certificates in TLS 1.3. Internet-Draft draft-kampanakis-tls-scas-latest-03, Internet Engineering Task Force, Jan. 2023. Work in Progress.

[7] P. Kampanakis and M. Kallitsis. Faster post-quantum tls handshakes without intermediate ca certificates. In S. Dolev, J. Katz, and A. Meisels, editors, *Cyber Security, Cryptology, and Machine Learning*, pages 337–355, Cham, 2022. Springer International Publishing.

[8] H. Lee, D. Kim, and Y. Kwon. Tls 1.3 in practice:how tls 1.3 contributes to the internet. In *Proceedings of the Web Conference 2021*, WWW '21, page 70–79, New York, NY, USA, 2021. Association for Computing Machinery.

[9] S. Lim, H. Lee, H. Kim, H. Lee, and T. Kwon. Ztls: A dns-based approach to zero round trip delay in tls handshake. In *Proceedings of the ACM Web Conference 2023*, WWW '23, page 2360–2370, New York, NY, USA, 2023. Association for Computing Machinery.

[10] S. Lohr. For Impatient Web Users, an Eye Blink Is Just Too Long to Wait. https://www.nytimes.com/2012/03/01/technology/impatient-web-users-flee-slow-loading-sites.html, 2012. Retrieved: 2024-01-04.

[11] M. Luo, B. Feng, L. Lu, E. Kirda, and K. Ren. On the complexity of the web's pki: Evaluating certificate validation of mobile browsers. *IEEE Transactions on Dependable and Secure Computing*, 21(1):419–433, 2024.

[12] D. Marchsreiter and J. Sepúlveda. Hybrid post-quantum enhanced tls 1.3 on embedded devices. In *2022 25th Euromicro Conference on Digital System Design (DSD)*, pages 905–912, 2022.

[13] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste. The cost of the "s" in https. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, page 133–140, New York, NY, USA, 2014. Association for Computing Machinery.

[14] NIST. Module-lattice-based digital signature standard. https://doi.org/10.6028/NIST.FIPS.204, Aug. 2024. Retrieved: 2025-01-03.

[15] NIST. Module-lattice-based key-encapsulation mechanism standard. https://doi.org/10.6028/NIST.FIPS.203, Aug. 2024. Retrieved: 2025-01-03.

[16] NIST. Stateless hash-based digital signature standard. https://doi.org/10.6028/NIST.FIPS.205, Aug. 2024. Retrieved: 2025-01-03.

[17] NIST. Post-Quantum Cryptography Security (Evaluation Criteria). https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/evaluation-criteria/security-(evaluation-criteria), 2025. Retrieved: 2025-01-11.

[18] Open Quantum Safe provider Project Authors. Open quantum safe provider for openssl. https://github.com/open-quantum-safe/oqs-provider, 2024. Retrieved: 2025-01-22.

[19] C. Paquin, D. Stebila, and G. Tamvada. Benchmarking post-quantum cryptography in tls. In J. Ding and J.-P. Tillich, editors, *Post-Quantum Cryptography*, pages 72–91, Cham, 2020. Springer International Publishing.

[20] S. Paul, Y. Kuzovkova, N. Lahr, and R. Niederhagen. Mixed certificate chains for the transition to post-quantum authentication in tls 1.3. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '22, page 727–740, New York, NY, USA, 2022. Association for Computing Machinery.

[21] M. Raavi, S. Wuthier, P. Chandramouli, Y. Balytskyi, X. Zhou, and S.-Y. Chang. Security comparisons and performance analyses of post-quantum signature algorithms. In *Applied Cryptography and Network Security: 19th International Conference, ACNS 2021, Kamakura, Japan, June 21–24, 2021, Proceedings, Part II*, page 424–447, Berlin, Heidelberg, 2021. Springer-Verlag.

[22] E. Rescorla, R. Barnes, H. Tschofenig, and B. M. Schwartz. Compact TLS 1.3. Internet-Draft draft-ietf-tls-ctls-10, Internet Engineering Task Force, Apr. 2024. Work in Progress.

[23] E. Rescorla, K. Oku, N. Sullivan, and C. A. Wood. TLS Encrypted Client Hello. Internet-Draft draft-ietf-tls-esni-24, Internet Engineering Task Force, Mar. 2025. Work in Progress.

[24] P. Schwabe, D. Stebila, and T. Wiggers. Post-quantum tls without handshake signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1461–1480, New York, NY, USA, 2020. Association for Computing Machinery.

[25] D. Sikeridis, S. Huntley, D. Ott, and M. Devetsikiotis. Intermediate certificate suppression in post-quantum tls: an approximate membership querying approach. In *Proceedings of the 18th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '22, page 35–42, New York, NY, USA, 2022. Association for Computing Machinery.

[26] D. Sikeridis, P. Kampanakis, and M. Devetsikiotis. Assessing the overhead of post-quantum cryptography in tls 1.3 and ssh. In *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '20, page 149–156, New York, NY, USA, 2020. Association for Computing Machinery.

[27] D. Sikeridis, P. Kampanakis, and M. Devetsikiotis. Post-quantum authentication in TLS 1.3: A performance study. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[28] A. Singla, B. Chandrasekaran, P. B. Godfrey, and B. Maggs. The internet at the speed of light. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, HotNets-XIII, page 1–7, New York, NY, USA, 2014. Association for Computing Machinery.

[29] M. Sosnowski, F. Wiedner, E. Hauser, L. Steger, D. Schoinianakis, S. Gallenmüller, and G. Carle. The performance of post-quantum tls 1.3. In *Companion of the 19th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT 2023, page 19–27, New York, NY, USA, 2023. Association for Computing Machinery.

[30] The OpenSSL Project Authors. Openssl project. https://github.com/openssl/openssl, 2024. Retrieved: 2025-01-22.

[31] I. Tzinos, K. Limniotis, and N. Kolokotronis. Evaluating the performance of post-quantum secure algorithms in the tls protocol. *Journal of Surveillance, Security and Safety*, 3(3), 2022.

[32] B. Westerbaan. Sizing Up Post-Quantum Signatures. https://blog.cloudflare.com/sizing-up-post-quantum-signatures/, 2021. Retrieved: 2024-01-12.

[33] B. Westerbaan and C. D. Rubin. Defending against future threats: Cloudflare goes post-quantum. https://blog.cloudflare.com/post-quantum-for-all/, Oct. 2022. Retrieved: 2025-02-12.

[34] N. Zilberman, M. Grosvenor, D. A. Popescu, N. Manihatty-Bojan, G. Antichi, M. Wójcik, and A. W. Moore. Where has my time gone? In *International Conference on Passive and Active network measurement (PAM)*, pages 201–214. Springer, 2017.