

# Benchmarking Post-Quantum Cryptography in TLS

Christian Paquin<sup>1</sup>, Douglas Stebila<sup>2</sup>, and Goutam Tamvada<sup>2</sup>

<sup>1</sup> Microsoft Research [cpaquin@microsoft.com](mailto:cpaquin@microsoft.com)

<sup>2</sup> University of Waterloo [dstebila@uwaterloo.ca](mailto:dstebila@uwaterloo.ca), [gtamvada@edu.uwaterloo.ca](mailto:gtamvada@edu.uwaterloo.ca)

**Abstract.** Post-quantum cryptographic primitives have a range of trade-offs compared to traditional public key algorithms, either having slower computation or larger public keys and ciphertexts/signatures, or both. While the performance of these algorithms in isolation is easy to measure and has been a focus of optimization techniques, performance in realistic network conditions has been less studied. Google and Cloudflare have reported results from running experiments with post-quantum key exchange algorithms in the Transport Layer Security (TLS) protocol with real users' network traffic. Such experiments are highly realistic, but cannot be replicated without access to Internet-scale infrastructure, and do not allow for isolating the effect of individual network characteristics.

In this work, we develop and make use of a framework for running such experiments in TLS cheaply by *emulating* network conditions using networking features of the Linux kernel. Our testbed allows us to independently control variables such as link latency and packet loss rate, and then examine the impact on TLS connection establishment performance of various post-quantum primitives, specifically hybrid elliptic curve/post-quantum key exchange and post-quantum digital signatures, based on implementations from the Open Quantum Safe project. Among our key results, we observe that packet loss rates above 3–5% start to have a significant impact on post-quantum algorithms that fragment across many packets, such as those based on unstructured lattices. The results from this emulation framework are also complemented by results on the latency of loading entire web pages over TLS in real network conditions, which show that network latency hides most of impact from algorithms with slower computations (such as supersingular isogenies).

**Keywords:** post-quantum key exchange · post-quantum authentication · Transport Layer Security (TLS) · network performance · emulation

## 1 Introduction

Compared to traditional public key algorithms, post-quantum key encapsulation mechanisms (KEMs) and digital signatures schemes have a range of trade-offs, either having slower computation, or larger public keys and ciphertexts / signatures, or both. Measuring the performance of these algorithms in isolation is easy; doing so accurately in the broader context of Internet protocols such as

the Transport Layer Security (TLS) protocol, and under realistic network traffic conditions, is more difficult.

Alongside the development and standardization of post-quantum algorithms in the NIST Post-Quantum Cryptography Standardization project, there have been various efforts to begin preparing the TLS ecosystem for post-quantum cryptography. We can see at least three major lines of work: (draft) specifications of how post-quantum algorithms could be integrated into existing protocol formats and message flows [9, 15, 29, 30, 32, 36]; prototype implementations demonstrating such integrations can be done [6, 7, 8, 13, 16, 17, 26, 27] and whether they would meet existing constraints in protocols and software [10]; and performance evaluations in either basic laboratory network settings [6, 7] or more realistic network settings [8, 13, 16, 18, 19]. This paper focuses on the last of these issues, trying to understand how post-quantum cryptography’s slower computation and larger communication sizes impact performance of TLS.

A line of work starting with initial experiments by Google [8, 18], with follow-up collaborations between Google, Cloudflare, and others [16, 19], has involved Internet companies running experiments to measure the performance of real connections using post-quantum key exchange (combined with traditional elliptic curve Diffie–Hellman, resulting in so-called “hybrid” key exchange), by modifying client browsers and edge servers to support select hybrid key exchange schemes in TLS 1.3. Such experiments are highly realistic, but cannot be replicated without access to commensurate infrastructure, and do not allow for isolating the effect of individual network characteristics: it is neither possible to precisely quantify the effect of just a change in (say) packet loss on a network route on the latency of TLS connection establishment, nor is it possible to (say) increase just the packet loss on a route and analyze the resulting effects.

**Contributions.** In this paper, we develop an experimental framework for measuring the performance of the TLS protocol under a variety of network conditions. Our framework is inspired by the NetMirage [35] and Mininet [20] network emulation software, and uses the Linux kernel’s networking stack to precisely and independently tune characteristics such as link latency and packet loss rate. This allows for emulation of client–server network experiments on a single machine.

Using this framework, we analyze the impact that post-quantum cryptography has on TLS 1.3 handshake completion time (i.e., until application data can be sent), specifically in the context of hybrid post-quantum key exchange using structured and unstructured lattices and supersingular isogenies; and post-quantum authentication using structured lattices and symmetric-based signatures. Our emulated experiments are run at 4 different latencies (emulating round-trip times between real-world data centres), and at packet loss rates ranging from 0–20%.

Some of our key observations from the network emulation experiments measuring TLS handshake completion time are as follows. For the median connection, handshake completion time is significantly impacted by substantially slower algorithms (for example, supersingular isogenies (SIKE p434) has a significant performance floor compared to the faster structured and unstructured lattice algorithms), although this effect disappears at the 95th percentile. For algorithms

with larger messages that result in fragmentation across multiple packets, performance degrades as packet loss rate increases: for example, median connection time for unstructured lattice key exchange (Frodo-640-AES) matches structured lattice performance at 5–10% packet loss, then begins to degrade; at the 95th percentile, this effect is less pronounced until around 15% packet loss. We see similar trends for post-quantum digital signatures, although with degraded performance for larger schemes starting around 3–5% packet loss since a TLS connection includes multiple public keys and signatures in certificates.

We also carry out an experiment across real networks, measuring page load time over TLS using geographically scattered virtual machines communication over the Internet. From this experiment, we observe that, page size or network latency increases, the overhead of slower TLS connection establishment diminishes as a proportion of the overall page load time.

Our key exchange results complement those of Google, Cloudflare, and others [16,19]: they provide a holistic look at how post-quantum key exchange algorithms perform for users on real network connections of whatever characteristic the users happened to have, whereas our results show the independent effect of each network characteristic, and our techniques can be applied without access to commensurate Internet-scale client and edge server infrastructure.

Closely related to our post-quantum signature experiments is the recent work by Kampanakis and Sikeridis [13] on the performance of two structured lattice signatures in TLS 1.3. They measure how handshake time varies with server distance (measured in number of hops) and how handshake time and failure rate varies with throughput. Our experiments complement theirs by measuring the impact of other network characteristics: connection latency and packet loss rates.

**Organization.** In Section 2, we describe how we integrated post-quantum algorithms into TLS. Section 3 describes the network emulation framework, and Section 4 describes the setup for our two experiments (emulated; and over the real Internet, data-centre-to-data-centre). Section 5 presents and discusses results from the two experiments. Section 6 concludes. Additional data appears in the appendix. Code and complete result data for all the experiments can be found at our GitHub repository: <https://github.com/xvzcf/pq-tls-benchmark>.

## 2 Post-quantum Cryptography in TLS

There have been a variety of proposed specifications, implementations, and experiments involving post-quantum cryptography in TLS 1.2 and TLS 1.3.

In the context of TLS 1.2, Schanck, Whyte, and Zhang [30] and Campagna and Crockett [9] submitted Internet-Drafts to the Internet Engineering Task Force (IETF) with proposals for adding post-quantum and hybrid key exchange to TLS 1.2; implementations of these drafts (or ad hoc specifications) in TLS 1.2 include experiments by Google [8] and Amazon [1], in research papers [6,7], as well as the Open Quantum Safe project’s OQS-OpenSSL 1.0.2 [26,33].

For hybrid and post-quantum key exchange in TLS 1.3, there have been Internet-Drafts by Kiefer and Kwiatowski [15], Whyte et al. [36], Schanck and

Stebila [29], and Stebila et al. [32]. Experimental demonstrations include earlier experiments by Google [17, 19], more recent experiments by a team involving Cloudflare, Google, and others [16], as well as the Open Quantum Safe project’s OQS-OpenSSL 1.1.1 [10, 27], a fork of OpenSSL 1.1.1. There has also been some work on experiments involving post-quantum and hybrid authentication in TLS 1.3, including OQS-OpenSSL 1.1.1 [27] and experiments based on it by Kampanakis and Sikeridis [13].

The experiments in this paper are based on the implementation of hybrid key exchange and post-quantum authentication in TLS 1.3 in OQS-OpenSSL 1.1.1. We now describe the mechanisms used in this particular instantiation of post-quantum cryptography in TLS 1.3. For a broader discussion of design choices and issues in engineering post-quantum cryptography in TLS 1.3, see [32].

## 2.1 Hybrid Key Exchange in TLS 1.3

Our experiments focused on hybrid key exchange, based on the perspective that early adopters of post-quantum cryptography may want post-quantum long-term forward secrecy while still using ECDH key exchange either because of a lack of confidence in newer post-quantum assumptions, or due to regulatory compliance.

The primary way to negotiate an ephemeral key in TLS 1.3 [28] is to use elliptic-curve Diffie-Hellman (ECDH). To do so, a client, in its `ClientHello` message, can send a `supported_groups` extension that names its supported elliptic curve groups; the client can then also provide corresponding `keyshares`, which are the public cryptographic values used to initiate key exchange. By defining new “groups” for each post-quantum and hybrid method, this framework can also be used in a straightforward manner to support the use of post-quantum key-exchange algorithms. Mapping these on to key encapsulation mechanisms, the client uses a KEM ephemeral public key as its `keyshare`, and the server encapsulates against the public key and sends the corresponding ciphertext as its `keyshare`. Despite performing ephemeral key exchange, we only use the IND-CCA versions of the post-quantum KEMs.

In the instantiation of hybrid methods in OQS-OpenSSL 1.1.1, the number of algorithms combined are restricted to two at a time, and a “group” identifier is assigned to each such pair; as a result, combinations are negotiated together, rather than individually. Moreover, in such a hybrid method, the public keys and ciphertexts for the hybrid scheme are simply concatenations of the elliptic curve and post-quantum algorithms’ values in the `keyshare` provided by the `ClientHello` and `ServerHello` messages. For computing the shared secret, individual shared secrets are concatenated and used in place of the ECDH shared secret in the TLS 1.3 key schedule. As OpenSSL does not have a generic KEM or key exchange API in its `libcrypto` component, the modified OpenSSL implementation primarily involves changes in OpenSSL’s `ssl` directory, and calls into OpenSSL’s `libcrypto` for the ECDH algorithms and into the Open Quantum Safe project’s `liboqs` for the post-quantum KEMs.

## 2.2 Post-quantum Authentication in TLS 1.3

Our experiments focused on post-quantum-only authentication, rather than hybrid authentication. We made this choice because, with respect to authenticating connection establishment, the argument for a hybrid mode is less clear: authentication only needs to be secure at the time a connection is established (rather than for the lifetime of the data as with confidentiality). Moreover, in TLS 1.3 there is no need for a server to have a hybrid certificate that can be used with both post-quantum-aware and non-post-quantum aware clients, as algorithm negotiation will be complete before the server needs to send its certificate.

In TLS 1.3, public key authentication is done via signatures, and public keys are usually conveyed via X.509 certificates. There are two relevant negotiation mechanisms in TLS 1.3: the `signature_algorithms_cert` extension which is used to negotiate which algorithms are supported for signatures in certificates; and the `signature_algorithms` extension for which algorithms are supported in the protocol itself. Both of these extensions are a list of algorithm identifiers [28].

In the instantiation in OQS-OpenSSL 1.1.1, new algorithm identifiers are added for each post-quantum signature algorithm to be used, and the algorithms themselves are added to OpenSSL’s generic “envelope public key” object (`EVP_PKEY`) in `libcrypto`, which then percolate upwards to the X.509 certificate generation and management and TLS authentication, with relatively few changes required at these higher levels.

## 3 The Network Emulation Framework

To carry out experiments with full control over network characteristics, we rely on features available in Linux to implement a network emulation framework.

The Linux kernel provides the ability to create *network namespaces* [3], which are independent, isolated copies of the kernel’s network stack; each namespace has its own routes, network addresses, firewall rules, ports, and network devices. Network namespaces can thus emulate separate network participants on a single system.

Two namespaces can be linked using pairs of *virtual ethernet* (`veth`) devices [4]: `veth` devices are always created in interconnected pairs, and packets transmitted on one device are immediately received on the other device in the pair. *Outgoing* traffic on these virtual devices can be controlled by the network emulation (`netem`) kernel module [21], which offers the ability to instruct the kernel to apply, among other characteristics, a delay, an independent or correlated packet loss probability, and a rate-limit to all outgoing packets from the device.

To give the link a minimum round trip time of  $x$  ms, `netem` can be used to instruct the kernel to apply on both `veth` devices a delay of  $\frac{x}{2}$  ms to each outgoing packet. Similarly, to give the link a desired packet loss rate  $y\%$ , `netem` can instruct the kernel to drop on both devices outgoing packets with (independent or correlated) probability  $y\%$ . While `netem` can be used to specify other traffic characteristics, such as network jitter or packet duplication, we consider varying

the round-trip time and packet loss probability to be sufficient to model a wide variety of network conditions. If the round-trip time on a link connecting a server and client conveys the geographical distance between them, then, for example, a low packet loss can model a high-quality and/or wired ethernet connection. Moderate to high packet losses can model low-quality connections or congested networks, such as when the server experiences heavy traffic, or when a client connects to a website using a heavily loaded WiFi network.

Tools such as NetMirage [35] and Mininet [20] offer the ability to emulate larger, more sophisticated, and more realistic networks where, for example, namespaces can serve as autonomous systems (AS) that group clients, and packets can be routed within an AS or between two ASes. We carried out our experiments over a single link (client-server topology) with direct control over network characteristics using `netem` to enable us to isolate the effect of individual network characteristics on the performance of post-quantum cryptography in TLS 1.3.

## 4 Experimental Setup

### 4.1 Cryptographic scenarios

We consider the two cryptographic scenarios in TLS 1.3: hybrid key exchange and post-quantum authentication. Table 1 shows the four key exchange algorithms and four signature algorithms used in our experiments.<sup>3</sup> Their integration into TLS 1.3 was as described in Section 2. We used `liboqs` for the implementations of the post-quantum algorithms; `liboqs` takes its implementations directly from teams’ submissions to NIST or via the PQClean project [14]. Tables 3 and 4 in the appendix show public key/ciphertext/signature size and raw performance on the machine used in our network emulation experiments.

For the key exchange scenario, the rest of the algorithms in the TLS connection were as follows: server-to-client authentication was performed using an ECDSA certificate over the NIST P-256 curve using the SHA-384 hash function. For the signature scenario, key exchange was using `ecdh-p256-kyber512_90s`; the hash function used was SHA-384. In both cases, application data was protected using AES-256 in Galois/counter mode, and the certificate chain was root → server, all of which were using the same algorithms.

### 4.2 Emulated network experiment setup

The goal of the emulated network experiments was to measure the time elapsed until completion of the TLS handshake under various network conditions.

<sup>3</sup> Our Internet data-centre-to-data-centre experiment actually included all Level 1 algorithms supported by `liboqs` (additionally `bike111cpa`, `newhope512cca`, `ntru_hps2048509`, and `lightsaber`; and `picnic211fs`) and additionally hybrid with RSA-3072. The network emulation experiments take much longer to run than the Internet experiments, so we did not have time to collect corresponding network emulation results. For parity, in this paper we only present the results obtained using the same algorithms as in the network emulation experiment. The additional data collected can be found on our GitHub repository.

Notation	Hybrid Family	Variant	Implementation	
<i>Key exchange</i>				
<code>ecdh-p256</code>	×	Elliptic-curve	NIST P-256	OpenSSL optimized
<code>ecdh-p256-sike-434</code>	✓	Supersingular isogeny	SIKE p434 [12]	Assembly optimized
<code>ecdh-p256-kyber512_90s</code>	✓	Module LWE	Kyber 90s level 1 [31]	Reference C
<code>ecdh-p256-frodo640aes</code>	✓	Plain LWE	Frodo-640-AES [23]	C with AES-NI
<i>Signatures</i>				
<code>ecdsa-p256</code>	×	Elliptic curve	NIST P-256	OpenSSL optimized
<code>dilithium2</code>	×	Module LWE/SIS	Dilithium2 [22]	Reference C
<code>qtesla-p-i</code>	×	Ring LWE/SIS	qTESLA provable 1 [5]	AVX2 optimized
<code>picnic-l1-fs</code>	×	Symmetric	Picnic-L1-FS [37]	AVX2 optimized

**Table 1.** Key exchange and signature algorithms used in our experiments

Following the procedure in Section 3, we created two network namespaces and connected them using a `veth` pair, one namespace representing a client, and the other a server. In the client namespace, we ran a modified version of OpenSSL’s `s_time` program, which measures TLS performance by making, in a given time period, as many synchronous (TCP) connections as it could to a remote host using TLS; our modified version (which we’ve called `s_timer`), for a given number of repetitions, synchronously establishes a TLS connection using a given post-quantum algorithm, closes the connection as soon as the handshake is complete, and records only the time taken to complete the handshake. In the server namespace, we ran the `nginx` [24] web server, built against OQS-OpenSSL 1.1.1 so that it is post-quantum aware.

We chose 4 round-trip times to model the geographical distance to servers at different locations: the values were chosen to be similar to the round-trip times in the Internet data-centre network experiment (see Section 4.3), but are not exactly the same, partly because `netem` internally converts a given latency to an integral number of kernel packet scheduler “ticks”, which results in a slight (and negligible) accuracy loss. For each round-trip time, the packet loss probability was varied from 0% to 20% (the probability applies to each packet independently). Finally, for each such combination, for each algorithm under test, 40 independent `s_timer` “client” processes were run, each making repeated synchronous connections to 21 `nginx` worker processes, each of which was instructed to handle 1024 connections.

The experiments were run on a Linux (Ubuntu 18.04) Azure D64a v4 virtual machine, which has 64 vCPUs (2.35 GHz AMD EPYCTM 7452 processor, bursting to 3.35 GHz) and 256 GiB of RAM, in order to give each process its “own” core so as to minimize noise from CPU process scheduling and make the client and server processes as independent of each other as possible.

### 4.3 Internet data-centre-to-data-centre experiment setup

The emulated network experiment concerned itself only with handshake times. In practice, the latency of establishing TLS might not be noticeable when compared to the latency of retrieving application data over the connection. Accordingly, we

Virtual machine	Azure region	Round-trip time
Client	East US 2 (Virginia)	–
Server – near	East US (Virginia)	6.193 ms
Server – medium	Central US (Iowa)	30.906 ms
Server – far	North Europe (Ireland)	70.335 ms
Server – worst-case	Australia East (New South Wales)	198.707 ms

**Table 2.** Client and server locations and network characteristics observed for Internet data-centre-to-data-centre experiment; packet loss rates were observed to be 0%

conducted a set of experiments that involved a client cloud VM requesting web pages of different sizes from various server VMs over the Internet, and measured the total time to receive the complete file.

We set up one client VM and four server VMs in various cloud data centres using Azure, ranging from the server being close to the client to the server being on the other side of the planet. Table 2 shows the data centre locations and gives the round-trip times between the client and server.

It should be noted that the RTT times between any two VMs depend on the state of the network between them, which is highly variable; our values in Table 2 are one snapshot. Given that these are data-centre-to-data-centre links, the packet loss on these links is practically zero. The VMs were all Linux (Ubuntu 18.04) Azure D8s virtual machines, which each have 8 vCPUs (either 2.4 GHz Intel Xeon E5-2673 v3 (Haswell) or 2.3 GHz Intel Xeon E5-2673 v4 (Broadwell), depending on provisioning, bursting to 3.5 GHz) and 32 GiB of RAM. The Apache Benchmark (`ab`) tool [2] was installed on the client VM to measure the connection time; it was modified to use TLS 1.3 via OQS-OpenSSL 1.1.1, verify the server certificate, and be post-quantum aware.

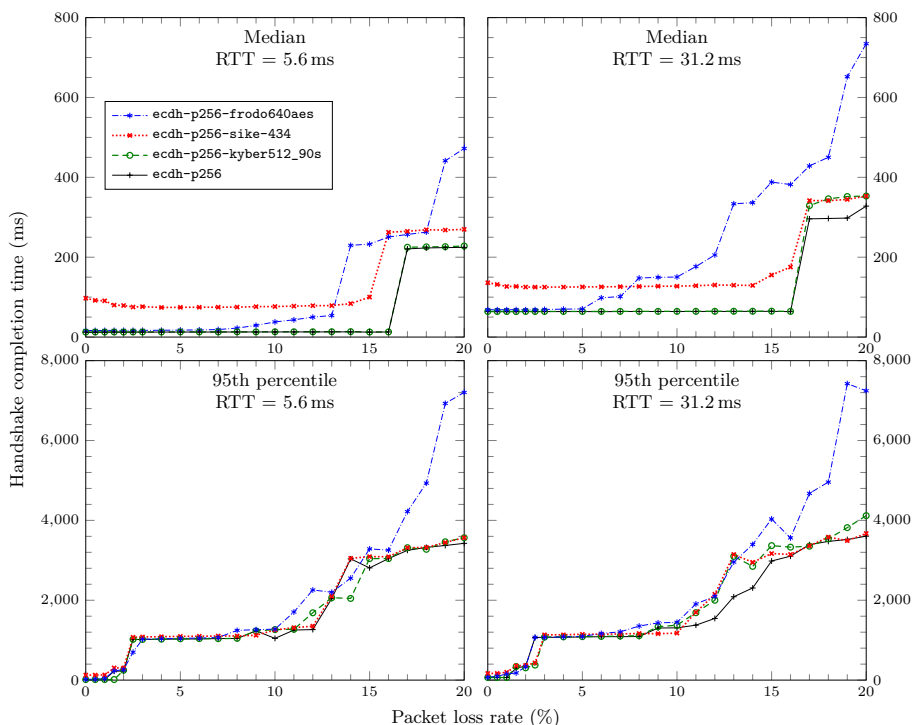
We installed `nginx` (compiled against OQS-OpenSSL 1.1.1) on all server VMs, and we configured it to serve HTML pages of various sizes (1 kB, 10 kB, 100 kB, 1000 kB). (The `http` archive [11] reports that the median desktop and mobile page weight is close to 1950 kB and 1750 kB respectively. Experiments with files as large as 2000 kB took an inordinate amount of time, and all the relevant trends can also be seen at the 1000 kB page size.)

## 5 Results and discussion

### 5.1 Emulated network experiment results

**Key exchange.** Figure 1 shows the results for round-trip times corresponding to “near” and “medium” distance links; Figure 4 in the appendix shows results for longer round-trip times. For each key exchange scenario, we collected between 1500 and 3000 samples (depending on volatility of the data). Even with this many samples, some noise still appears in the results, especially at the 95th percentile, due to high variability inherent in such a noisy process.

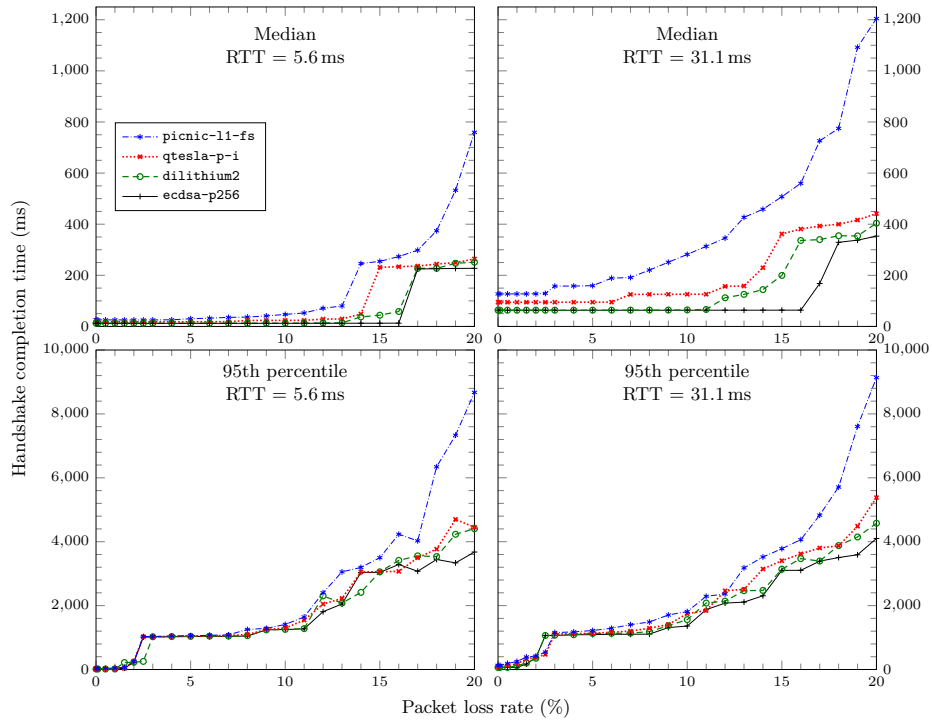




**Fig. 1. Network emulation experiment, key exchange scenario:** handshake completion time (median & 95th percentile) vs. packet loss at shorter round-trip times

At the median, over high quality network links (packet loss rates  $\leq 1\%$ ), we observe that public key and ciphertext size have little impact on handshake completion time, and the predominant factor is cryptographic computation time: ECDH, Kyber512-90s, and Frodo-640-AES have raw cryptographic processing times less than 2 ms resulting in comparable handshake completion times; the slower computation of SIKE p434, where the full cryptographic sequence takes approximately 60 ms, results in a higher latency floor.

As packet loss rates increase, especially above 5%, key exchange mechanisms with larger public keys / ciphertexts, by inducing more packets, bring about longer completion times. For example, at the 31.2 ms RTT, we observe that median Frodo-640-AES completion time starts falling behind. This is to be expected since the maximum transmission unit (MTU) of an ethernet connection is 1500 bytes whereas Frodo-640-AES public key and ciphertext sizes are 9616 bytes and 9720 bytes respectively, resulting in fragmentation across multiple packets. Using the packet analyzer `tcpdump`, we determined that 16 IP packets must be sent by the client to establish a TLS connection using `ecdh-p256-frodo640aes`. If the packet loss probability is  $p = 5\%$ , the probability of at least one packet getting dropped is already  $1 - (1 - p)^{16} \approx 58\%$ , so the median `ecdh-p256-frodo640aes` has required a retransmission. In contrast, only 5 IP packets are required to



**Fig. 2. Network emulation experiment, signature scenario:** Handshake completion time (median & 95th percentile) vs. packet loss at shorter round-trip times

establish a TLS connection with `ecdh-p256` and `ecdh-p256-sike-434`, and 6 packets for `ecdh-p256-kyber512_90s`, which explains why SIKE p434’s small public-key and ciphertext sizes do not offset its computational demand.

At the 95th percentile, we see the impact of raw cryptographic processing times nearly eliminated. Up to 10% packet loss, the performance of the 4 key exchange algorithms are quite close. Past 15% packet loss, the much larger number of packets causes `ecdh-p256-frodo640aes` completion times to spike.

**Digital signatures.** Figure 2 shows the results for round-trip times corresponding to “near” and “medium” distance links; Figure 5 in the appendix shows results for longer round-trip times. For each point, we collected 2500 samples.

The trends here are similar to key exchange, with respect to impact of computation costs and number of packets: at the median, `dilithium2` imposes the least slowdown of all post-quantum signature schemes, and is commensurate with `ecdsa-p256` at low latencies and packet loss rates. `qtesla-p-i` results in a higher latency floor. `picnic-l1-fs`, which produces 34,036-byte signatures, also degrades rapidly as the link latency and packet loss probability increases.

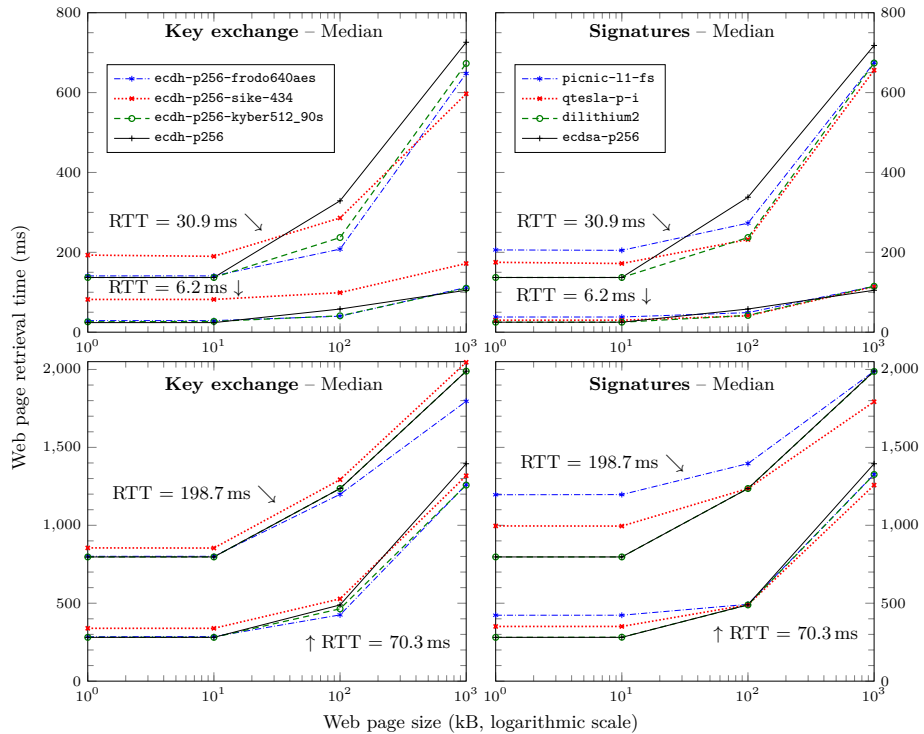
### 5.2 Internet data-centre-to-data-centre experiment results

For each post-quantum scenario, we collected data points by running the `ab` tool for 2 minutes, resulting in between 100 and 500 samples for each scenario.

**Key exchange.** Figure 3 (left) shows the results for median page download times from our four data centres. Figure 6 in the appendix shows results for the 95th percentile; behaviour at the 95th percentile is not too different from median behaviour, likely due to the extremely low packet loss rate on our connections.

For small-RTT connections and small web pages, the TLS handshake constitutes a significant portion of the overall connection time; faster algorithms perform better. As page size and RTT time increase, the handshake becomes less significant. For example, for the near server (US East, 6.2 ms RTT), in comparing `ecdh-p256` with `ecdh-p256-sikep434`, we observe that, at the median, `ecdh-p256` is 3.42 times faster than `ecdh-p256-sikep434` for 1 kB web pages. However this ratio decreases as page sizes increase to 100 or 1000 kB, and as round trip time increases; for example decreasing to  $1.07\times$  and  $1.03\times$  for the worst-case server (Australia, 198.7 ms RTT) at 1 and 1000 KB.

We do observe some variability in the comparisons between algorithms in the Internet experiment in Figure 3 (for example, ECDH and ECDSA in going from



**Fig. 3. Internet data-centre-to-data-centre experiment:** median retrieval time for various web page sizes from 4 data centres; key exchange (left), signatures (right)

being at least as fast as all the PQ hybrids to being slightly slower at 100 and 1000 kB) that we believe may be due to real-world network conditions changing when running different batches sequentially. This effect might be reduced by interweaving batches, which we would like to try in future experimental runs.

**Digital signatures.** Figure 3 (right) shows the results for median round-trip times to the four data centres; Figure 6 in the appendix shows results for the 95th percentile. Just like with the emulated experiment, we observe similar trends between the signature and the key encapsulation mechanisms tests. While the TLS handshake represents a significant portion of the connection establishment time, over increasingly long distances or with increasingly larger payloads, the proportion of time spent on handshake cryptographic processing diminishes.

## 6 Conclusion and Future Work

Our experimental results show under which conditions various characteristics of post-quantum algorithms affect performance. In general, on fast, reliable network links, TLS handshake completion time of the median connection is dominated by the cost of public key cryptography, whereas the 95th percentile completion time is not substantially affected. On unreliable network links with packet loss rates of 3–5% or higher, communication sizes come to govern handshake completion time. As application data sizes grow, the relative cost of TLS handshake establishment diminishes compared to application data transmission.

With respect to the effect of communication sizes, it is clear that the maximum transmission unit (MTU) size imposed by the link layer significantly affects the TLS establishment performance of a scheme. Large MTUs may be able to improve TLS establishment performance for post-quantum primitives with large messages. Some ethernet devices provide (non-standard) support for “jumbo frames”, which are frames sized anywhere from 1500 to 9000 bytes [34]. Since the feature is non-standard, it is not suitable for use in Internet-facing applications, which cannot make assumptions about the link-layer MTUs of other servers/intermediaries, but may help in local or private networks where every link can be accounted for.

Future work obviously includes extending these experiments to cover more algorithms and more security levels; we intend to continue our experiments and will post future results to our repository at <https://github.com/xvzcf/pq-tls-benchmark>. It would be interesting to extend the emulation results to bigger networks that aim to emulate multiple network conditions simultaneously using NetMirage or Mininet. On the topic of post-quantum authentication, our experiments focused on a certificate chain where the root CA and endpoint used the same algorithms (resulting in transmission of one public key and two signatures); it would be interesting to experiment with different chain sizes, and with multi-algorithm chains, perhaps optimized for overall public key + signature size. It would also be possible to measure throughput of a server under load from many clients. Finally, our emulation framework could be applied to investigate other protocols, such as SSH, IPsec, Wireguard, and others.

## Acknowledgements

We would like to thank Eric Crockett for helpful discussions in the early parts of this work. We are grateful to Geovandro C. C. F. Pereira, Justin Tracey, and Nik Unger for their help with the network emulation experiments.

Contributors to the Open Quantum Safe project are listed on the project website [25]. The Open Quantum Safe project has received funding from Amazon Web Services and the Tutte Institute for Mathematics and Computing, and in-kind contributions of developer time from Amazon Web Services, Cisco Systems, evolutionQ, IBM Research, and Microsoft Research. The post-quantum algorithm implementations used in the experiments are directly or indirectly from the original NIST submission teams. Some implementations have been provided by the PQClean project [14].

D.S. is supported in part by Natural Sciences and Engineering Research Council (NSERC) of Canada Discovery grant RGPIN-2016-05146 and a NSERC Discovery Accelerator Supplement. Computation time on Azure was donated by Microsoft Research.

## References

1. Amazon Web Services. s2n. <https://github.com/aws-labs/s2n>, 2014.
2. Apache Software Foundation. ab – Apache HTTP server benchmarking tool, 2019. URL: <https://httpd.apache.org/docs/current/programs/ab.html>.
3. E. W. Biederman. IP-NETNS(8), Jan. 2013. URL: <http://man7.org/linux/man-pages/man8/ip-netns.8.html>.
4. E. W. Biederman and T. Pospíšek. VETH(4), Feb. 2018. URL: <http://man7.org/linux/man-pages/man4/veth.4.html>.
5. N. Bindel, S. Akleyek, E. Alkim, P. S. L. M. Barreto, J. Buchmann, E. Eaton, G. Gutoski, J. Kramer, P. Longa, H. Polat, J. E. Ricardini, and G. Zanon. qTESLA. Technical report, National Institute of Standards and Technology, 2019. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions>.
6. J. W. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 1006–1018. ACM Press, Oct. 2016. doi:10.1145/2976749.2978425.
7. J. W. Bos, C. Costello, M. Naehrig, and D. Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *2015 IEEE Symposium on Security and Privacy*, pages 553–570. IEEE Computer Society Press, May 2015. doi:10.1109/SP.2015.40.
8. M. Braithwaite. Experimenting with post-quantum cryptography, July 2016. URL: <https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html>.
9. M. Campagna and E. Crockett. Hybrid Post-Quantum Key Encapsulation Methods (PQ KEM) for Transport Layer Security 1.2 (TLS). Internet-Draft draft-campagna-tls-bike-sike-hybrid-01, Internet Engineering Task Force, May 2019. Work in Progress. URL: <https://datatracker.ietf.org/doc/html/draft-campagna-tls-bike-sike-hybrid-01>.

10. E. Crockett, C. Paquin, and D. Stebila. Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH. In *NIST 2nd Post-Quantum Cryptography Standardization Conference 2019*, August 2019.
11. http archive. Page weight, Nov. 2019. URL: <https://httparchive.org/reports/page-weight>.
12. D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, L. De Feo, B. Hess, A. Jalali, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, J. Renes, V. Soukharev, D. Urbanik, and G. Pereira. SIKE. Technical report, National Institute of Standards and Technology, 2019. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions>.
13. P. Kampanakis and D. Sikeridis. Two post-quantum signature use-cases: Non-issues, challenges and potential solutions. Cryptology ePrint Archive, Report 2019/1276, 2019. <https://eprint.iacr.org/2019/1276>.
14. M. J. Kannwischer, J. Rijneveld, P. Schwabe, D. Stebila, and T. Wiggers. The PQCclean project, Nov. 2019. URL: <https://github.com/PQCclean/PQCclean>.
15. F. Kiefer and K. Kwiatkowski. Hybrid ECDHE-SIDH key exchange for TLS. Internet-Draft draft-kiefer-tls-ecdh-sidh-00, Internet Engineering Task Force, Nov. 2018. Work in Progress. URL: <https://datatracker.ietf.org/doc/html/draft-kiefer-tls-ecdh-sidh-00>.
16. K. Kwiatkowski, A. Langley, N. Sullivan, D. Levin, A. Mislove, and L. Valenta. Measuring TLS key exchange with post-quantum KEM. In *NIST 2nd Post-Quantum Cryptography Standardization Conference 2019*, Aug. 2019.
17. A. Langley. CECPQ2, Dec. 2018. URL: <https://www.imperialviolet.org/2018/12/12/cecpq2.html>.
18. A. Langley. Post-quantum confidentiality for TLS, Apr. 2018. URL: <https://www.imperialviolet.org/2018/04/11/pqconftls.html>.
19. A. Langley. Real-world measurements of structured-lattices and supersingular isogenies in TLS, Oct. 2019. URL: <https://www.imperialviolet.org/2019/10/30/pqsivssl.html>.
20. B. Lantz, B. Heller, N. Handigol, V. Jeyakumar, B. O'Connor, and C. Burkard. Mininet, Nov. 2019. URL: <http://mininet.org/>.
21. F. Ludovici and H. P. Pfeifer. NETEM(4), Nov. 2011. URL: <http://man7.org/linux/man-pages/man8/tc-netem.8.html>.
22. V. Lyubashevsky, L. Ducas, E. Kiltz, T. Lepoint, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2019. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions>.
23. M. Naehrig, E. Alkim, J. Bos, L. Ducas, K. Easterbrook, B. LaMacchia, P. Longa, I. Mironov, V. Nikolaenko, C. Peikert, A. Raghunathan, and D. Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2019. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions>.
24. NGINX, Inc. NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy, 2019. URL: <https://www.nginx.com/>.
25. Open Quantum Safe Project. Open Quantum Safe, Nov. 2019. URL: <https://openquantumsafe.org/>.
26. Open Quantum Safe Project. OQS-OpenSSL\_1\_0\_2-stable, Nov. 2019. URL: [https://github.com/open-quantum-safe/openssl/tree/OQS-OpenSSL\\_1\\_0\\_2-stable](https://github.com/open-quantum-safe/openssl/tree/OQS-OpenSSL_1_0_2-stable).

27. Open Quantum Safe Project. OQS-OpenSSL\_1\_1\_1-stable, Nov. 2019. URL: [https://github.com/open-quantum-safe/openssl/tree/OQS-OpenSSL\\_1\\_1\\_1-stable](https://github.com/open-quantum-safe/openssl/tree/OQS-OpenSSL_1_1_1-stable).
28. E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, Aug. 2018. URL: <https://rfc-editor.org/rfc/rfc8446.txt>.
29. J. M. Schanck and D. Stebila. A Transport Layer Security (TLS) extension for establishing an additional shared secret. Internet-Draft draft-schanck-tls-additional-keyshare-00, Internet Engineering Task Force, Apr. 2017. Work in Progress. URL: <https://datatracker.ietf.org/doc/html/draft-schanck-tls-additional-keyshare-00>.
30. J. M. Schanck, W. Whyte, and Z. Zhang. Quantum-safe hybrid (QSH) ciphersuite for Transport Layer Security (TLS) version 1.2. Internet-Draft draft-whyte-qsh-tls12-02, Internet Engineering Task Force, July 2016. Work in Progress. URL: <https://datatracker.ietf.org/doc/html/draft-whyte-qsh-tls12-02>.
31. P. Schwabe, R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, G. Seiler, and D. Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2019. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions>.
32. D. Stebila, S. Fluhrer, and S. Gueron. Design issues for hybrid key exchange in TLS 1.3. Internet-Draft draft-stebila-tls-hybrid-design-01, Internet Engineering Task Force, July 2019. Work in Progress. URL: <https://datatracker.ietf.org/doc/html/draft-stebila-tls-hybrid-design-01>.
33. D. Stebila and M. Mosca. Post-quantum key exchange for the internet and the open quantum safe project. In R. Avanzi and H. M. Heys, editors, *SAC 2016*, volume 10532 of *LNCS*, pages 14–37. Springer, Heidelberg, Aug. 2016. doi:10.1007/978-3-319-69453-5\_2.
34. The Ethernet Alliance. Ethernet jumbo frames, Nov. 2009. URL: <http://ethernetalliance.org/wp-content/uploads/2011/10/EA-Ethernet-Jumbo-Frames-v0-1.pdf>.
35. N. Unger, I. Goldberg, Qatar University, and the Qatar Foundation for Education, Science and Community Development. Netmirage, Nov. 2019. URL: <https://crisp.uwaterloo.ca/software/netmirage/>.
36. W. Whyte, Z. Zhang, S. Fluhrer, and O. Garcia-Morchon. Quantum-safe hybrid (QSH) key exchange for Transport Layer Security (TLS) version 1.3. Internet-Draft draft-whyte-qsh-tls13-06, Internet Engineering Task Force, Oct. 2017. Work in Progress. URL: <https://datatracker.ietf.org/doc/html/draft-whyte-qsh-tls13-06>.
37. G. Zaverucha, M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Reiberger, D. Slamanig, J. Katz, X. Wang, and V. Kolesnikov. Picnic. Technical report, National Institute of Standards and Technology, 2019. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions>.

## A Additional Tables and Charts

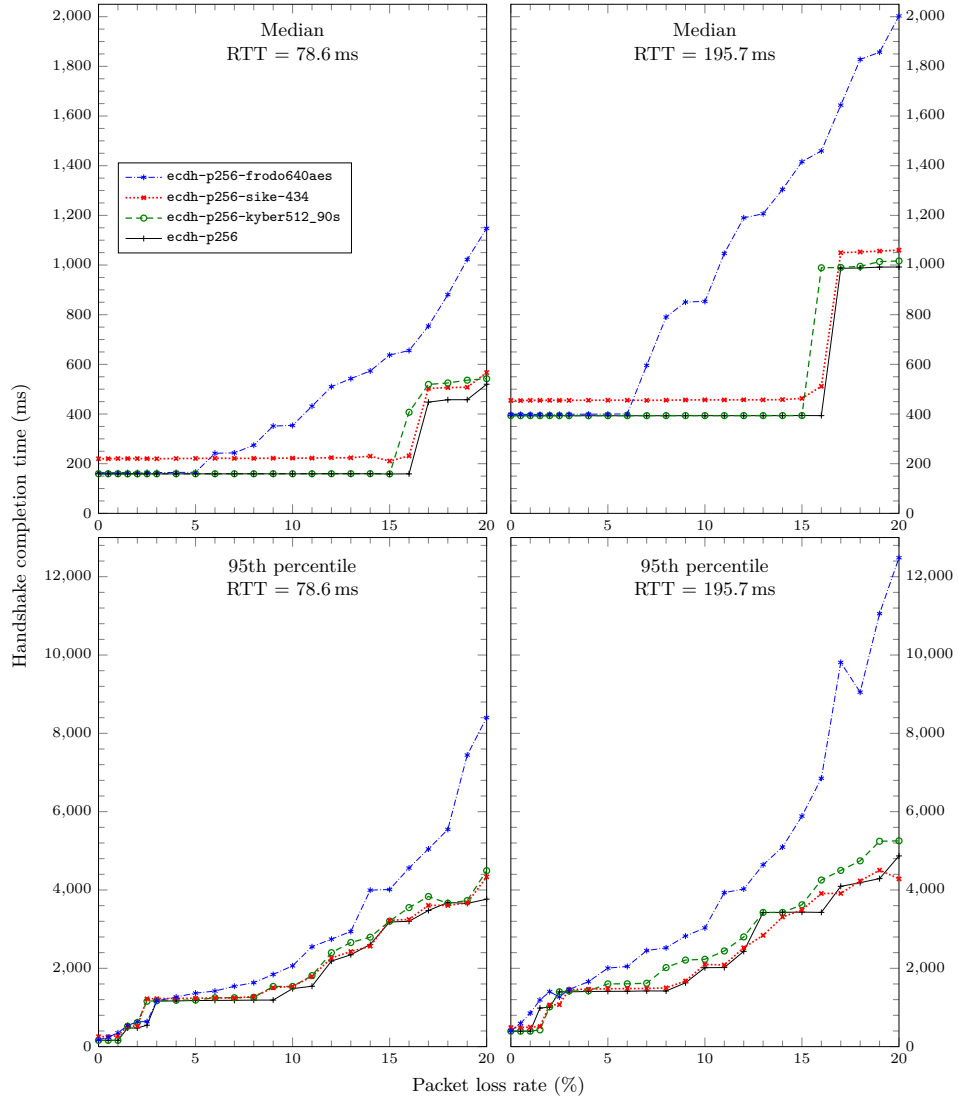
Algorithm	Public key	Ciphertext	Key gen.	Encaps.	Decaps.
ECDH NIST P-256	64	64	0.072	0.072	0.072
SIKE p434	330	346	13.763	22.120	23.734
Kyber512-90s	800	736	0.049	0.063	0.078
FrodoKEM-640-AES	9,616	9,720	1.929	1.048	1.064

**Table 3.** Key exchange algorithm communication size (in bytes) and runtime (in milliseconds)

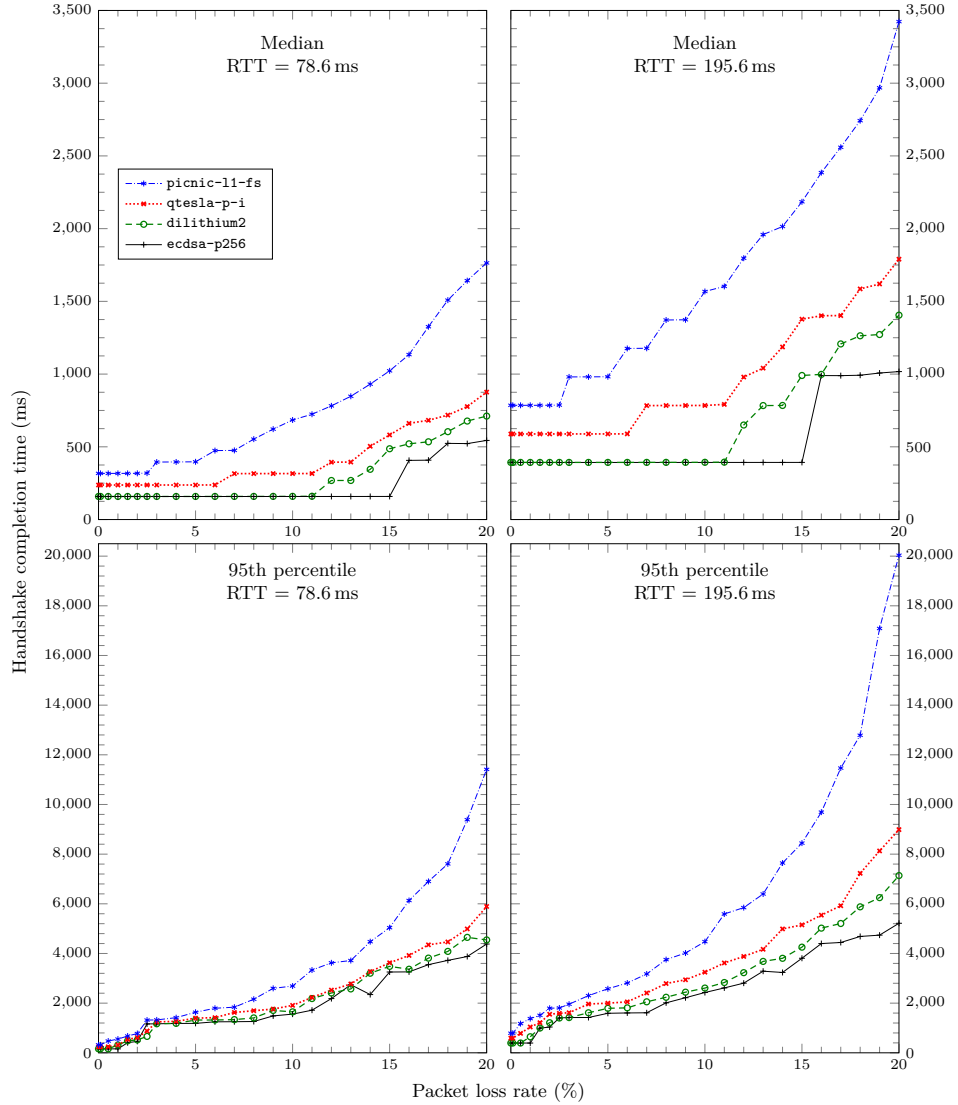
Algorithm	Public key	Signature	Sign	Verify
ECDSA NIST P-256	64	64	0.031	0.096
Dilithium2	1,184	2,044	0.670	0.121
qTESLA-P-I	14,880	2,592	1.055	0.312
Picnic-L1-FS	33	34,036	3.429	2.584

**Table 4.** Signature scheme communication size (in bytes) and runtime (in milliseconds)

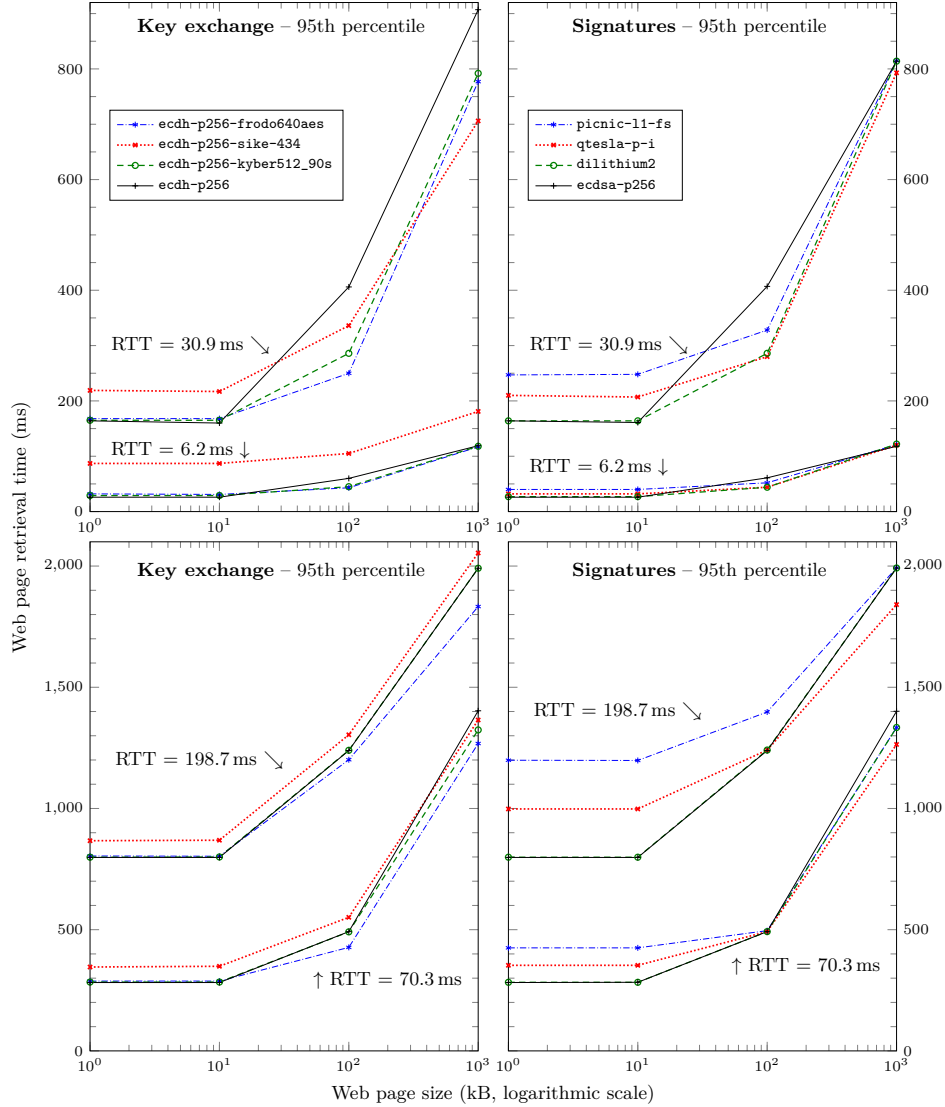




**Fig. 4. Network emulation experiment, key exchange scenario:** handshake completion time (median and 95th percentile) versus packet loss rate at longer round-trip times



**Fig. 5. Network emulation experiment, signature scenario:** Handshake completion time (median and 95th percentile) versus packet loss rate at longer round-trip times



**Fig. 6. Internet data-centre-to-data-centre experiment:** 95th percentile retrieval time for various web page sizes from four data centres; key exchange scenario (left), signature scenario (right)