Post-quantum TLS

Douglas Stebila

UNIVERSITY OF WATERLOO

NSERC CRSNG

UNIVERSITY OF WATERLOO

IQC Institute for Quantum Computing

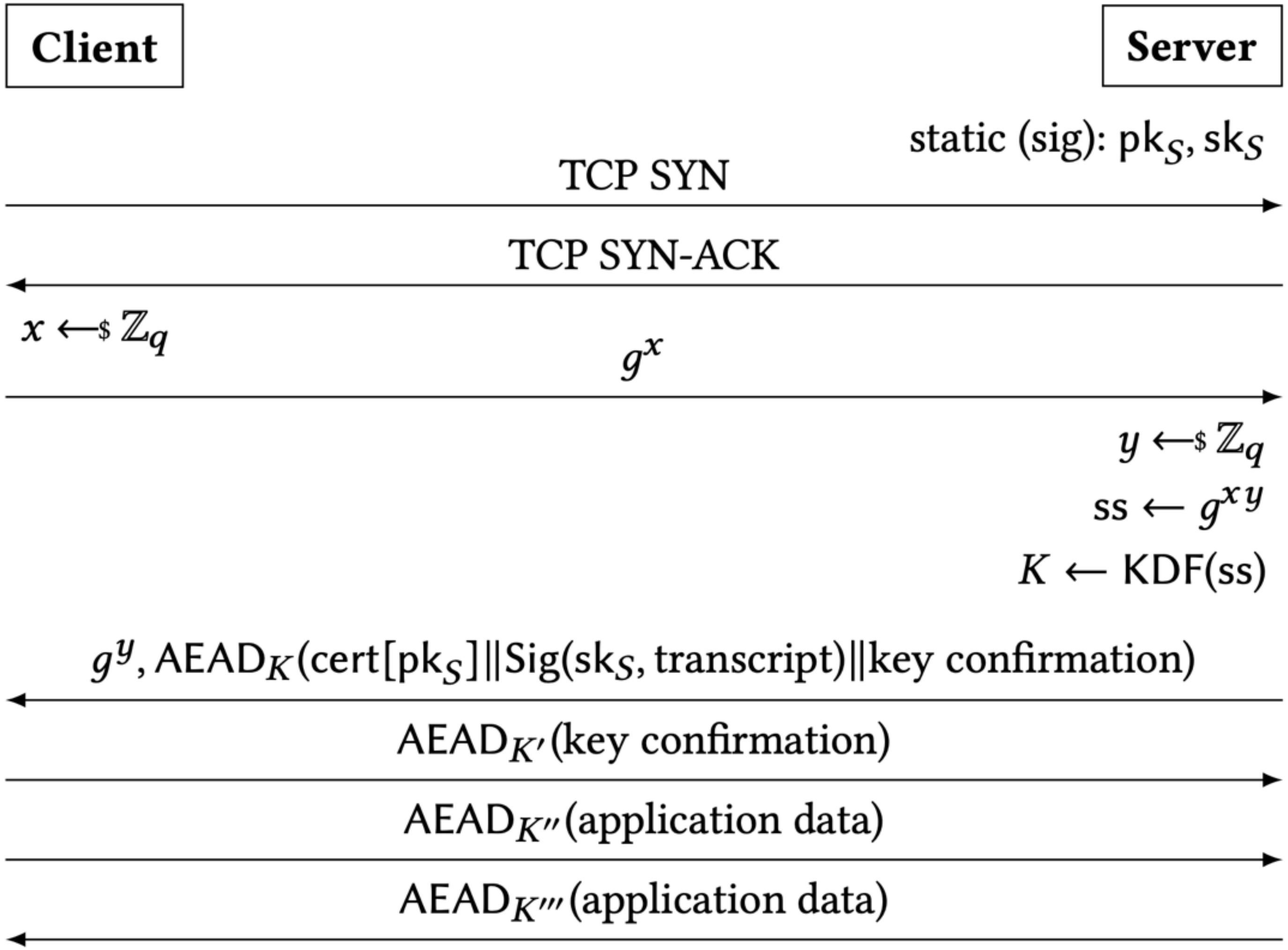
CYBER SECURITY AND PRIVACY INSTITUTE
UNIVERSITY OF WATERLOO

Post-quantum crypto @ University of Waterloo

- UW involved in 4 NIST Round 3 submissions:
  - Finalists: CRYSTALS-Kyber, NTRU
  - Alternates: FrodoKEM, SIKE
- Isogeny-based crypto led by David Jao
- Quantum cryptanalysis led by Michele Mosca
- Post-quantum protocols and implementations (Open Quantum Safe project) led by Douglas Stebila
- + quantum key distribution, quantum computing, privacy and security, …

TLS 1.3 handshake

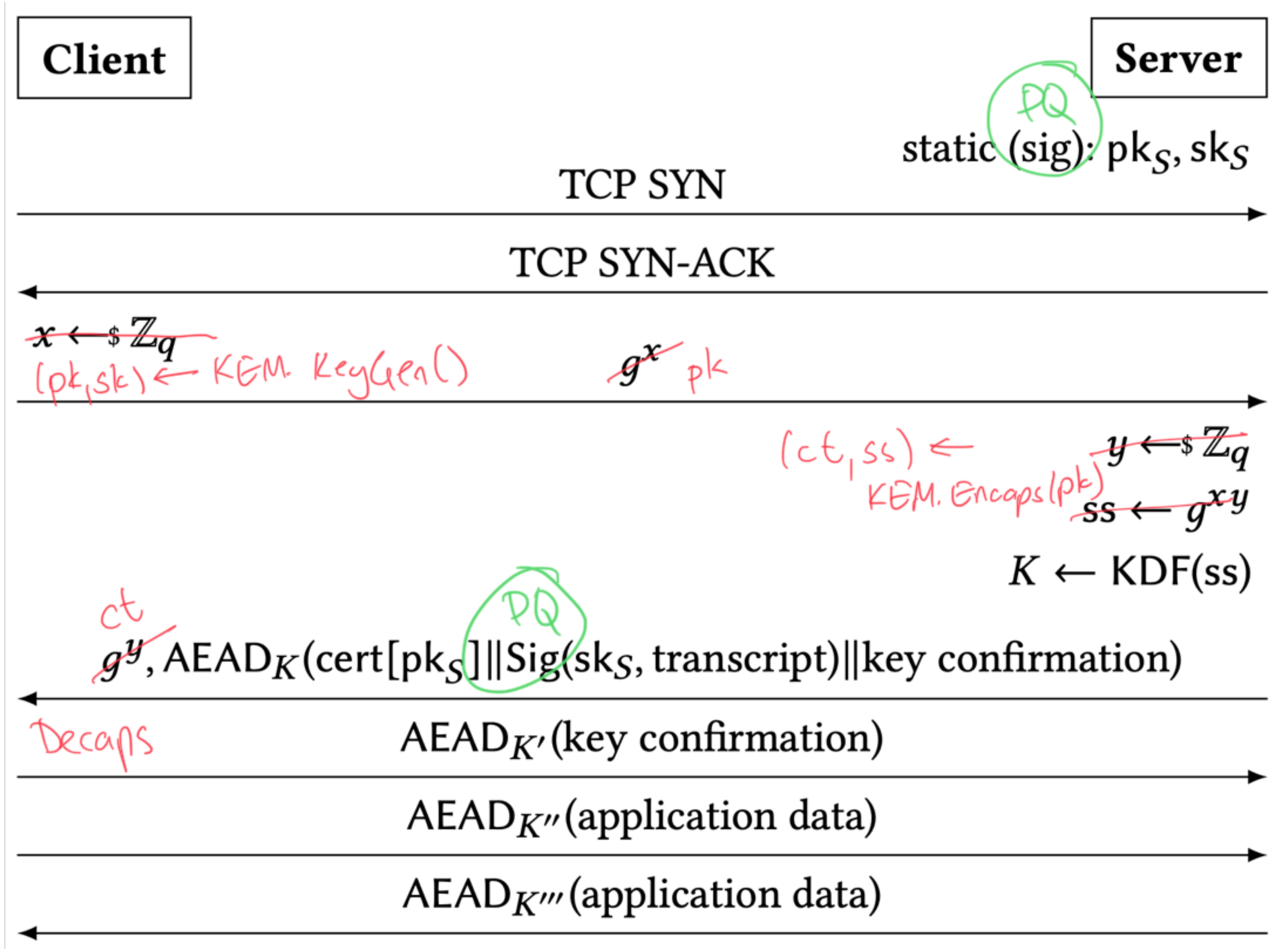
Signed Diffie–Hellman



TLS 1.3 handshake

~~Signed Diffie–Hellman~~

Post-Quantum!!!



Outline

Hybrid

Prototyping with OQS

Benchmarking

New protocol designs (KEMTLS)

Hybrid cryptography
“Dual algorithm”

Combining traditional and post-quantum algorithms

Security goals for hybridization

- PQ security for early adopters without sacrificing current security

- “Robust” security:
  - Final session key should be secure as long as at least one of the ingredient keys is unbroken
- Most obvious techniques are fine, though with some subtleties [GHP18], [BBFGS19]

[GHP18] Giacon, Heuer, Poettering. PKC 2018. https://eprint.iacr.org/2018/024
[BBFGS19] Bindel, Brendel, Fischlin, Goncalves, Stebila. PQCrypto 2019. https://eprint.iacr.org/2018/903

Functionality goals for hybridization

- Backwards compatibility
  - Hybrid-aware client, hybrid-aware server
  - Hybrid-aware client, non-hybrid-aware server
  - Non-hybrid-aware client, hybrid-aware server
- Low computational overhead
- Low latency
- No extra round trips
- No duplicate information

Design options

1. How to negotiate algorithms
2. How to convey cryptographic data (public keys / ciphertexts)
3. How to combine keying material

- How combine keying material
  - XOR keys
  - Concatenate keys and use directly
  - Concatenate keys then apply a hash function / KDF
  - Extend the protocol's "key schedule" with new stages for each key
  - Insert the 2nd key into an unused spot in the protocol's key schedule

[SFG19] Stebila, Fluhrer, Gueron. https://tools.ietf.org/html/draft-stebila-tls-hybrid-design-03

Draft standards

- **NIST SP 800-56C**
  - "Recommendation for Key-Derivation Methods in Key Establishment Schemes" – includes various combiners
- **Hybrid key exchange in TLS** [SFG20]
- **Hybrid key exchange in SSH** [KSFHS20]
- ETSI

[NIST] https://csrc.nist.gov/publications/detail/sp/800-56c/rev-2/final
[SFG20] Stebila, Fluhrer, Gueron. https://tools.ietf.org/html/draft-ietf-tls-hybrid-design-01
[KSFHS20] Kampanakis, Stebila, Friedl, Hansen, Sikeridis. https://tools.ietf.org/html/draft-kampanakis-curdle-pq-ssh-00

Protocol constraints

- TLS 1.2
  - Message size limit: 2^{24} bytes
  - Fragment size limit: 2^{14} bytes
  - OpenSSL key exchange message buffer: 20,480 bytes
    - FrodoKEM level 5: 21,600 bytes public key / ciphertext
    - Classic McEliece level 1: 261,120 bytes public key
- TLS 1.3
  - Key exchange message size limit: 2^{16} bytes (OpenSSL: 20,000 bytes)
  - Certificate size limit: 2^{24} bytes (OpenSSL $2^{16.6}$ bytes)
  - Signature size limit: 2^{16} bytes (OpenSSL 2^{14} bytes)
    - Picnic1 level 1: 34,000 bytes signature (but Picnic 3 is small enough)
    - Rainbow: 58KB-1.7MB public keys

Implementation patch to fix

Need protocol changes to fix

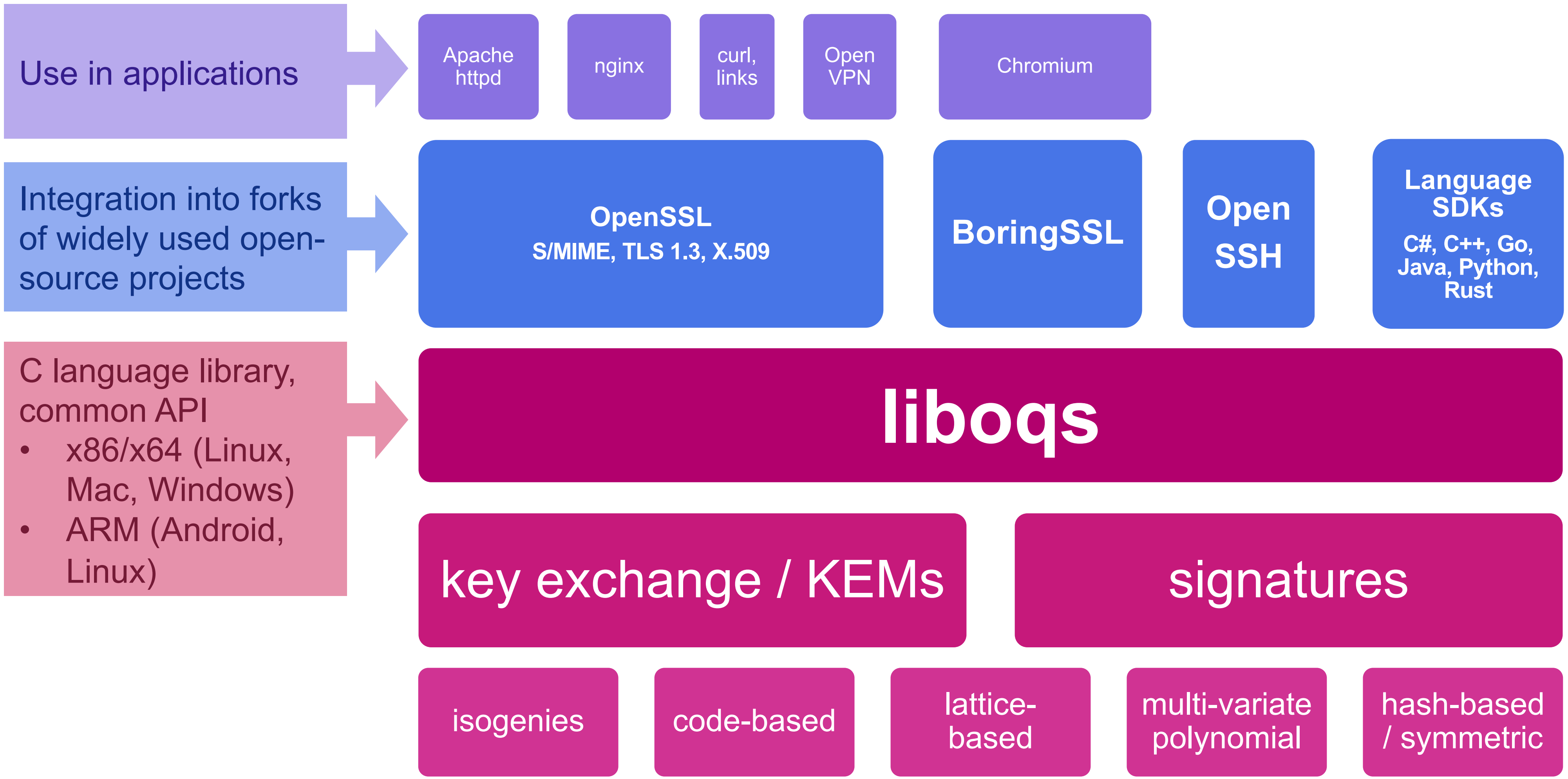
[CPS19] Crockett, Paquin, Stebila. NIST 2nd PQC Standardization Conference 2019. https://eprint.iacr.org/2019/858

OPEN QUANTUM SAFE

software for prototyping quantum-resistant cryptography

https://openquantumsafe.org https://github.com/open-quantum-safe

Open Quantum Safe Project



Industry partners:
- Amazon Web Services
- evolutionQ
- IBM Research
- Microsoft Research

Additional contributors:
- Cisco
- Senetas
- PQClean project
- Individuals

Financial support:
- AWS
- Canadian Centre for Cyber Security
- NSERC

https://openquantumsafe.org/ • https://github.com/open-quantum-safe/

Benchmarking post-quantum crypto in TLS

Christian Paquin, Douglas Stebila, Goutam Tamvada.
PQCrypto 2020.
https://eprint.iacr.org/2019/1447

Goal

- Measure effect of **network latency** and **packet loss rate** on handshake completion time for post-quantum connections of various sizes

- Out of scope:
  - Effect of different CPU speeds from client or server
  - Effect of different post-quantum algorithms on server throughput

Related work

- [BCNS15] and [BCD+16] measured the impact of their post-quantum key-exchange schemes on the performance of an Apache server running TLS 1.2
- [KS19] and [SKD20] measured the impact of post-quantum signatures in TLS 1.3 on handshake time (with various server distances), and handshake failure rate and throughput for a heavily loaded server

[BCNS15] Bos, Costello, Naehrig, Stebila. IEEE S&P 2015. https://eprint.iacr.org/2014/599
[BCD+16] Bos, Costello, Ducas, Mironov, Naehrig, Nikolaenko, Raghunathan, Stebila. ACM CCS 2016. https://eprint.iacr.org/2016/659
[KS19] Kampanakis, Sikeriis. https://eprint.iacr.org/2019/1276
[SKD20] Sikeridis, Kampanaokis, Devetsikiotis. NDSS 2020. https://eprint.iacr.org/2020/071

Related work: Internet-wide experiments

2016

Google, with NewHope in TLS 1.2

→

2018

Google, with "dummy extensions"

→

2019

Google and Cloudflare, with SIKE and NTRU-HRSS in TLS 1.3

Langley, 2016. https://www.imperialviolet.org/2016/11/28/cecpq1.html
Langley, 2018. https://www.imperialviolet.org/2018/12/12/cecpq2.html
Sullivan, Kwiatkowski, Langley, Levin, Mislove, Valenta. NIST 2nd PQC Standardization Conference 2019. https://csrc.nist.gov/Presentations/2019/measuring-tls-key-exchange-with-post-quantum-kem

What if you don’t have billions of clients and millions of servers?

(Inspired by NetMirage and Mininet)

Emulate the network!

+ more control over experiment parameters

+ easier to isolate effects of network characteristics

– loss in realism

Network emulation in Linux

- Kernel can create **network namespaces**: Independent copies of the kernel’s network stack
- **Virtual ethernet devices** can be created to connect the two namespaces
- **netem (network emulation)** kernel module
  - Can instruct kernel to apply a specified delay to packets
  - Can instruct kernel to drop packets with a specified probability

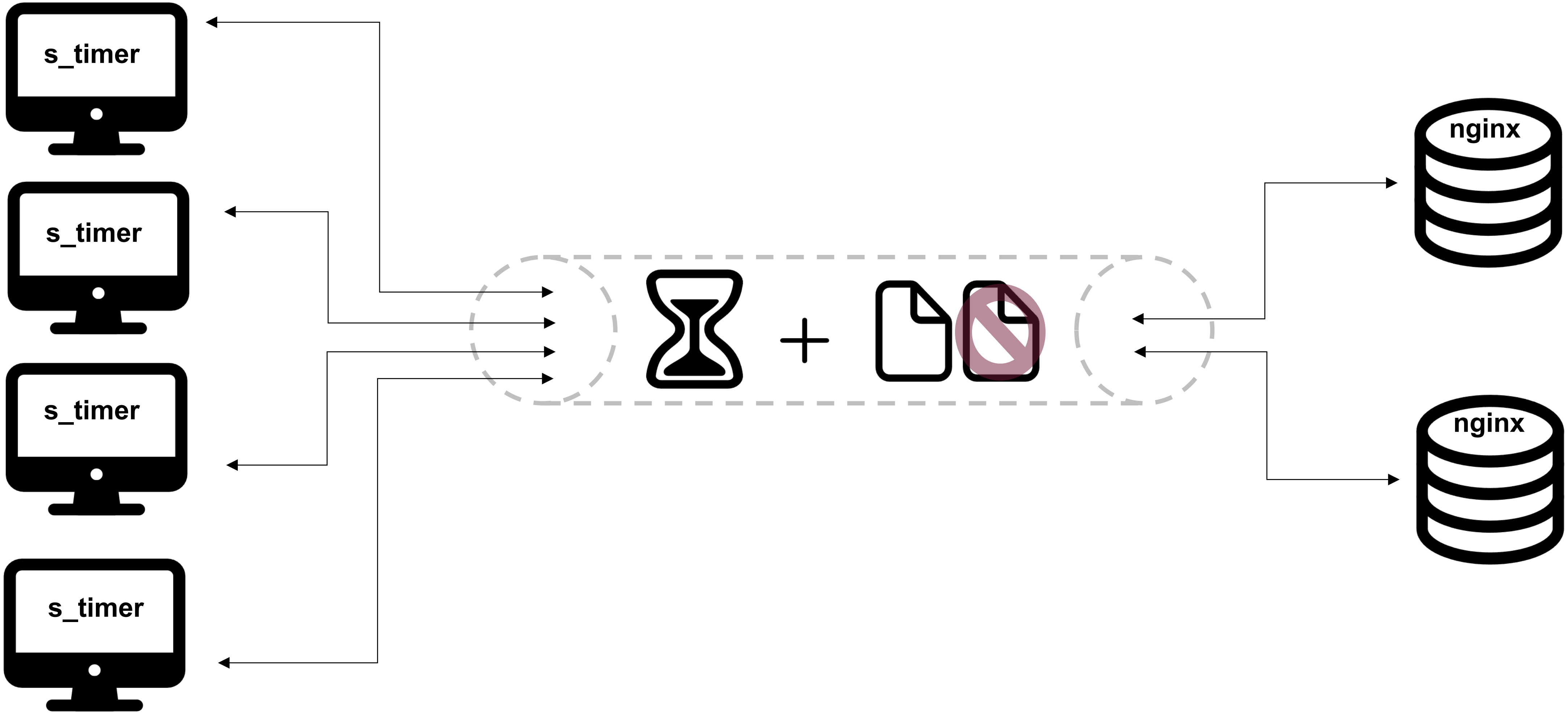
Network emulation experiment

- **Client namespace:** s_timer (Modified version of OpenSSL s_time)
  - Closes the connection on handshake completion, and records only the time taken to complete the handshake. i.e. **No application data is exchanged**
  - Built against **OQS-OpenSSL 1.1.1** (OpenSSL fork which adds post-quantum+classical key exchange and authentication to TLS 1.3)

- **Server namespace**: nginx, built against OQS-OpenSSL 1.1.1

Code available at https://github.com/xvzcf/pq-tls-benchmark

Network emulation experiment (contd.)



Icons from https://ionicons.com/

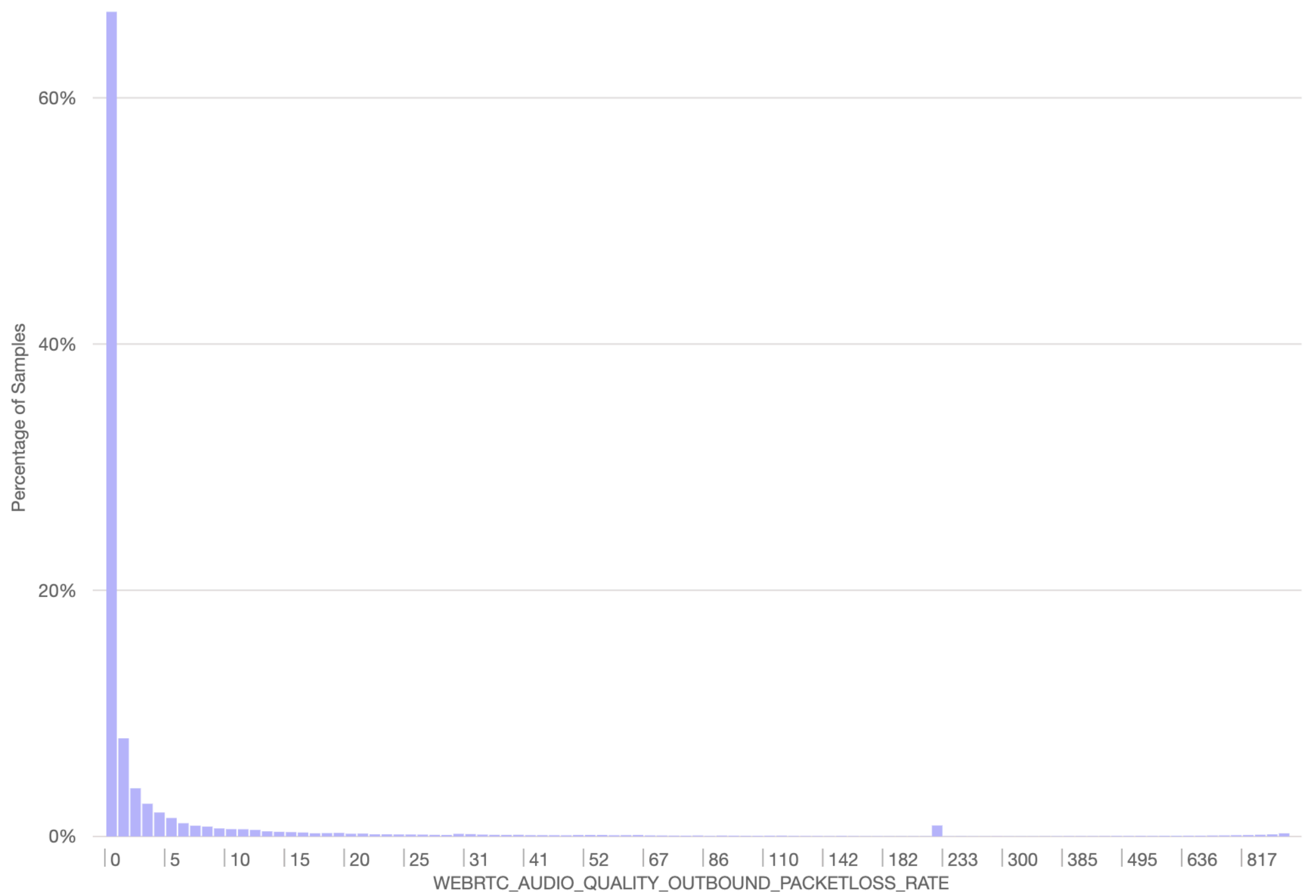
Experiment round-trip times

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

Packet loss rates

WEBRTC_AUDIO_QUALITY_OUTBOUND_PACKETLOS... **distribution for** **Firefox Desktop** **nightly 71**, **on** **any OS (62)** **any architecture (3)** **with** **any process** **and compare by** **none**

RTCP-reported packet loss by remote recipient of outbound audio (permille). Sampled every second of a call (for easy comparison). More details



Histogram Type	exponential
Ping Count	15.74k
Sample Count	35.49M
Sample Sum	657.28M
Number of dates	49
Selected Dates	2019/09/02 to 2019/10/21

5th Percentile	0
25th Percentile	0
Median	0
75th Percentile	2
95th Percentile	43.88

Notice percentiles are estimated based on values in the histogram. Values are only guaranteed to be accurate to the nearest bucket.

Export CSV Export JSON

https://telemetry.mozilla.org

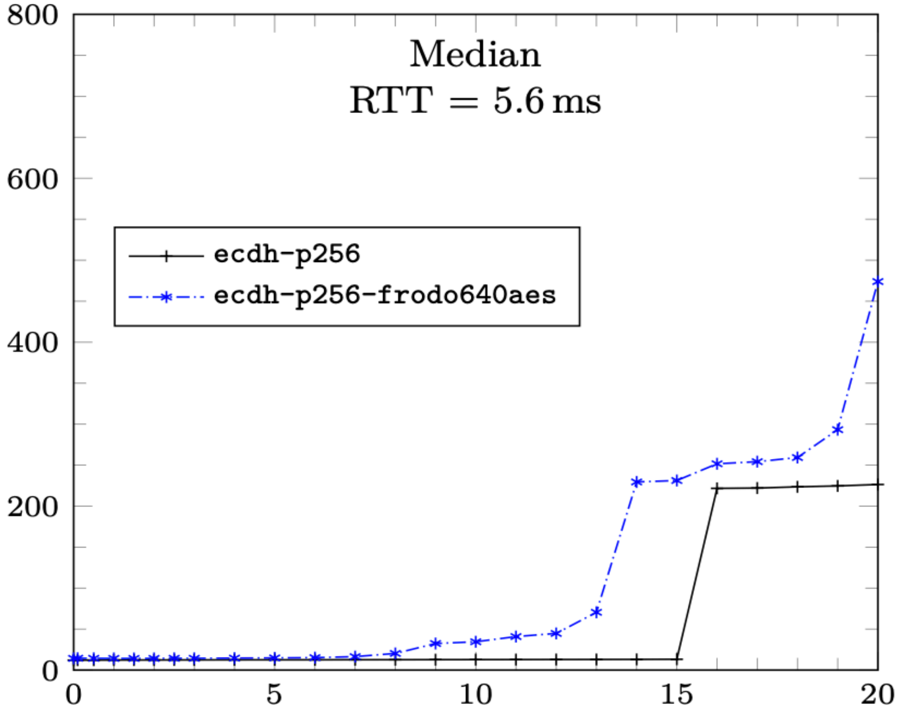
Algorithms evaluated

Notation	Hybrid	Family	Variant	Implementation
Key exchange				
`ecdh-p256`	×	Elliptic-curve	NIST P-256	OpenSSL optimized
`ecdh-p256-sike-p434`	✓	Supersingular isogeny	SIKE p434 [JAC+19]	Assembly optimized
`ecdh-p256-kyber512_90s`	✓	Module LWE	Kyber 90s level 1 [SAB+19]	AVX2 optimized
`ecdh-p256-frodo640aes`	✓	Plain LWE	Frodo-640-AES [NAB+19]	C with AES-NI
Signatures				
`ecdsa-p256`	×	Elliptic curve	NIST P-256	OpenSSL optimized
`dilithium2`	×	Module LWE/SIS	Dilithium2 [LDK+19]	AVX2 optimized
`qtesla-p-i`	×	Ring LWE/SIS	qTESLA provable 1 [BAA+19]	AVX2 optimized
`picnic-l1-fs`	×	Symmetric	Picnic-L1-FS [ZCD+19]	AVX2 optimized

Experiments were run on an **Ubuntu 18.04 Azure D64s v3 VM**, with **64 vCPUs** (2.60 GHz Intel Xeon Platinum 8171M) and **256 GiB of RAM**

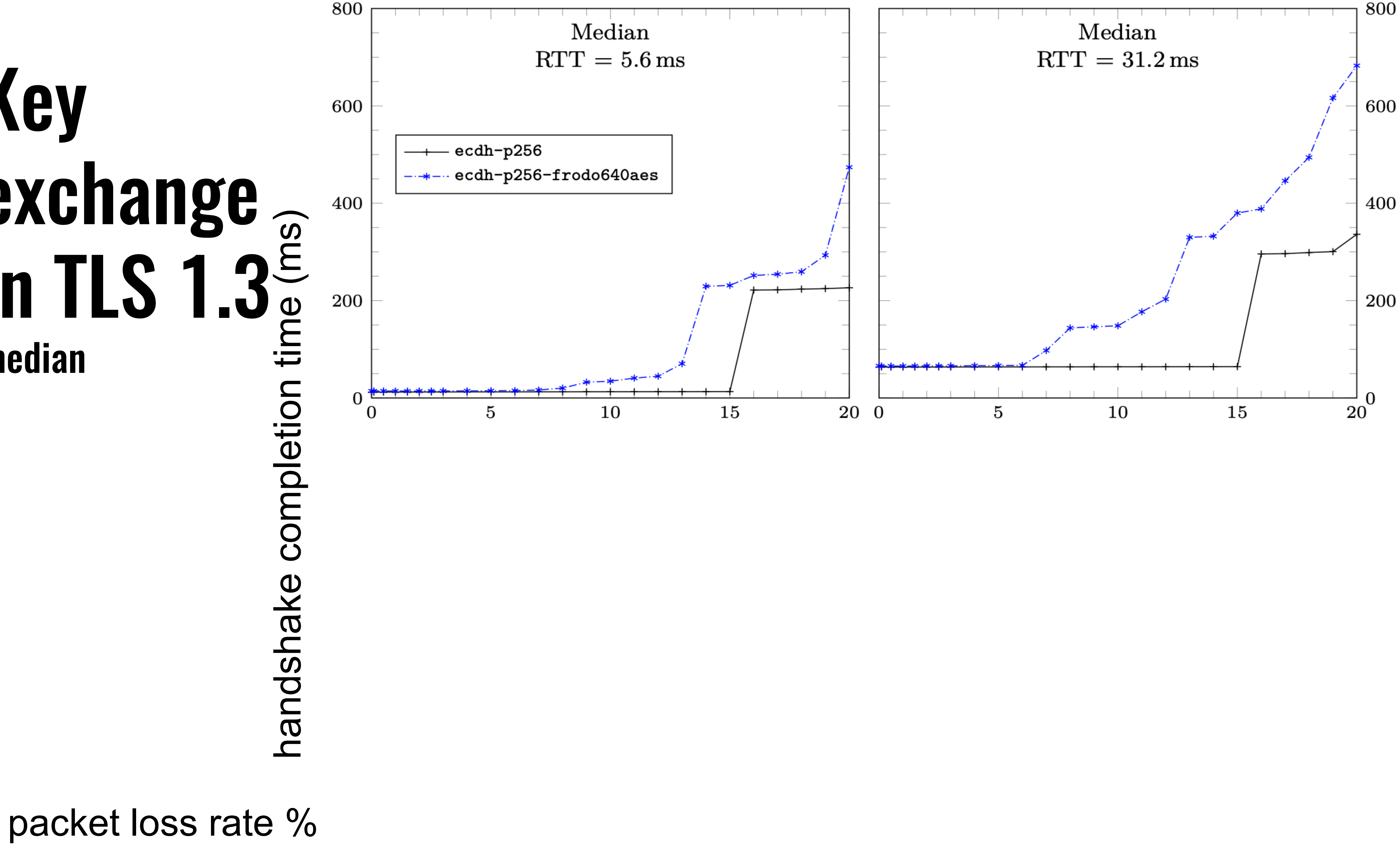
Key exchange in TLS 1.3

median



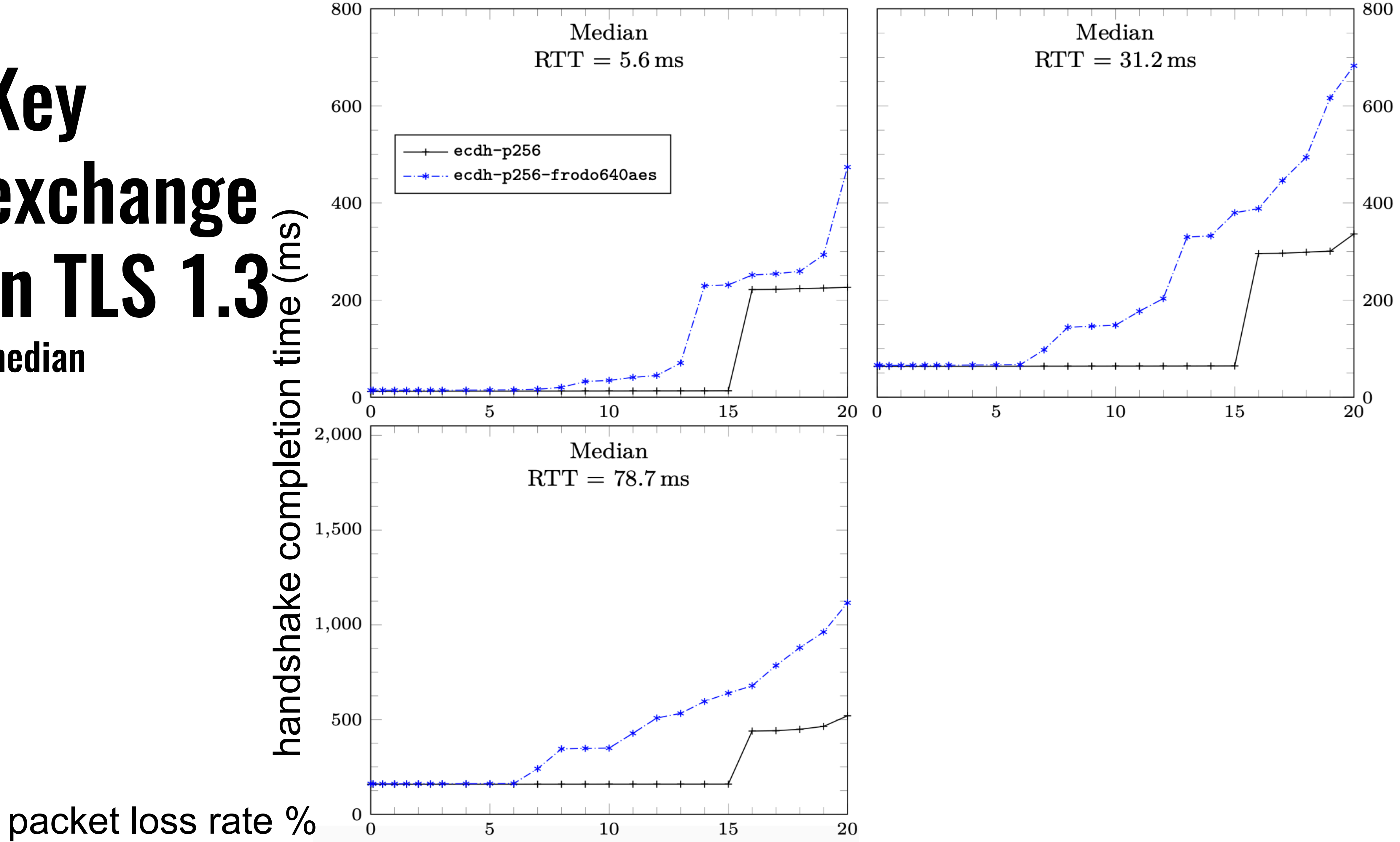
Key exchange in TLS 1.3

median



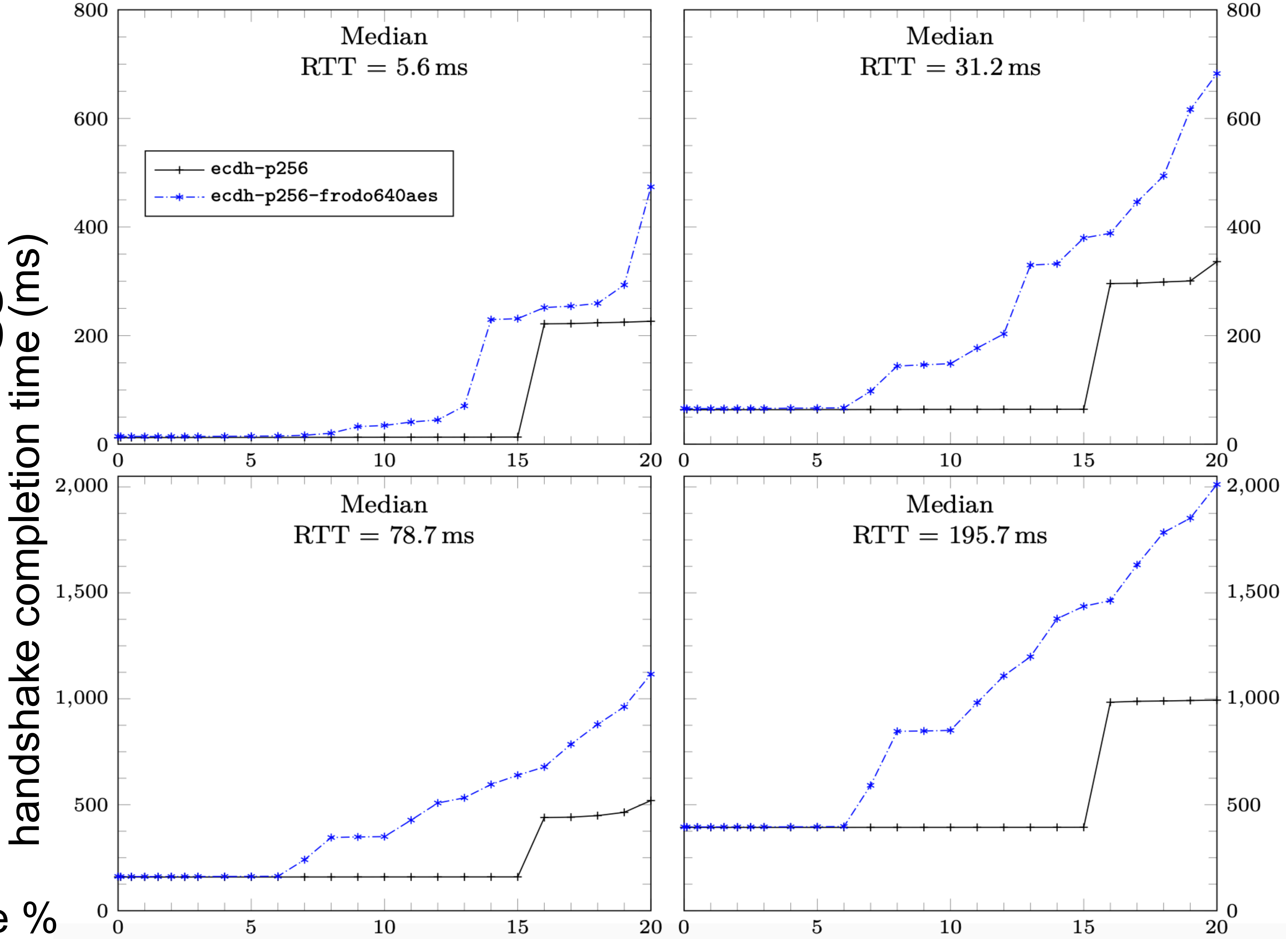
Key exchange in TLS 1.3

median



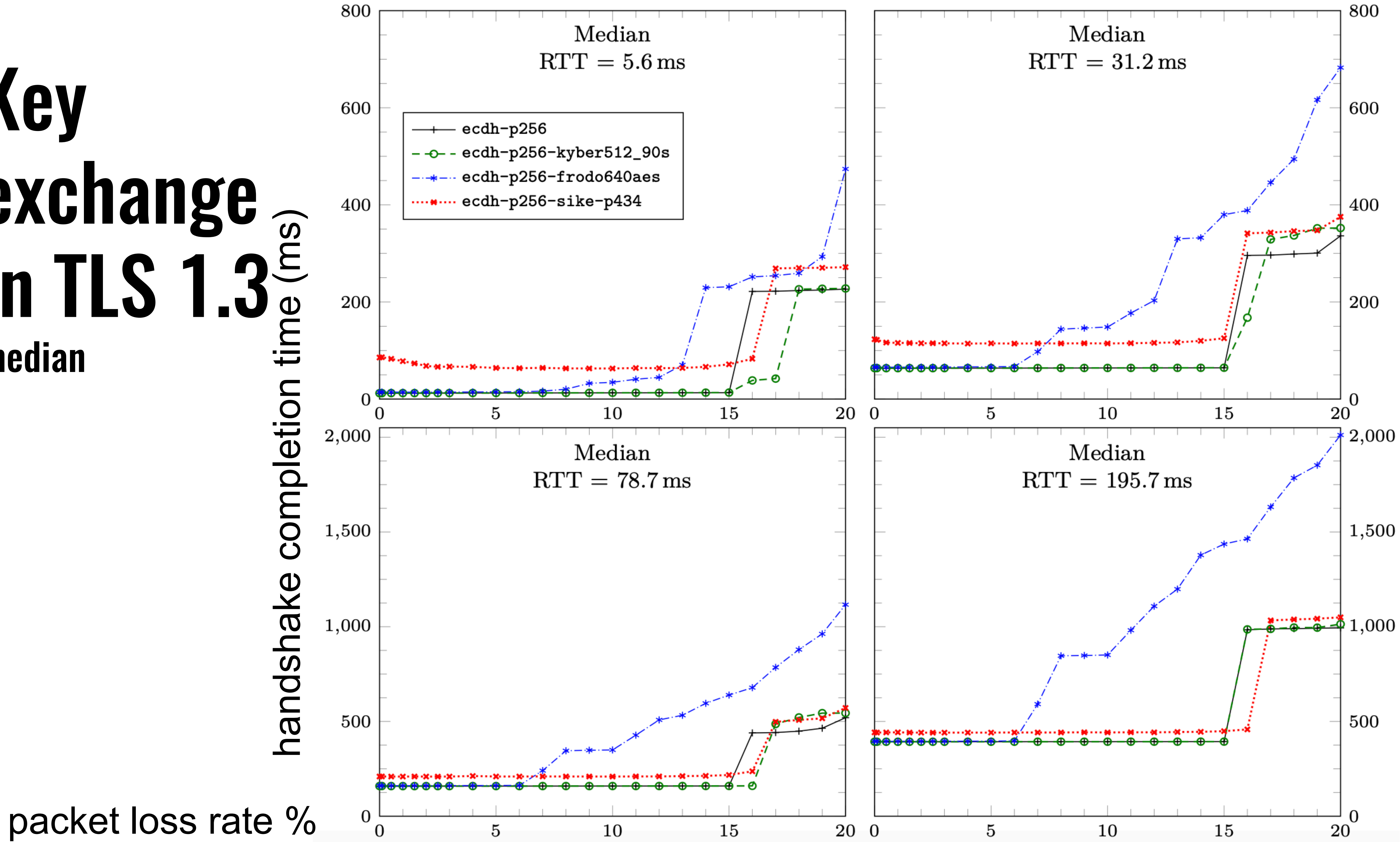
Key exchange in TLS 1.3

median



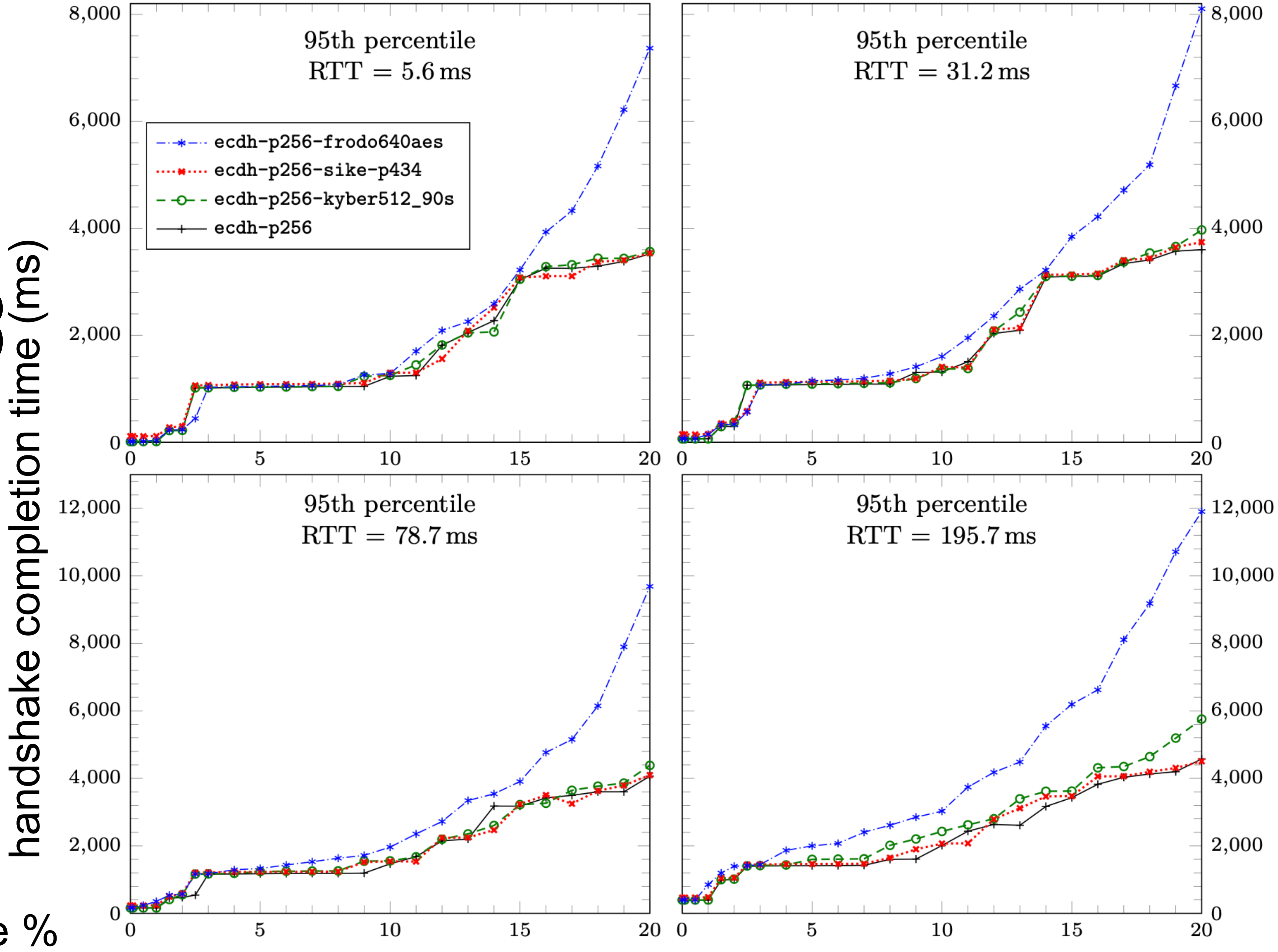
Key exchange in TLS 1.3

median



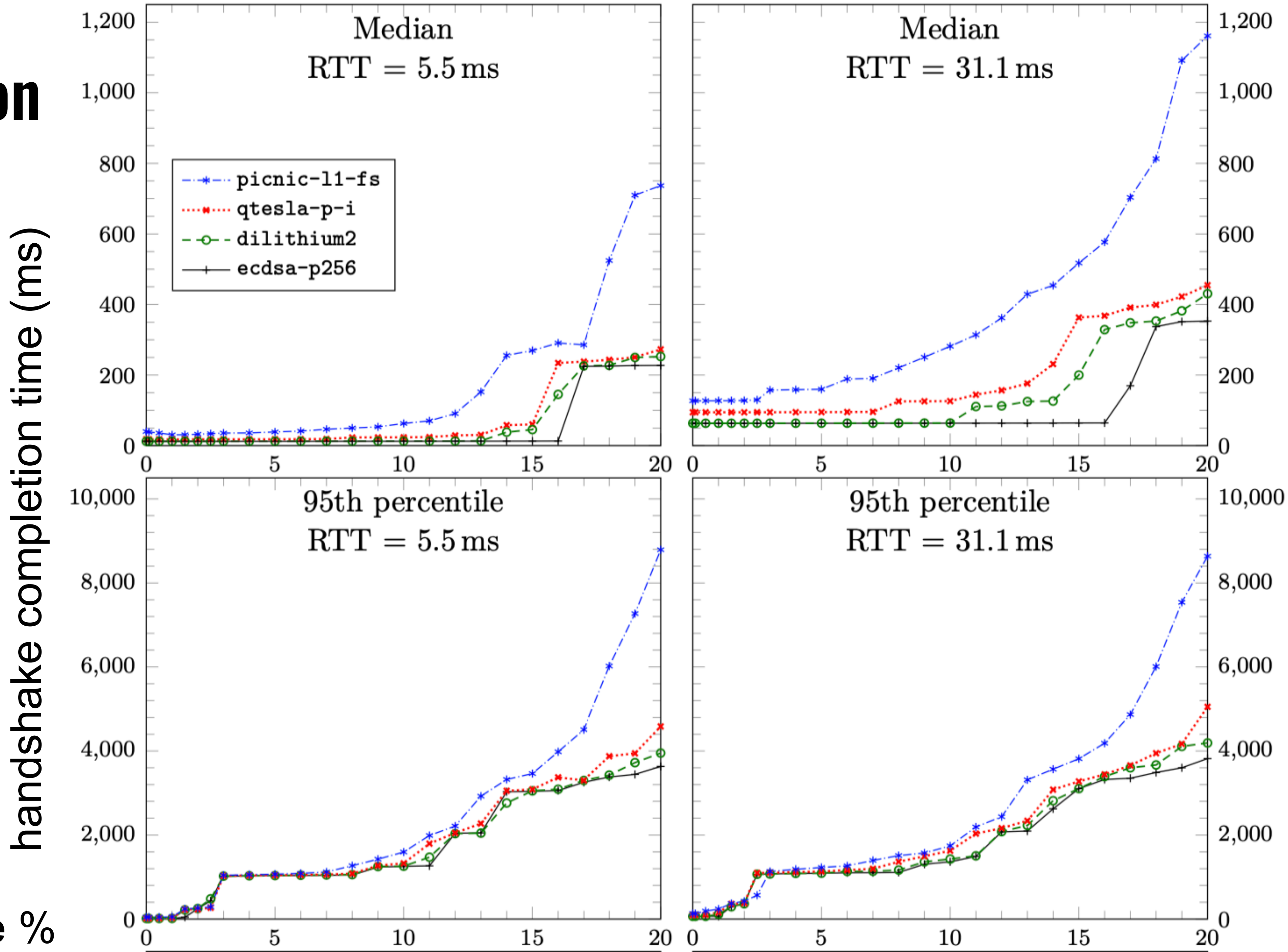
Key exchange in TLS 1.3

95th percentile



Authentication in TLS 1.3

median and 95th percentiles, lower network latencies



Conclusions

- On **fast, reliable network links**, the cost of public key cryptography dominates the median TLS establishment time, but does not substantially affect the 95th percentile establishment time
- On **unreliable network links** (packet loss rates >= 3%), 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

Future work

- Update the results for **Round 3**
- Automated benchmarking framework
- Extend the emulation results to **bigger networks** that aim to emulate multiple network conditions simultaneously **using NetMirage** or **Mininet**
- Investigate protocols such as **SSH, IPsec,** and **Wireguard** with our emulation framework

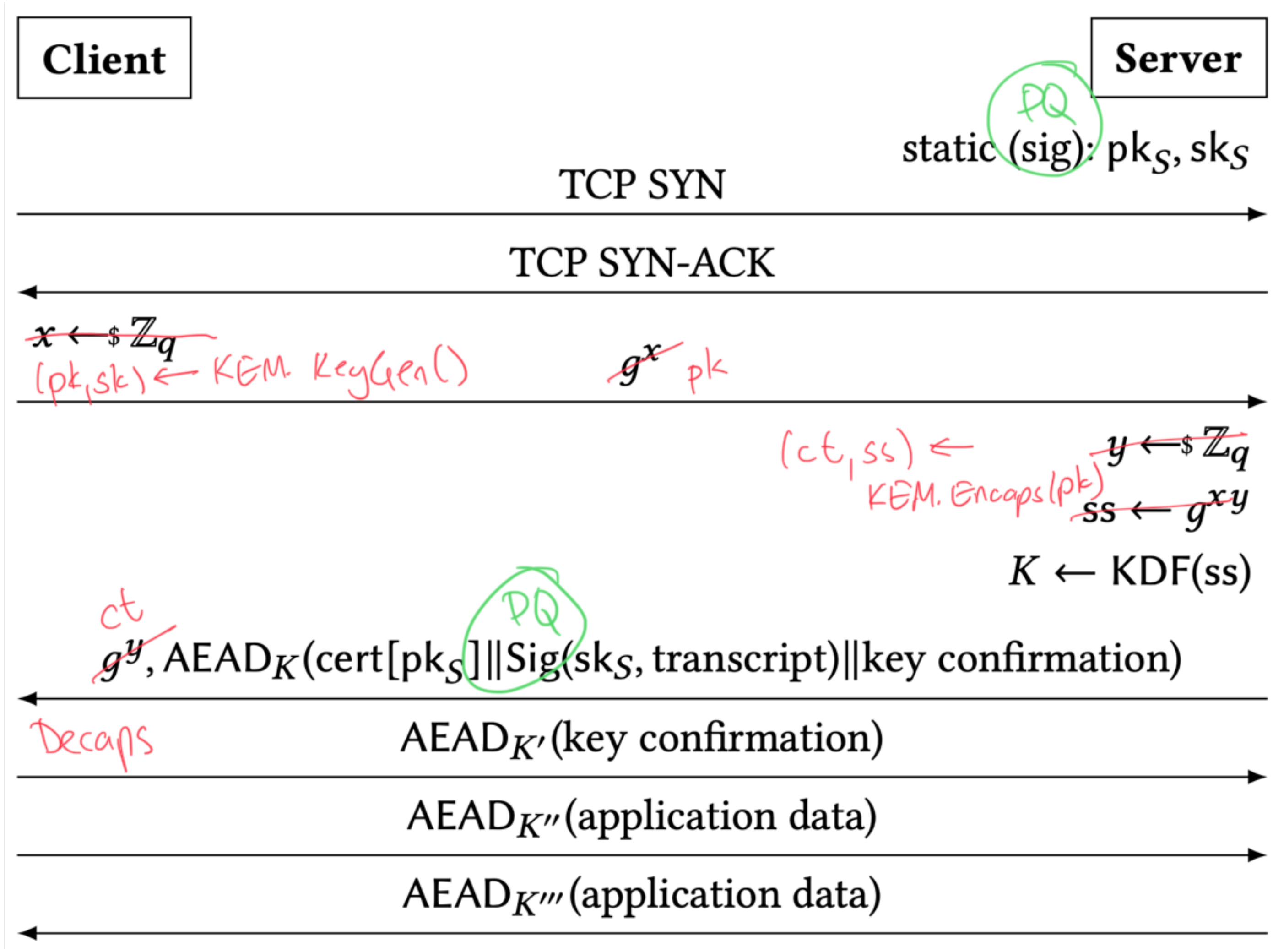
Post-quantum TLS without handshake signatures

Peter Schwabe, Douglas Stebila, Thom Wiggers.
ACM CCS 2020.
https://eprint.iacr.org/2020/534

TLS 1.3 handshake

~~Signed Diffie–Hellman~~

Post-Quantum!!!



Problem

post-quantum
signatures
are big

Solution

use
post-quantum KEMs
for authentication

Implicitly authenticated KEX is not new

In theory

- DH-based: SKEME, MQV, HMQV, …
- KEM-based: BCGP09, FSXY12

In practice

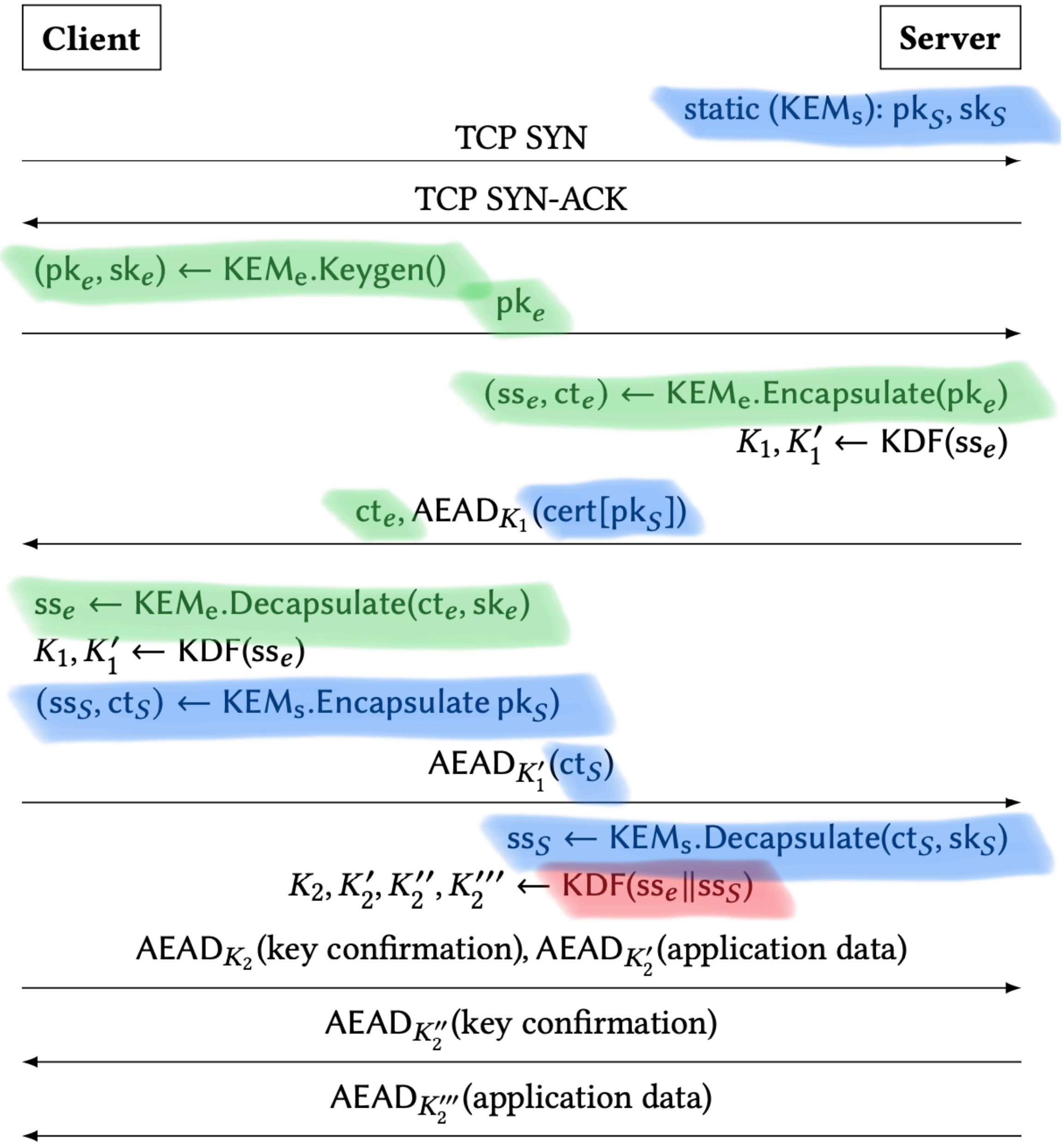
- RSA key transport in TLS ≤ 1.2
  - Lacks forward secrecy
- Signal, Noise, Wireguard
  - DH-based
  - Different protocol flows
- OPTLS
  - DH-based
  - Requires a non-interactive key exchange (NIKE)

“KEMTLS” handshake

KEM for
ephemeral key exchange

KEM for
server-to-client
authenticated key exchange

Combine shared secrets



Algorithm choices

KEM for ephemeral key exchange

- IND-CCA (or IND-1CCA)
- Want small public key
  + small ciphertext

KEM for authenticated key exchange

- IND-CCA
- Want small public key
  + small ciphertext

Signature scheme for intermediate CA

- Want small public key
  + small signature

Signature scheme for root CA

- Want small signature

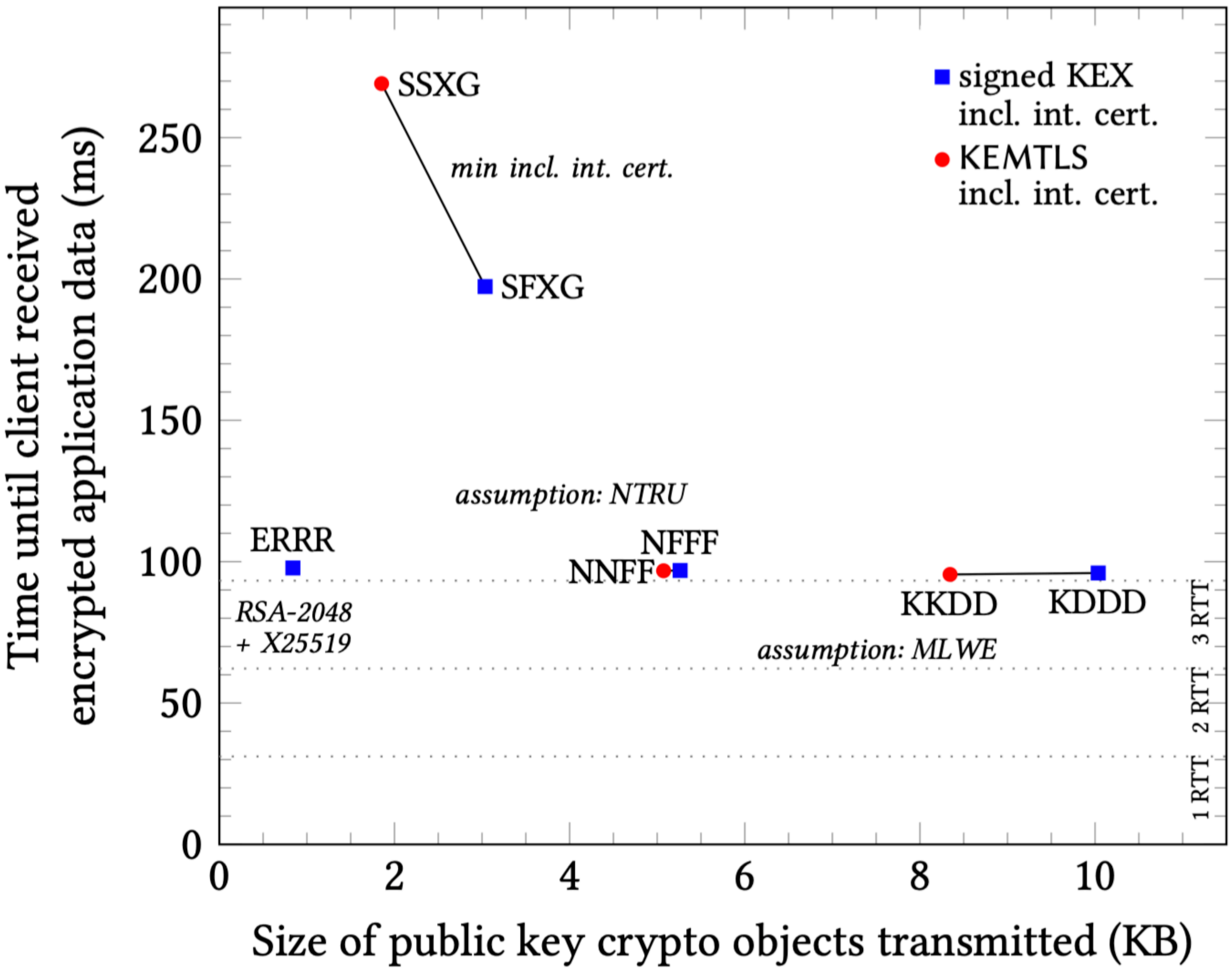
4 scenarios

1. Minimize size when intermediate certificate transmitted
2. Minimize size when intermediate certificate not transmitted (cached)
3. Use solely NTRU assumptions
4. Use solely module LWE/SIS assumptions

Signed KEX versus KEMTLS

Labels ABCD:
A = ephemeral KEM
B = leaf certificate
C = intermediate CA
D = root CA

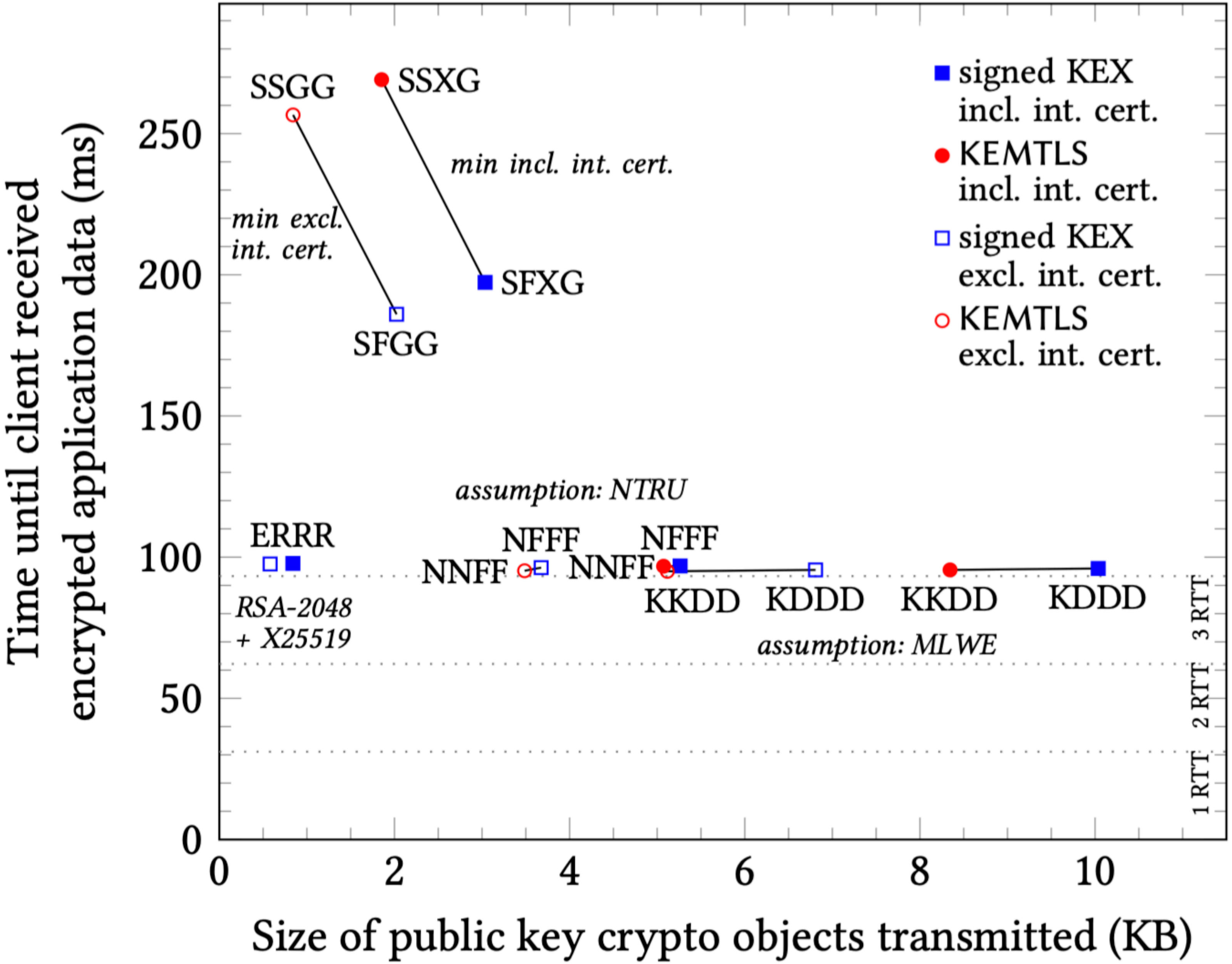
Algorithms: (all level 1)
Dilithium,
ECDH X25519,
Falcon,
GeMSS,
Kyber,
NTRU,
RSA-2048,
SIKE,
XMSS'



Signed KEX versus KEMTLS

Labels ABCD:
A = ephemeral KEM
B = leaf certificate
C = intermediate CA
D = root CA

Algorithms: (all level 1)
Dilithium,
ECDH X25519,
Falcon,
GeMSS,
Kyber,
NTRU,
RSA-2048,
SIKE,
XMSS'



Observations

- Size-optimized KEMTLS requires < ½ communication of size-optimized PQ signed-KEM
- Speed-optimized KEMTLS uses 90% fewer server CPU cycles and still reduces communication
  - NTRU KEX (27 μs) 10x faster than Falcon signing (254 μs)
- No extra round trips required until client starts sending application data
- Smaller trusted code base (no signature generation on client/server)

Security

Security model: multi-stage key exchange, extending [DFGS21]

- Key indistinguishability
- Forward secrecy
- Implicit and explicit authentication

Ingredients in security proof:

- **IND-CCA for long-term KEM**
- **IND-1CCA for ephemeral KEM**
- Collision-resistant hash function
- Dual-PRF security of HKDF
- EUF-CMA of HMAC

[DFGS21] Dowling, Fischlin, Günther, Stebila. Journal of Cryptology, 2021. https://eprint.iacr.org/2020/1044

Security subtleties: authentication

Implicit authentication

- Client's first application flow can't be read by anyone other than intended server, but client doesn't know server is live at the time of sending
- Also provides a form of deniable authentication since no signatures are used
  - Formally: offline deniability [DGK06]

Explicit authentication

- Explicit authentication once key confirmation message transmitted
- *Retroactive* explicit authentication of earlier keys

[DGK06] Di Raimondo, Gennaro, Krawczyk. ACM CCS 2006. https://eprint.iacr.org/2006/280

Security subtleties: downgrade resilience

- Choice of cryptographic algorithms not authenticated at the time the client sends its first application flow
  - MITM can't trick client into using undesirable algorithm
  - But MITM can trick them into temporarily using suboptimal algorithm

- Formally model 3 levels of downgrade-resilience:
  1. Full downgrade resilience
  2. No downgrade resilience to unsupported algorithms
  3. No downgrade resilience

Security subtleties: forward secrecy

- **Weak forward secrecy 1:** adversary passive in the test stage
- **Weak forward secrecy 2:** adversary passive in the test stage or never corrupted peer's long-term key
- **Forward secrecy:** adversary passive in the test stage or didn't corrupt peer's long-term key before acceptance

- Can make detailed forward secrecy statements, such as:
  - Stage 1 and 2 keys are wfs1 when accepted, retroactive fs once stage 6 accepts

My most applied, ready for adoption idea ever!!!!!!

Reviewer 2:

“What about 0-RTT?
What about QUIC and TCP FastOpen?
What about encrypted SNI?”

Reviewer 2 does not like puppies. Don’t be like Reviewer 2. http://relationsinternational.com/dont-be-reviewer-2/

Chris Wood:

Cloudflare
Co-chair of TLS working group

“Server can’t send application data in its first TLS flow. Will that break HTTP/3 where the server sends a SETTINGS frame?”

Mike Ounsworth:

EntrustDataCard

“How do you do certificate lifecycle management with KEM public keys?”

Certificate lifecycle management for KEM public keys

Proof of possession: How does requester prove possession of corresponding secret keys?

- Not really addressed in practice, since RSA and DL/ECDL keys can be used for both signing and encryption/KEX
- Can't sign like in a Certificate Signing Request (CSR)
- Could do interactive challenge-response protocol (or just run KEMTLS), but need online verification (RFC 4210 Sect. 5.2.8.3)
- Send cert to requestor encrypted under key in the certificate (RFC 4210 Sect. 5.2.8.2) – but maybe broken by Certificate Transparency?
- Zero-knowledge proof of knowledge?

Certificate lifecycle management for KEM public keys

Revocation: How can certificate owner authorize a revocation request?

- Put a (hash of a) signature public key in the cert which can be used to revoke the cert?
  - Possibly could simplify to just revealing a hash preimage

Conclusions on KEMTLS

- Summary of protocol design: implicit authentication via KEMs

- Saves bytes on the wire and server CPU cycles
- Preserves client request after 1-RTT
- Caching intermediate CA certs brings even greater benefits

- Protocol design is simple to implement, provably secure
- Also have a variant supporting client authentication
- Working with Cloudflare to test within their infrastructure

Post-quantum TLS

Douglas Stebila

UNIVERSITY OF WATERLOO

Hybrid PQ + traditional

- Design and security
  - https://tools.ietf.org/html/draft-ietf-tls-hybrid-design-01
  - https://eprint.iacr.org/2019/858
  - https://eprint.iacr.org/2018/903
- Standardization
  - https://tools.ietf.org/html/draft-ietf-tls-hybrid-design-01
  - https://tools.ietf.org/html/draft-kampanakis-curdle-pq-ssh-00

Prototyping

- Open Quantum Safe project
  - https://openquantumsafe.org
  - https://github.com/open-quantum-safe/

Benchmarking

- https://eprint.iacr.org/2019/1447
- https://github.com/xvzcf/pq-tls-benchmark
- https://github.com/open-quantum-safe/speed

New protocol design

- Implicit authentication using KEMs
  - https://eprint.iacr.org/2020/534
  - https://github.com/thomwiggers/kemtls-experiment/

https://www.douglas.stebila.ca/research/presentations/

Appendix

KEMTLS data

		Abbrv.	KEX (pk+ct)	Excluding intermediate CA certificate: HS auth (ct/sig)	Leaf crt. subject (pk)	Leaf crt. (signature)	Sum excl. int. CA cert.	Including intermediate CA certificate: Int. CA crt. subject (pk)	Int. CA crt. (signature)	Sum incl. int. CA crt.	Root CA (pk)	Sum TCP payloads of TLS HS (incl. int. CA crt.)
	TLS 1.3	ERRR	ECDH (X25519) 64	RSA-2048 256	RSA-2048 272	RSA-2048 256	**848**	RSA-2048 272	RSA-2048 256	**1376**	RSA-2048 272	2711
TLS 1.3 (Signed KEX)	**Min. incl. int. CA cert.**	SFXG	SIKE 405	Falcon 690	Falcon 897	$XMSS_s^{MT}$ 979	**2971**	$XMSS_s^{MT}$ 32	GeMSS 32	**3035**	GeMSS 352180	4056
	Min. excl. int. CA cert.	SFGG	SIKE 405	Falcon 690	Falcon 897	GeMSS 32	**2024**	GeMSS 352180	GeMSS 32	**354236**	GeMSS 352180	355737
	Assumption: MLWE+MSIS	KDDD	Kyber 1536	Dilithium 2044	Dilithium 1184	Dilithium 2044	**6808**	Dilithium 1184	Dilithium 2044	**10036**	Dilithium 1184	11094
	Assumption: NTRU	NFFF	NTRU 1398	Falcon 690	Falcon 897	Falcon 690	**3675**	Falcon 897	Falcon 690	**5262**	Falcon 897	6227
KEMTLS	**Min. incl. int. CA cert.**	SSXG	SIKE 405	SIKE 209	SIKE 196	$XMSS_s^{MT}$ 979	**1789**	$XMSS_s^{MT}$ 32	GeMSS 32	**1853**	GeMSS 352180	2898
	Min. excl. int. CA cert.	SSGG	SIKE 405	SIKE 209	SIKE 196	GeMSS 32	**842**	GeMSS 352180	GeMSS 32	**353054**	GeMSS 352180	354578
	Assumption: MLWE+MSIS	KKDD	Kyber 1536	Kyber 736	Kyber 800	Dilithium 2044	**5116**	Dilithium 1184	Dilithium 2044	**8344**	Dilithium 1184	9398
	Assumption: NTRU	NNFF	NTRU 1398	NTRU 699	NTRU 699	Falcon 690	**3486**	Falcon 897	Falcon 690	**5073**	Falcon 897	6066

https://eprint.iacr.org/2020/534

		Computation time for asymmetric crypto				Handshake time (31.1 ms latency, 1000 Mbps bandwidth)					
		Excl. int. CA cert.		Incl. int. CA cert.		Excl. int. CA cert.			Incl. int. CA cert.		
		Client	Server	Client	Server	Client sent req.	Client recv. resp.	Server HS done	Client sent req.	Client recv. resp.	Server HS done
TLS 1.3	**ERRR**	0.134	0.629	0.150	0.629	66.4	**97.6**	35.4	66.6	**97.8**	35.6
	SFXG	40.058	21.676	40.094	21.676	165.8	**196.9**	134.0	166.2	**197.3**	134.4
	SFGG	34.104	21.676	34.141	21.676	154.9	**186.0**	123.1	259.0	**290.2**	227.1
	KDDD	0.080	0.087	0.111	0.087	64.3	**95.5**	33.3	64.8	**96.0**	33.8
	NFFF	0.141	0.254	0.181	0.254	65.1	**96.3**	34.1	65.6	**96.9**	34.7
KEMTLS	**SSXG**	61.456	41.712	61.493	41.712	202.1	**268.8**	205.6	202.3	**269.1**	205.9
	SSGG	55.503	41.712	55.540	41.712	190.4	**256.6**	193.4	293.3	**359.5**	296.3
	KKDD	0.060	0.021	0.091	0.021	63.4	**95.0**	32.7	63.9	**95.5**	33.2
	NNFF	0.118	0.027	0.158	0.027	63.6	**95.2**	32.9	64.2	**95.8**	33.5

https://eprint.iacr.org/2020/534