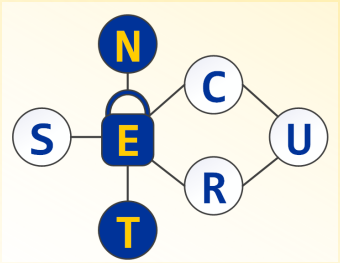


Enhancing Cross-Sectoral Collaboration in Cybersecurity in Estonia, Czechia, Lithuania, Ukraine, and the Netherlands

Funded by the European Union

SECURE-NET SEMINAR · 27 AUGUST 2026

Empirical Overhead Decomposition of Hybrid Post-Quantum Key Exchange in TLS 1.3

Where does the cost of a hybrid handshake actually come from — computation or transmission?

Nerijus Šatkauskas · Šarūnas Grigaliūnas

Cyber Security Centre of Excellence · Kaunas University of Technology

Funded by the European Union Horizon Europe programme under Grant Agreement No. 101217315. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or European Research Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

Why post-quantum, and why now

TLS 1.3 protects the modern web

Its key exchange (ECDHE / X25519) rests on the hardness of discrete-log problems.

A quantum computer breaks that assumption

Shor's algorithm would recover the shared secret from captured handshakes.

“Harvest now, decrypt later”

Traffic recorded today can be stored and decrypted once a machine exists — long-lived data is already at risk.

Deployment is not optional

Because the threat is retroactive, migrating to quantum-resistant key exchange is a decision that must be made now — not after large quantum machines arrive.

The real question becomes:
not “if”, but “what does it cost?”

Hybrid key exchange: keep the old, add the new

X25519 (ECDHE)

Classical, battle-tested. Secure against every classical adversary today.

+

ML-KEM-768

NIST-standardised (FIPS 203). Lattice-based, quantum-resistant.

→

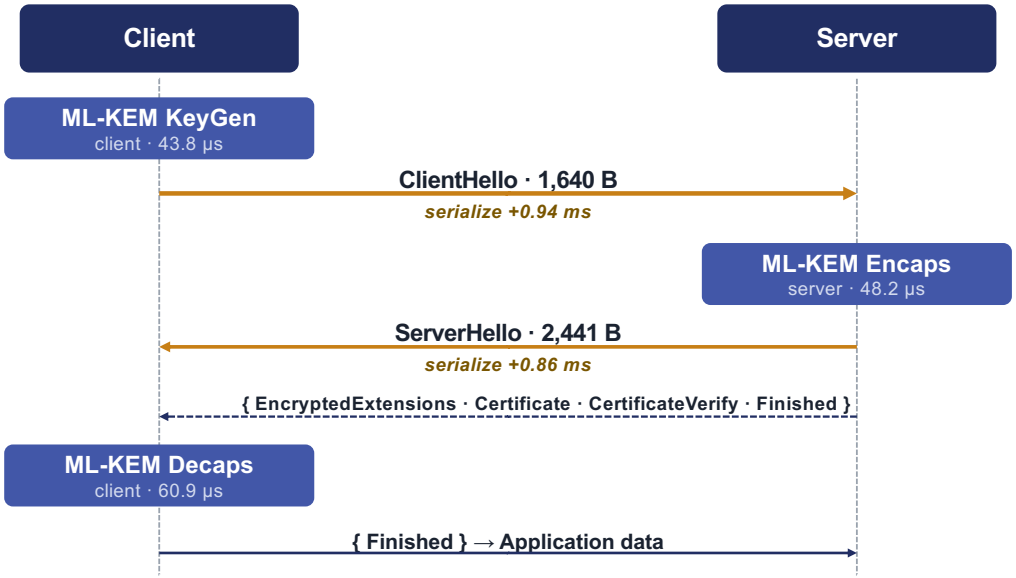
X25519MLKEM768

The most deployment-relevant hybrid group — already a first-class option in modern OpenSSL.

Why combine them?

Security holds as long as at least one component stays unbroken. The connection is safe against today's classical attackers via X25519, and against a future quantum attacker via ML-KEM. Hybrids also satisfy regulatory guidance during the transition period.

Where the delay sits in the TLS 1.3 handshake



*Compute (blue) runs on the endpoints in microseconds; the overhead is the flights taking longer to **serialize (ms)**. Propagation (RTT) is unchanged — which is why the overhead barely grows from S2 to S3.*

Where the time goes

crypto compute

0.153 ms · 5%

KeyGen + Encaps + Decaps — microseconds, on the endpoints

bytes on the wire

2.869 ms · 95%

serialize +2,250 B (CH +1,176 · SH +1,074)

Set by bandwidth, not RTT: propagation (~17.5 ms each way) is identical for classical and hybrid.

A controlled, reproducible testbed

01

Isolated testbed

Ubuntu 24.04 LTS in VirtualBox. Client & server in separate Linux network namespaces joined by a veth pair.

02

Emulated networks

Bidirectional tc/netem shapes delay, loss & bandwidth across four canonical scenarios (S0–S3).

03

500 handshakes / cell

Full TLS 1.3 handshakes per configuration, classical (C1) vs. hybrid (H1), with timing instrumentation.

04

Three evidence layers

Crypto micro-benchmarks + handshake latency + pcap packet analysis, linked together.

SOFTWARE STACK

liboqs 0.12.0 · OpenSSL 3.4.1 · OQS Provider 0.8.0 · Nginx 1.27.3 · RSA-2048 certs (held constant)

Four conditions, from datacenter to mobile

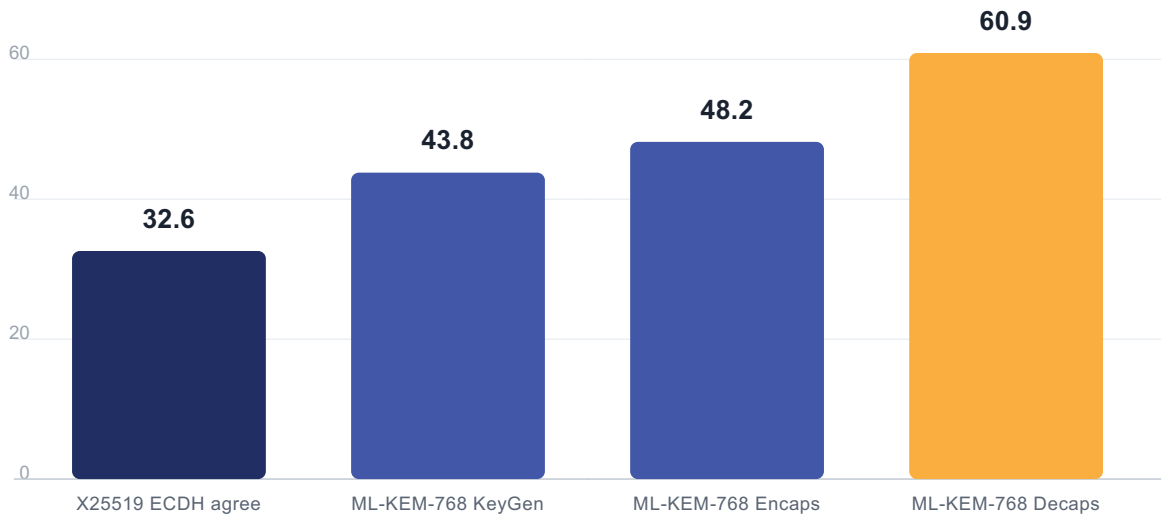
	Scenario	Delay (ms)	Loss (%)	Avg RTT (ms)	Bandwidth
S0	**Direct veth link**	0	0	0.041	Unlimited
S1	**Datacenter**	0.5	0	2.975	1 Gbps
S2	**Broadband**	17.5	0.1	37.309	10 Mbps
S3	**Mobile 4G**	35	3	72.462	10 Mbps

Why these four?

They span a direct local link, a low-latency datacenter, a typical broadband line, and a lossy mobile path — matching conditions used across prior PQC-TLS studies.

Focus on within-scenario deltas. Because both endpoints run inside one VM, absolute medians are not Internet RTTs. The analysis compares classical vs. hybrid within each scenario — S2 (broadband) is the representative case.

ML-KEM-768 is cheap; the crypto is not the problem



Per-operation cost, generic C build (no AVX2)

1.3–1.9×

ML-KEM-768 vs. X25519 per operation — same order of magnitude.

144–426×

HQC-192 is 2–3 orders of magnitude slower — a very different regime.

Takeaway: a compact lattice KEM can't dominate the TLS cost by computation alone. So where does the overhead come from?

Overhead appears once RTT and loss enter

Scenario	C1 P50	H1 P50	Δ (ms)	Ovhd %	C1 P90	H1 P90
S0 · Direct	11.32	11.77	**+0.46**	+4.0%	15.73	15.95
S1 · Datacenter	17.90	19.18	**+1.28**	+7.1%	21.94	23.69
S2 · Broadband	124.12	127.14	**+3.02**	+2.4%	127.41	130.15
S3 · Mobile 4G	233.63	236.66	**+3.03**	+1.3%	**577.64**	**627.20**

P50 = median (Table 3) · P90 = 90th percentile (Table 4). ***Under 3% loss (S3) the tail diverges far more than the median.***

All medians differ significantly (Mann–Whitney U, $p < 0.001$); 95% bootstrap CIs exclude zero.

% peaks in the middle

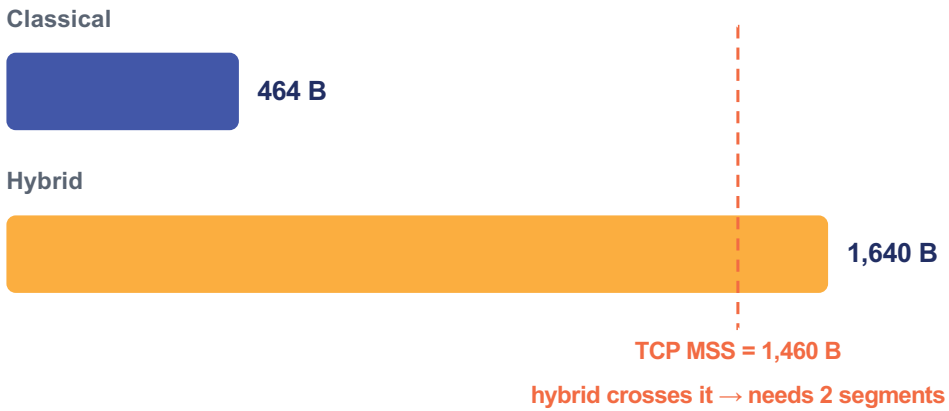
Relative overhead is largest at S1 (+7.1%), not at the highest RTT — the extra TCP segment adds a full ACK round-trip while the baseline is still small.

The median hides the tail

At S3 the median gap is a tame +3 ms, but at **P90 the hybrid is ~50 ms slower (627 vs 578 ms)** — highlighted at right. Under loss, the tail is what users feel.

The ClientHello outgrows a single TCP segment

ClientHello size on the wire



Mechanism. The hybrid ClientHello (1,640 B) exceeds the 1,460 B segment limit by 180 B, so it needs two TCP segments instead of one. Each extra segment adds an acknowledgment dependency proportional to RTT — negligible at S0, substantial at S2/S3.

Total handshake byte growth

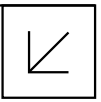
	Classical	Hybrid	Δ
ClientHello	464 B	1,640 B	**+1,176 B**
ServerHello	1,367 B	2,441 B	**+1,074 B**

+2,250 B

added to every full hybrid handshake

HQC-192 would add an estimated +13,300 B — nearly 6× more.

~95% of the overhead is bytes, not crypto



Computation

5.1%

0.153 ms — ML-KEM KeyGen + Encaps + Decaps

Bandwidth & transport

94.9%

2.869 ms — from +2,250 bytes, their segmentation, ACKs and queueing effects

The arithmetic

Δ total (S2 P50)

3.022 ms

− Δ crypto

0.153 ms

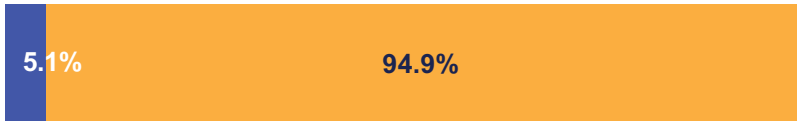
= Δ bandwidth

2.869 ms

Reported as +3.02 ms (+2.4%) in Table 3; 3.022 ms is the bootstrap median difference used in the decomposition.

With AVX2, the split tilts even further to bandwidth

VirtualBox (generic C)

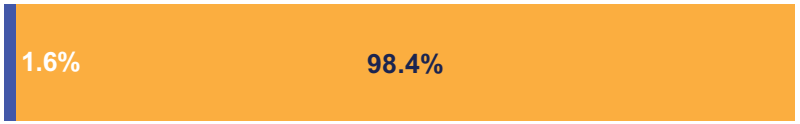


computation 0.153 ms **bandwidth** 2.869 ms

Δ total 3.022 ms

vs

Azure Standard_B2s (AVX2)



computation 0.038 ms **bandwidth** 2.265 ms

Δ total 2.303 ms

Crypto share falls to 1.6%

AVX2 accelerates ML-KEM operations 3.4–4.9×. Faster crypto shrinks an already-tiny slice.

Bandwidth is unchanged

The +2,250 bytes are identical on both machines. Hardware can't compress the wire cost away.

Conclusion strengthens

On production hardware the overhead is even more decisively bandwidth-dominated (98.4%).

A network-engineering problem, not a crypto one

Tune the wire, not the CPU

Right lever: keep ClientHello within the MSS, compression, larger initcwnd, MTU & congestion tuning. Hardware crypto touches only 1.6–5%.

Resumption amortises it

40–60% of production TLS connections are resumed — no KEM, ClientHello back to ~464 B. Amortised cost falls from ~3 ms toward ~0.2 ms.

This is the floor, not the ceiling

PQC certificates (ML-DSA / SLH-DSA) and larger KEMs like HQC (+13,300 B) push the same mechanism much harder. The small case validates the method.

The security is mandatory. So the useful question isn't "is 2.4% worth it?" but "what's the cheapest way to pay a cost we must pay?" — and that is a packet-budget problem operators already know how to solve.

Engineering the packet budget

Keep ClientHello in one segment

Trim extensions so the first message fits a single TCP segment — the **MSS** (Maximum Segment Size, ≈ 1,460 B, the largest chunk TCP sends at once).

→ removes the split into 2 segments and its extra ACK (the receiver's "got it" reply ≈ one round-trip)

Compress the handshake

Certificate compression (**RFC 8879**, an Internet standard) shrinks the server's reply; key-share compression trims the ClientHello.

→ attacks the +2,250 bytes on the wire directly

Larger initial congestion window

Raise the server's **initcwnd** — how many packets it may send before it must wait for an ACK — so the reply ships in one burst.

→ avoids an extra server-side round-trip

Bigger packets & congestion tuning

A larger **MTU** (Maximum Transmission Unit — the biggest packet a link carries) raises the MSS; congestion control such as **BBR** smooths sending under loss.

→ fewer segments · gentler P90 (slowest-10% latency) under loss

All four levers target the 95% that lives in bytes on the wire *— faster crypto (hardware / AVX2 vector instructions) only touches the 1.6–5% computation slice.*

What to take away

~95%

Overhead is bandwidth-driven

In S2, only 0.153 ms (5.1%) of the 3.02 ms hybrid overhead is cryptographic; the rest is +2,250 wire bytes and their transport effects.

1 → 2

The mechanism is segmentation

The hybrid ClientHello crosses the TCP MSS (1,640 > 1,460 B), forcing a second segment and an RTT-bound ACK.

98.4%

It strengthens on real hardware

AVX2 speeds ML-KEM 3.4–4.9× but leaves the bytes untouched, tilting the split further to bandwidth.

≈2.5%

Delta saturation

Doubling RTT from S2 to S3 barely changes the median overhead — extra bytes ride inside existing round-trips.

Migration challenge for hybrid TLS 1.3 is not a cryptographic efficiency problem — it is a packet-budget problem in the first handshake flights.

Thank you

Questions & discussion welcome

Code & results (open):

github.com/nerijusKTU/PQC_Overhead_Decomposition

Nerijus Šatkauskas
nerijus.satkauskas@ktu.lt
Cyber Security Centre of Excellence · Kaunas University of Technology

Prepared within the DigiDefence initiative · project No. 02-002-P-0001