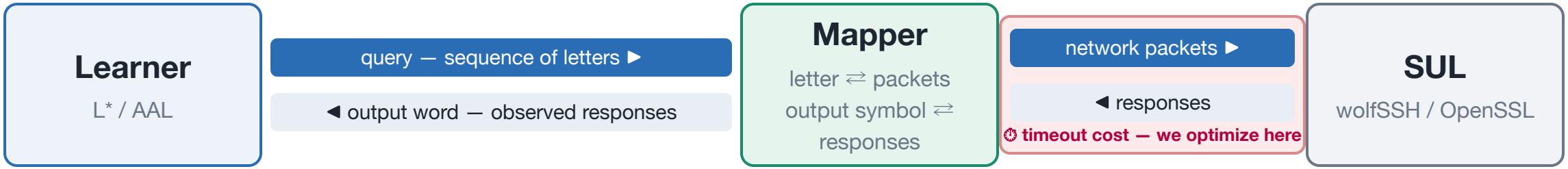
Out-of-Band Optimizations for SSH and TLS State Machine Inference

Van Nam Pham, Olivier Levillain — Télécom SudParis

StANP Workshop, July 2026

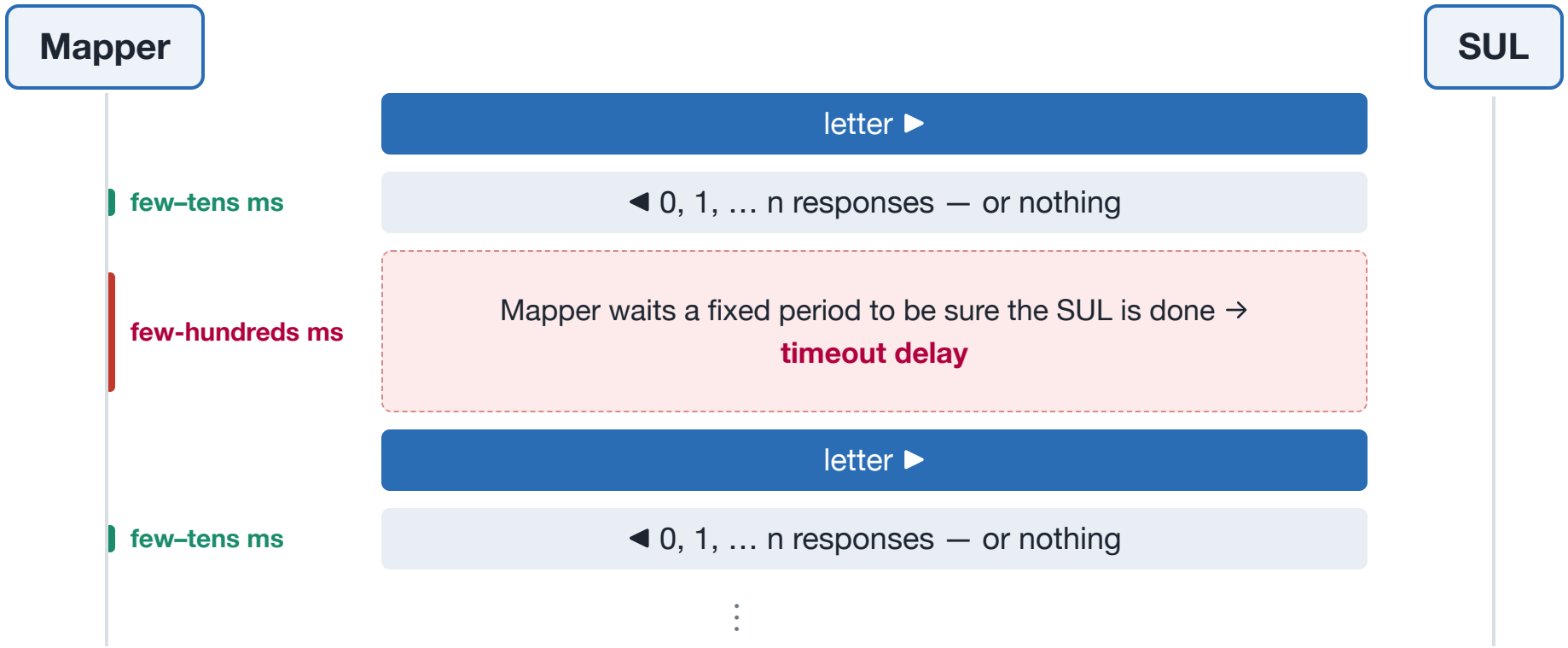
1. Active Automata Learning — and Where We Optimize

- **Learner** asks a *query* (input word = sequence of *letters*)
- **Mapper** translates each letter ⇄ packets, returns an *output word*
- Many queries → Learner infers the **state machine**



… many queries → inferred **state machine**

2. Timeout Problem in Mapper



- Conservative timeout, up to seconds — **repeated per letter**
- **Scope:** two **SCN-team** tools — **ssh-mapper** (wolfSSH) · **pylstar-tls** (OpenSSL)

3. Existing Optimization Approaches: EOF + Known Replies

EOF+KnownReplies *avoids* timeout in specific cases, by exploiting **L* determinism**

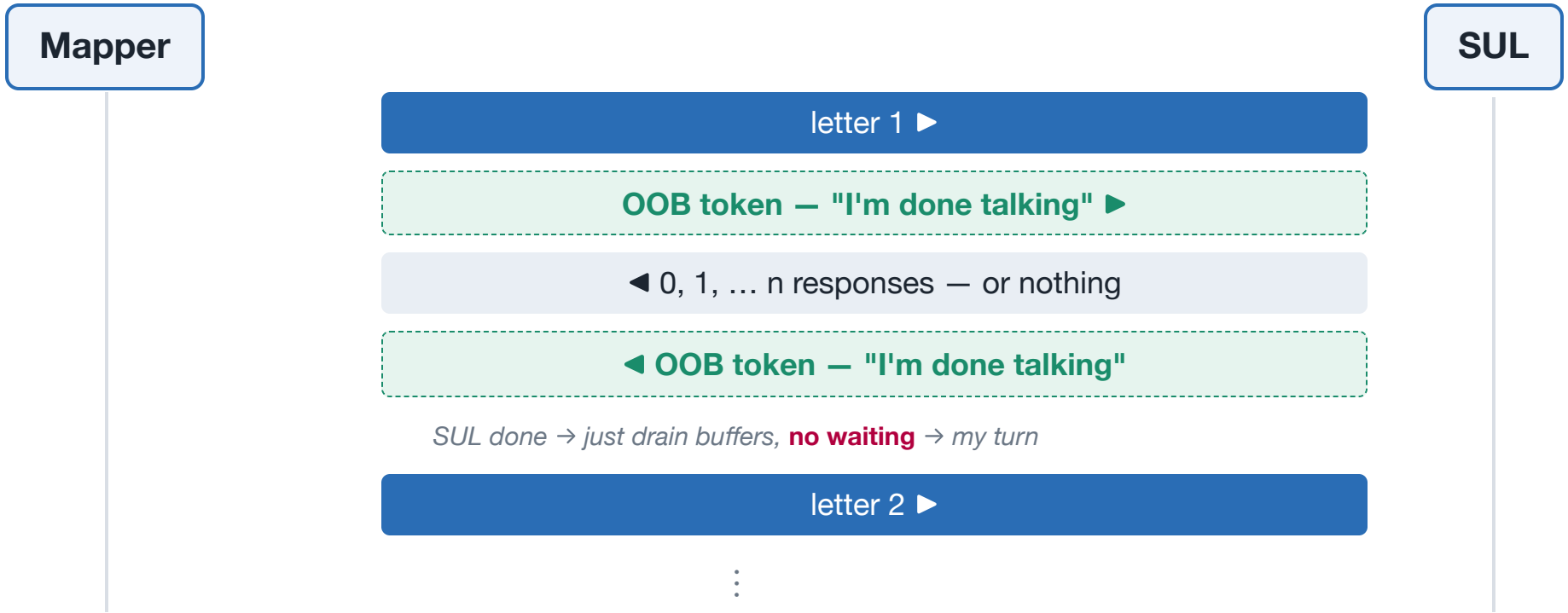
Optimization	Idea	The gap
EOF	SUL sends EOF → every later letter also yields EOF → skip sending them	every letter **before** the EOF → still needs a full timeout
Known Replies	skip the timeout for an already-observed prefix of a new query	the new query's suffix is never observed → still needs a full timeout

> **Gap:** each only cuts timeout partially → **leftover timeouts remain on most queries.**

Our Approach: Out-of-Band Signaling

4. Our Approach: Out-of-band Signaling

- Use **TCP URG** as a side channel for a "*done sending*" signal → the **OOB token**
- Holding it = *allowed to talk* · sending it = "*I'm done, your turn*"



4b. How to Implement the Token Logic

- **Mapper:** modify the SCN-team codebase
- **SUL** (wolfSSH / OpenSSL): inject via `LD_PRELOAD` — no source change

Injecting into the SUL (no source access):

1. `strace` the server while it talks to the Mapper → study its **syscall pattern**
2. Find the point where the server's `send()` of a response **ends** → fire the OOB token there
3. Wrap the server's `recv()` calls → add logic to **receive** the incoming OOB token

> Stays within a small syscall set (`send` / `recv` / `select`) → portable across SULs

4c. Example: a Real wolfSSH ⇄ ssh-client Trace

We `strace` **both sides** of one learning run with 1 second timeout. Transcript (sent/received):

`KexInit/KexInit > KexECDHInit/KexECDHReply+NewKeys > NewKeys/—`
`> ServiceRequest/ServiceAccept > AuthRequestPassword/UserAuthSuccess`

Mapper (client)		SUL · wolfSSH (server)
+0.000 connect()	▶	+0.004 accept()
	⋮	
+1.191 sendto(KexECDHInit)	▶	+1.196 recvfrom(KexECDHInit)
+1.241 recvfrom(KexECDHReply+NewKeys)	◀	+1.238 sendto(KexECDHReply+NewKeys)
… ~1.07 s idle (timeout) …		
+2.323 sendto(NewKeys)	▶	+2.331 recvfrom(NewKeys)
… ~1.07 s idle — waiting for nothing …		*No Replies*
+3.417 sendto(ServiceRequest)	▶	+3.421 recvfrom(ServiceRequest)
+3.437 recvfrom(ServiceAccept)	◀	+3.429 sendto(ServiceAccept)
… ~1.07 s idle (timeout) …		

- Whole sequence runs until the Mapper exits at **+5.78 s** — almost all of it idle
- *Full transcript through to exit in the Appendix*

4d. The Hook: the wolfSSH Server's Syscall Pattern

Server-side trace around the `KexECDHInit / KexECDHReply+NewKeys` exchange:

```
+0.141  recvfrom(sfd) = -1 EAGAIN              # nothing to read yet
+0.142  pselect6([sfd]) <unfinished ...>       # wait on sfd for the next letter
+1.020  ... = 0 (Timeout)                      # ← re-select until data — the ~1 s gap
+1.154  pselect6([sfd]) = 1 (in [sfd])         # data has arrived → Mapper's letter is here
+1.196  recvfrom(sfd) = 5                       # read the letter (KexECDHInit)
+1.203  recvfrom(sfd) = 72                      #   …
+1.238  sendto(sfd, reply) = 664               # send KexECDHReply + NewKeys
+1.239  recvfrom(sfd) = -1 EAGAIN              # done sending
+1.240  pselect6([sfd]) <unfinished ...>       # wait on sfd for the next letter
```

Server read pattern: `recvfrom → EAGAIN → select` on `sfd` — the server enters this **only when it has nothing left to send**, so a `select` call on `sfd` reliably marks "*done sending.*"

4e. The Hook: Two Wrappers via `LD_PRELOAD`

- The `LD_PRELOAD` library uses a `has_token` flag to control the ownership of the OOB token
- `select` **hook** — if server holds the token, send it out
- `recv` **hook** — check for an incoming token first, then read normally

Server trace with OOB, `KexECDHInit / KexECDHReply+NewKeys` exchange:

```
+0.120  recvfrom(sfd) = -1 EAGAIN                          # nothing to read yet
+0.120  pselect6([sfd]) = 1 (in [sfd])                     # data has arrived → Mapper's letter is here
+0.121  recvfrom(sfd, "X", MSG_OOB|MSG_DONTWAIT) = 1       # ← got token: Mapper done → no timeout gap
+0.122  recvfrom(sfd) = 4                                   # read the letter (KexECDHInit)
+0.123  recvfrom(sfd) = 72                                  #   …
+0.152  sendto(sfd, reply) = 664                            # send KexECDHReply + NewKeys
+0.153  recvfrom(sfd) = -1 EAGAIN                           # done sending
+0.153  sendto(sfd, "X", MSG_OOB) = 1                       # ← select hook: pass token back
+0.154  pselect6([sfd]) = 1                                 # wait on sfd for the next letter
```

4f. wolfSSH ⇄ ssh-client Trace after applying OOB optimization

Mapper (client)		SUL · wolfSSH (server)
+0.000 connect()	▶	+0.007 accept()
⋮		
+0.117 sendto(KexECDHInit) **+ sendto OOB "X"**	▶	+0.122 recvfrom(KexECDHInit) **+ recvfrom OOB "X"**
+0.153 recvfrom(KexECDHReply+NewKeys)	◀	+0.152 sendto(KexECDHReply+NewKeys)
+0.162 recvfrom OOB "X"	◀	+0.153 sendto OOB "X"
+0.167 sendto(NewKeys) **+ sendto OOB "X"**	▶	+0.168 recvfrom(NewKeys) **+ recvfrom OOB "X"**
+0.176 recvfrom OOB "X"	◀	+0.174 sendto OOB "X"
+0.178 sendto(ServiceRequest) **+ sendto OOB "X"**	▶	+0.180 recvfrom(ServiceRequest) **+ recvfrom OOB "X"**
+0.185 recvfrom(ServiceAccept)	◀	+0.183 sendto(ServiceAccept)
+0.187 recvfrom OOB "X"	◀	+0.184 sendto OOB "X"

- No idle anywhere — the Mapper exits at **+0.38 s** vs **+5.78 s** baseline (**~15× faster on this query**)
- *Full transcript through to exit in the Appendix*

Experimental Setup & Results

Does OOB actually pay off — and is the learned model still correct?

5. Experimental Setup

- **Targets:**
  - `ssh-mapper` → wolfSSH (18 states)
  - `pylstar-tls` → OpenSSL (4 states)
- **Two configurations** (EOF kept in both — it cuts the *number* of queries):

Config	Coordination
Config 1	EOF + **KnownReplies** (SCN baseline)
Config 2	EOF + **OOB** (ours)

- **Equivalence method:** *Branching Distance*
  - **bdist** values: the deep of equivalence queries
  - Higher = more thorough, exponentially more queries
- **Correctness:** a state machine is a graph
  → Checking **isomorphic** of learned models

5b. Test Scenarios — wolfSSH

Vocabulary	Letters	bdist	Queries	Timeout
Small (3)	KexInit, KexECDHInit, NewKeys	3	~30	0.5 s
Medium (5)	+ UserAuthRequest, ServiceRequest	3	~1,300	0.5 s
Full	all SSH letters	1	~5,000	1 s
Full	all SSH letters	3	~225,000	1 s

- At **0.5 s** the full vocabulary sometimes shows non-determinism → **1 s** for full runs
- *"Full" = all letters currently implemented in* `ssh-mapper` *— see the Appendix for the list*

5c. Results — wolfSSH

Scenario	Timeout	EOF+KnownReplies	EOF+OOB	Speedup
Small (3-letter)	0.5 s	6.67 s	**0.33 s**	**20×**
Medium (5-letter)	0.5 s	~550 s	**34.8 s**	**16×**
Full · bdist=1	1 s	3,991 s (1.1 h)	**209 s**	**19×**
Full · bdist=3	1 s	198,183 s (**~50 h**)	**13,037 s (3.6 h)**	**15×**

> Reduce execution time from days → hours

5d. Results — OpenSSL · generalizes beyond SSH

Scenario	Timeout	EOF+KnownReplies	EOF+OOB	Speedup
OpenSSL · 4 states · bdist=3 · ~41k queries	1 s	47.6 s	**21.2 s**	**2.2×**

- Same **approach**
  - Trace the syscalls, find the hook points
  - The `LD_PRELOAD` library is written a bit differently for OpenSSL
- Smaller speedup: there is less timeouts to remove
- *Full TLS letter list in the Appendix*

6. Summary & Future Work

Achieved:

- **Removed the timeout bottleneck** with OOB signaling → **up to ~20×** (wolfSSH) · **~2×** (OpenSSL)
- **Preserved correctness** — the learned model is provably unchanged (isomorphic)

Next:

- Extend to implementations with a **complex multi-process management** — e.g. OpenSSH, which calls `execve()` across the stages of an SSH connection
- The isomorphism check: **not a complete guarantee** — Needs a baseline run to compare against → look for a stronger, standalone way to verify the learned model

Thank you for listening!

Questions?

Appendix — Full Baseline Trace (1 s timeout, Mapper side)

```
+0.000  connect()
+0.096  sendto(KexInit)
+0.101  recvfrom(KexInit)
        … ~1.07 s idle …
+1.191  sendto(KexECDHInit)
+1.241  recvfrom(KexECDHReply+NewKeys)
        … ~1.07 s idle …
+2.323  sendto(NewKeys)                    (no reply)
        … ~1.07 s idle …
+3.417  sendto(ServiceRequest)
+3.437  recvfrom(ServiceAccept)
        … ~1.07 s idle …
+4.501  sendto(AuthPassword)
+4.709  recvfrom(UserAuthSuccess)
        … ~1.07 s idle …
+5.785  exit
```

5 letters · **~5.78 s total**, > 90 % spent idle waiting out timeouts

Appendix — Full Baseline Trace (1 s timeout, wolfSSH server side)

```
+0.004  accept()
+0.021  sendto(KexInit)           # server sends its own KEXINIT early (RFC)
+0.098  recvfrom(KexInit)
        … select() on sfd for incoming data …
+1.196  recvfrom(KexECDHInit)
+1.238  sendto(KexECDHReply+NewKeys)
        … select() on sfd for incoming data …
+2.331  recvfrom(NewKeys)             (no reply to send)
        … select() on sfd for incoming data …
+3.421  recvfrom(ServiceRequest)
+3.429  sendto(ServiceAccept)
        … select() on sfd for incoming data …
+4.509  recvfrom(AuthPassword)
+4.706  sendto(UserAuthSuccess)
        … select() on sfd for incoming data …
+5.790  exit
```

Appendix — Full OOB Trace (Mapper side)

```
+0.000  connect()
+0.096  sendto(KexInit) + sendto OOB
+0.100  recvfrom(KexInit)
+0.107  recvfrom OOB
+0.117  sendto(KexECDHInit) + sendto OOB
+0.153  recvfrom(KexECDHReply+NewKeys)
+0.162  recvfrom OOB
+0.167  sendto(NewKeys) + sendto OOB          (no reply)
+0.176  recvfrom OOB
+0.178  sendto(ServiceRequest) + sendto OOB
+0.185  recvfrom(ServiceAccept)
+0.187  recvfrom OOB
+0.189  sendto(AuthPassword) + sendto OOB
+0.367  recvfrom(UserAuthSuccess)
+0.370  recvfrom OOB
+0.385  exit
```

Same 5 letters · **~0.38 s total** — no idle gaps (**~15× faster**)

Appendix — Full OOB Trace (wolfSSH server side)

```
+0.004          accept()
+0.021          sendto(KexInit)                 # server sends its own KEXINIT early (RFC)
+0.094→0.100    recvfrom(KexInit)               + recvfrom OOB
+0.103          sendto OOB
+0.118→0.122    recvfrom(KexECDHInit)           + recvfrom OOB
+0.152          sendto(KexECDHReply+NewKeys)    + sendto OOB
+0.168→0.172    recvfrom(NewKeys)               + recvfrom OOB
+0.174          sendto OOB    (nothing to reply, just return the OOB token)
+0.180→0.181    recvfrom(ServiceRequest)        + recvfrom OOB
+0.183          sendto(ServiceAccept)           + sendto OOB
+0.190→0.192    recvfrom(AuthPassword)          + recvfrom OOB
+0.366          sendto(UserAuthSuccess)         + sendto OOB
+0.388          exit
```

OOB token (URG) often arrives **while the server is still reading the letter** · total **~0.39 s** (**~15× faster**)

Appendix — Full SSH Letter Vocabulary (`ssh-mapper`)

- **Transport:** `DISCONNECT` (1) · `IGNORE` (2) · `UNIMPLEMENTED` (3) · `DEBUG` (4) · `SERVICE_REQUEST` (5) · `SERVICE_ACCEPT` (6) · `EXT_INFO` (7) · `KEXINIT` (20) · `NEWKEYS` (21)
- **Key exchange:** `KEXDH_INIT` / `KEX_ECDH_INIT` (30) · `KEXDH_REPLY` / `KEX_ECDH_REPLY` / `KEX_DH_GEX_GROUP` (31) · `KEX_DH_GEX_REQUEST` (34)
- **User auth:** `USERAUTH_REQUEST` (50, password) · `USERAUTH_FAILURE` (51) · `USERAUTH_SUCCESS` (52)
- **Connection:** `CHANNEL_OPEN` (90, session) · `CHANNEL_OPEN_CONFIRMATION` (91) · `CHANNEL_DATA` (94) · `CHANNEL_REQUEST` (98) · `CHANNEL_SUCCESS` (99)

Appendix — Full TLS Letter Vocabulary (`pylstar-tls`)

`TLS13ClientHello` · `TLSChangeCipherSpec` · `TLS13EmptyCertificate` ·
`TLSInvalidCertificateVerify` · `TLSFinished` · `TLSApplicationData` ·
`TLSApplicationDataEmpty` · `TLSCloseNotify` · `NoRenegotiation` ·
`TLSCertificateVerify` · `TLS13Certificate_{crypto_material_name}`