Improving Secure Connection Establishment

Networked Systems (H)
Lecture 4

Lecture Outline

- Limitations of TLS v1.3
  - Slow Connection Establishment
  - Metadata Leakage
  - Protocol Ossification
- QUIC Transport Protocol
  - Performance, security, and avoiding ossification
  - Unified protocol handshake
  - Reliable multi-streaming transport

Limitations of TLS v1.3

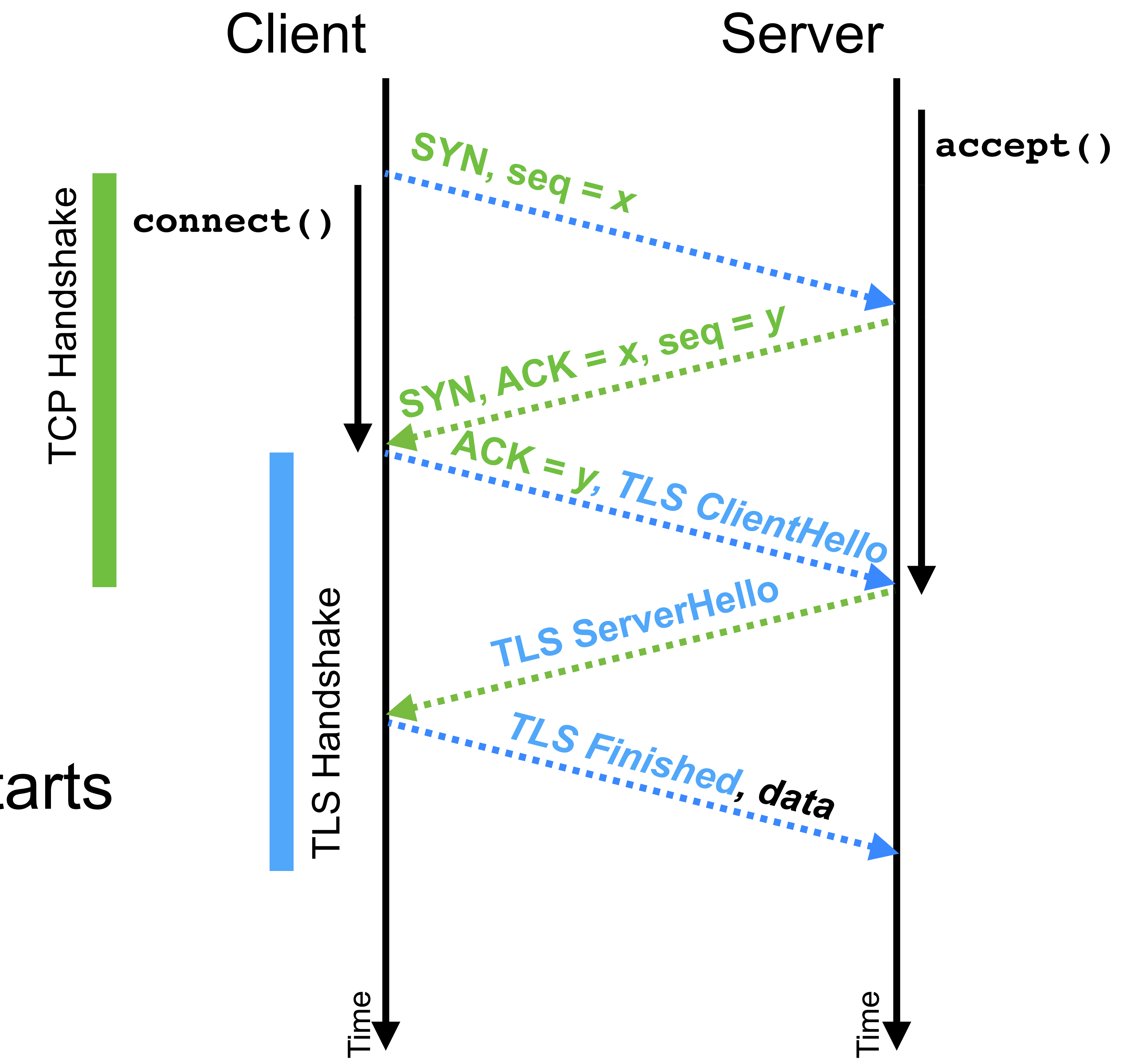
- Slow Connection Establishment
- Metadata Leakage
- Protocol Ossification

Limitations of TLS v1.3

- TLS v1.3 is a tremendous success
  - **Significant security improvements** compared to TLS v1.2
    - Removed support for older and less secure encryption and key exchange algorithms
    - Removed support for secure algorithms that have proven difficult to implement correctly
  - Some **performance improvements** to the initial handshake and with 0-RTT mode
- Despite this, TLS v1.3 has some limitations that are hard to fix
  - Connection establishment is still relatively slow
  - Connection establishment leaks potentially sensitive metadata
  - The protocol is ossified due to middlebox interference

TLS v1.3 Connection Establishment Performance (1/2)

- TCP connection established as usual:
  - **`SYN`** → **`SYN+ACK`** → **`ACK`**
- TLS handshake protocol runs inside TCP connection:
  - TLS **`ClientHello`** sent with final **`ACK`**
  - TLS **`ServerHello`** sent in response
  - TLS **`Finished`** message concludes, and carries initial secure data record
- First data sent 2x RTT after connection establishment starts
- Earliest response received 3x RTT after connection establishment starts



TLS v1.3 Connection Establishment Performance (2/2)

- Average web page comprises 1.7 MB of data, fetched as 69 HTTP requests, using 15 TCP connections
- 83% of HTTP requests run over TLS
  https://httparchive.org/reports/page-weight

- **Enormous amount of time wasted, waiting for TCP and TLS connection establishment handshakes**

- Can we speed up TLS connection setup?
  - 0-RTT Connection Reestablishment – speed-up connections to known servers
  - Concurrent TCP and TLS handshake – speed-up connections to all servers

Destination	Average RTT (ms)
London	72.5
New York	153.3
Los Angeles	221.2
Sydney	381.2

RTT measurements (ping times) from residential site in Glasgow

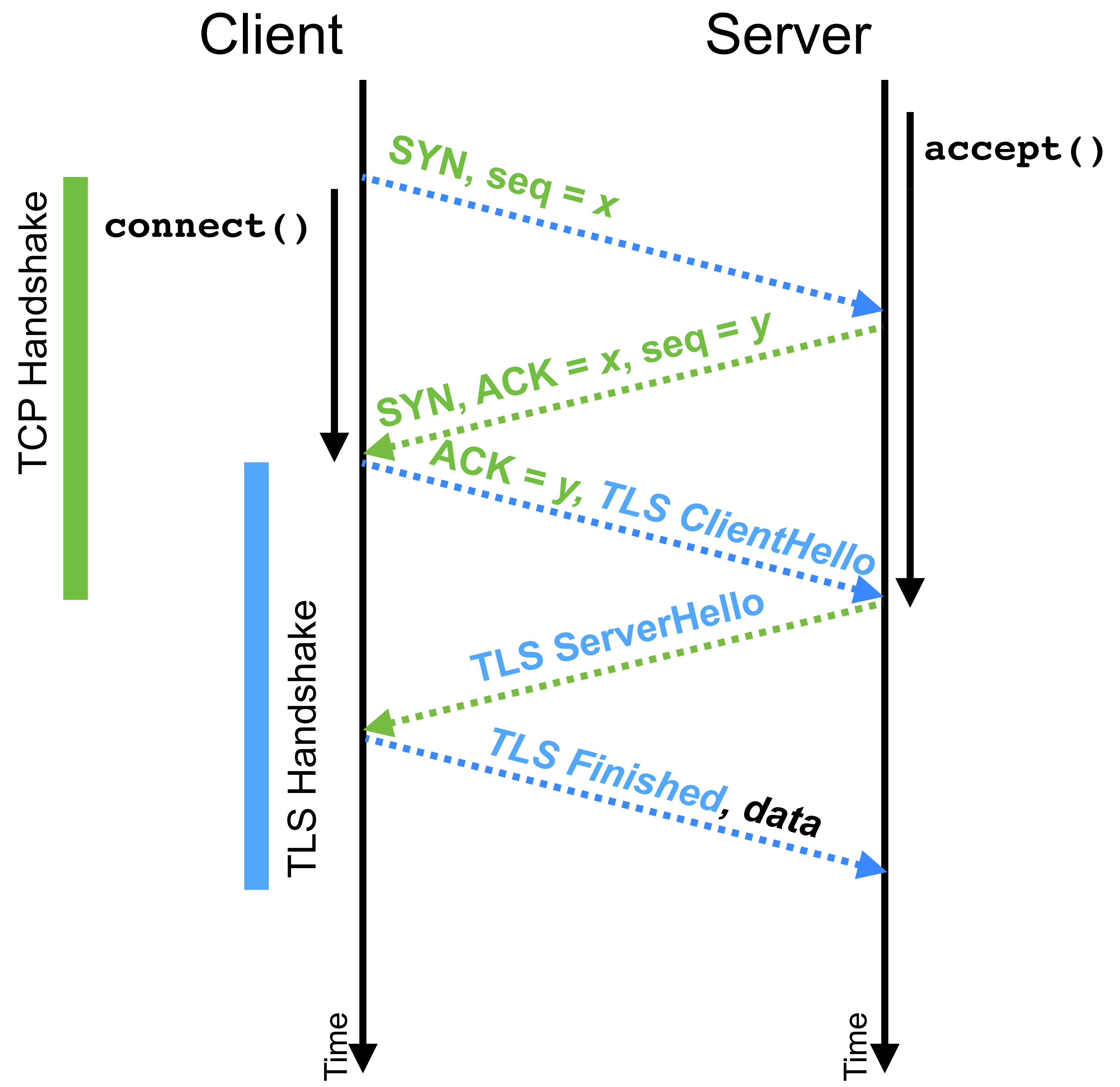
0-RTT Connection Reestablishment (1/4)

- Common to connect to a previously known TLS server – is it possible to shortcut the connection establishment in such cases?

- Need to understand:
  - What is the role of the TLS handshake?
  - How to encrypt initial data?
  - What are the potential risks?

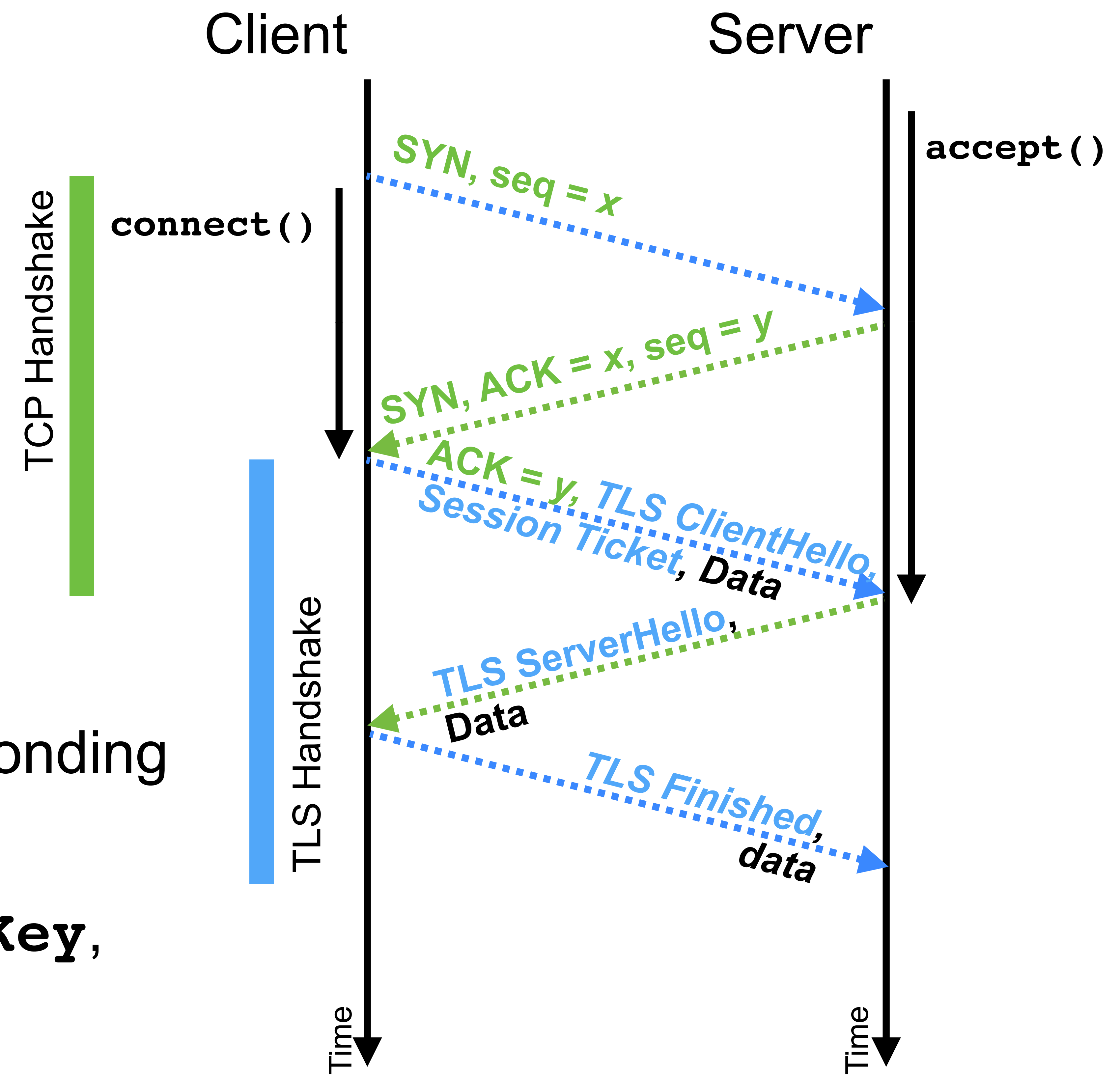
0-RTT Connection Reestablishment (2/4)

- **What is the role of the TLS handshake?**
  - Uses public key cryptographic techniques to establish an **ephemeral session key**, used to encrypt the data
    - The `ClientHello` and `ServerHello` are used to exchange material used to derive a session key – *using ECDHE key negotiation*
    - The session is ephemeral – different for each connection; derived from the public keys and a random value
    - The ephemeral session key provides **forward secrecy** – each connection has a unique key;
      if the encryption key for one session leaks, it doesn't help an attacker break other sessions
  - Retrieve the server's certificate, allowing the client to authenticate the server
    - The `ServerHello` contains the certificate



0-RTT Connection Reestablishment (3/4)

- **How to encrypt initial data?**
  - Cannot negotiate ephemeral session key for initial data → relies on data exchanged in the handshake
  - Reuse a **pre-shared key** agreed in previous TLS session
- In a previous TLS connection:
  - Server sends a **`PreSharedKey`** with a **`SessionTicket`** to identify the key
- When reestablishing a connection:
  - Client sends **`SessionTicket`**, data encrypted using corresponding **`PreSharedKey`**, along with **`ClientHello`**
  - The server uses **`SessionTicket`** to find saved **`PreSharedKey`**, decrypt the data
  - **`ClientHello`** and **`ServerHello`** complete usual key exchange; data sent with **`ServerHello`** and later protected using ephemeral session key → no additional round-trips due to TLS

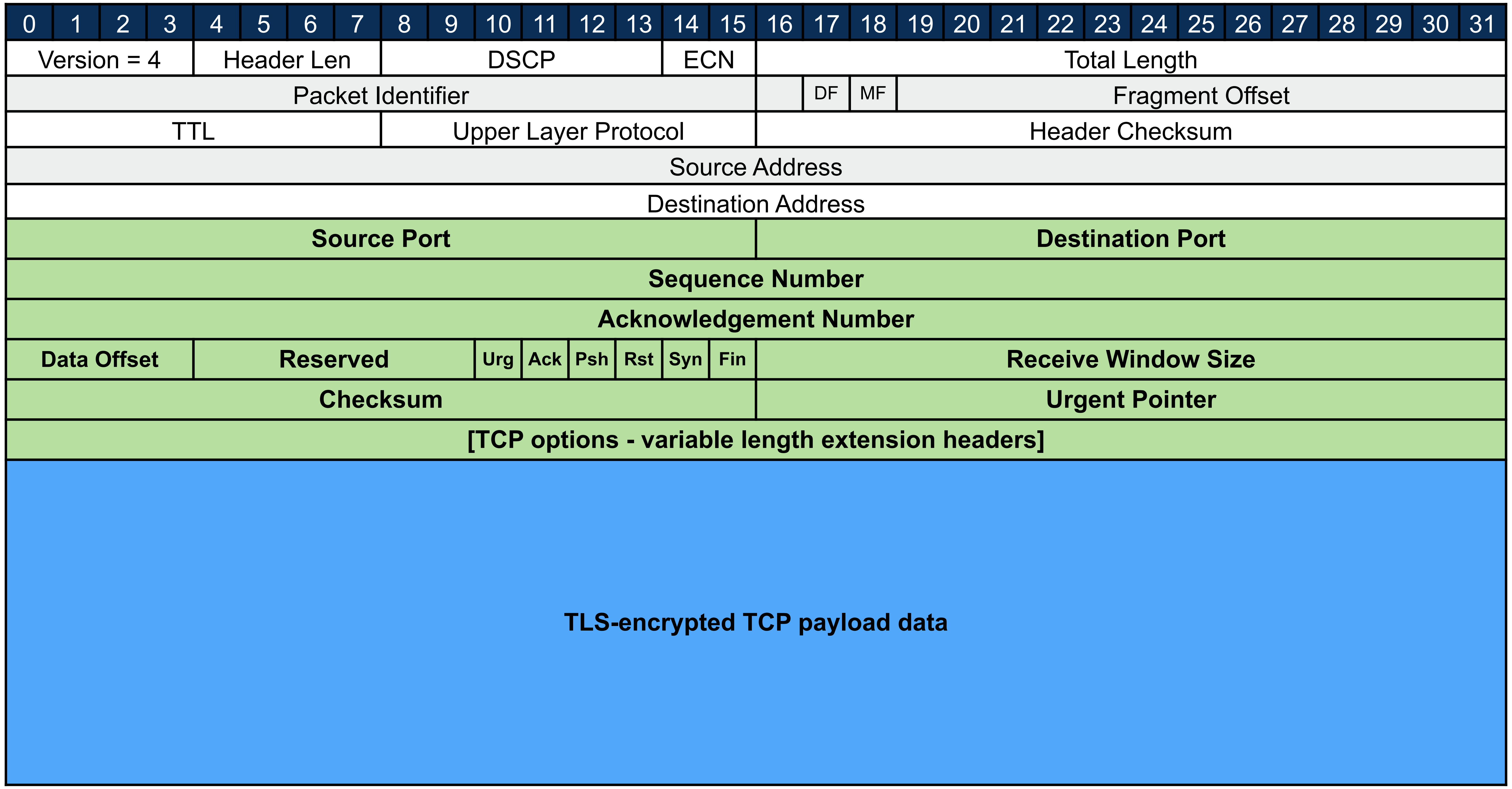


0-RTT Connection Reestablishment (4/4)

- **What are the potential risks?**
  - 0-RTT data sent with `ClientHello` using a `PreSharedKey` is not forward secret
    - Use of `PreSharedKey` links TLS connections – if session where `PreSharedKey` is distributed is compromised, 0-RTT data sent using that key in future connections will also be compromised
  - 0-RTT data sent with `ClientHello` using a `PreSharedKey` is subject to replay attack
    - The 0-RTT data is accepted during TLS connection establishment
    - If on-path attacker captures and replays the TCP segment with the `ClientHello`, `SessionTicket`, and data protected with the `PreSharedKey`, that data will be accepted by the server again
      - The server will respond to the replay, trying to complete the handshake – this might fail
      - But – by then, the data will have been accepted
    - Ensure 0-RTT data is **idempotent** to avoid this risk

- **Be very careful using 0-RTT data in TLS v1.3** – trades performance for safety

TLS v1.3 Metadata Leakage (1/2)



IP exposes addresses

TCP exposes port numbers and connection metadata

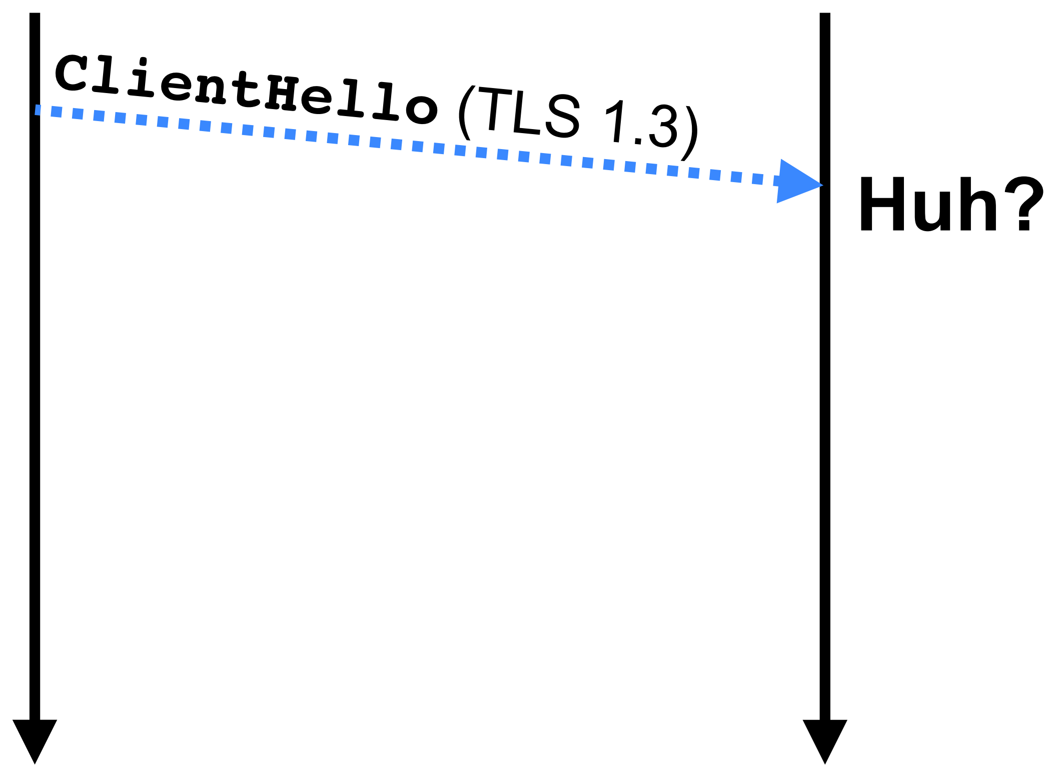
Is there a privacy concern?

TLS v1.3 Metadata Leakage (2/2)

- When TLS is used with HTTPS, **`ClientHello`** includes the Server Name Indication (SNI) extension
  - Identifies requested site, so server knows what public key to use in **`ServerHello`**
    - Required to support shared hosting, with multiple websites on one server
    - Has to be unencrypted – sent before session keys are negotiated
    - Can't encrypt with **`PreSharedKey`**, since that's provided by server, and goal is to select the server
- **A privacy concern with TLS v1.3**
  - See https://datatracker.ietf.org/doc/draft-ietf-tls-esni/ for work-in-progress attempt to resolve

TLS v1.3 Protocol Ossification (1/3)

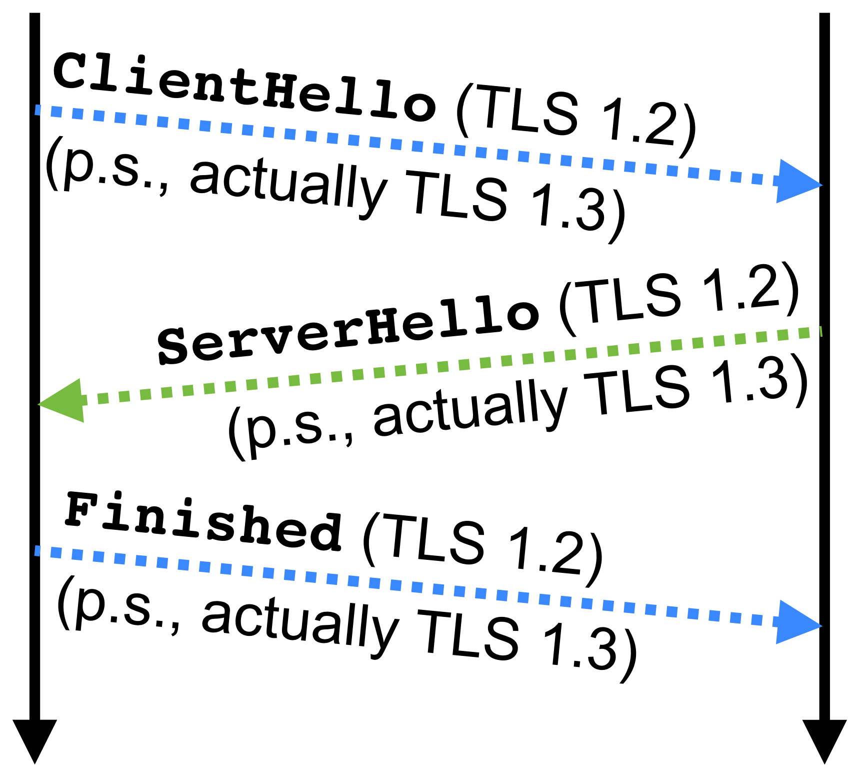
- TLS is widely implemented, but many poor quality implementations:
  - Some TLS servers fail if **`ClientHello`** uses unexpected version number, rather than try to negotiate older version
  - Some firewalls block connections if **`ClientHello`** is structured differently to that used by TLS 1.2 and earlier, even if TLS 1.3 is signalled



- Original design of TLS 1.3 changed **`ClientHello`**
  - Updated the version number (1.2 → 1.3)
  - Removed some now unused header fields
- Measurements showed this caused ~8% of TLS 1.3 connections to fail

TLS v1.3 Protocol Ossification (2/3)

- Later versions of TLS 1.3 changed the design to work around these bugs
  - Version number in `ClientHello` says **TLS 1.2**; unused header fields present with dummy values; extension header to `ClientHello` signals actual version
  - (Similar changes in `ServerHello`)
  - When TLS 1.3 client talks to TLS 1.3 server, version negotiated in extensions
  - When TLS 1.3 client talks to TLS 1.2 server, extension ignored and TLS 1.2 is negotiated



- **Protocol ossification is a significant concern**
  - TLS is not the only protocol to include such workarounds
  - Widely deployed faulty implementations constrain design of most protocols

How to Avoid Protocol Ossification?

- Ossification happens when extension mechanisms, or allowed flexibility, are not used
  - TLS 1.3 was released ten years after TLS 1.2
  - Allowed products to be built and deployed that didn't do version negotiation correctly, since no new versions to negotiate
  - Allowed products to be built that relied on the presence and order of fields in `ClientHello`, since all implementations included the same fields in the same order
- **Generate Random Extensions And Sustain Extensibility** (GREASE)
  - If the protocol allows extensions, send extensions
  - If the protocol allows different versions, negotiate different versions
  - **Do this even if you don't need to → "use it or lose it"**
    - Send meaningless dummy extensions that are ignored
    - Change the version number to prove you can

Limitations of TLS v1.3

- TLS v1.3 is a significant improvement on prior versions: faster and more secure
- TLS v1.3 runs within a TCP connection:
  - Must wait for TCP connection establishment
  - Some metadata leakage
- Implementations of TLS are ossified and hard to extend