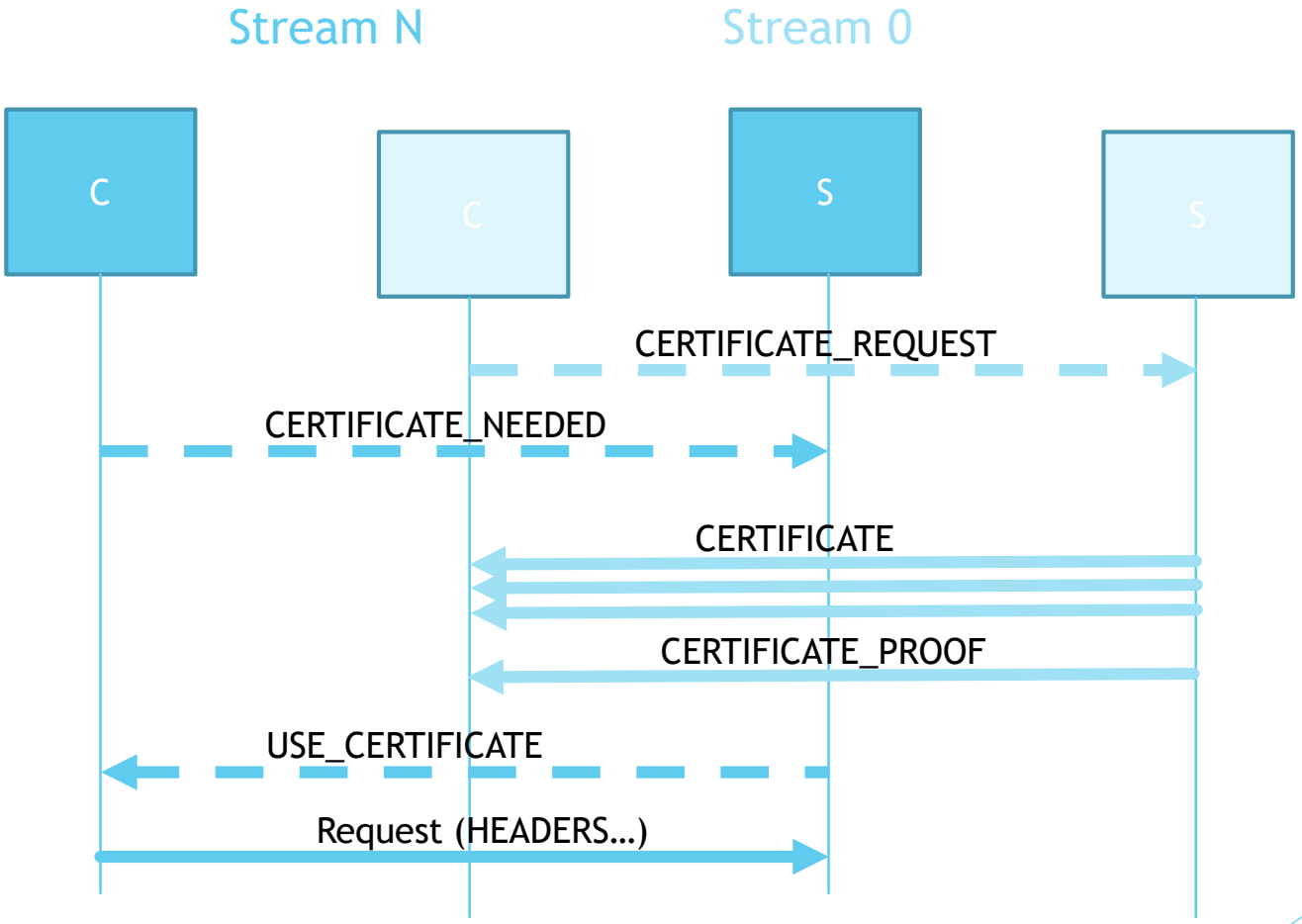
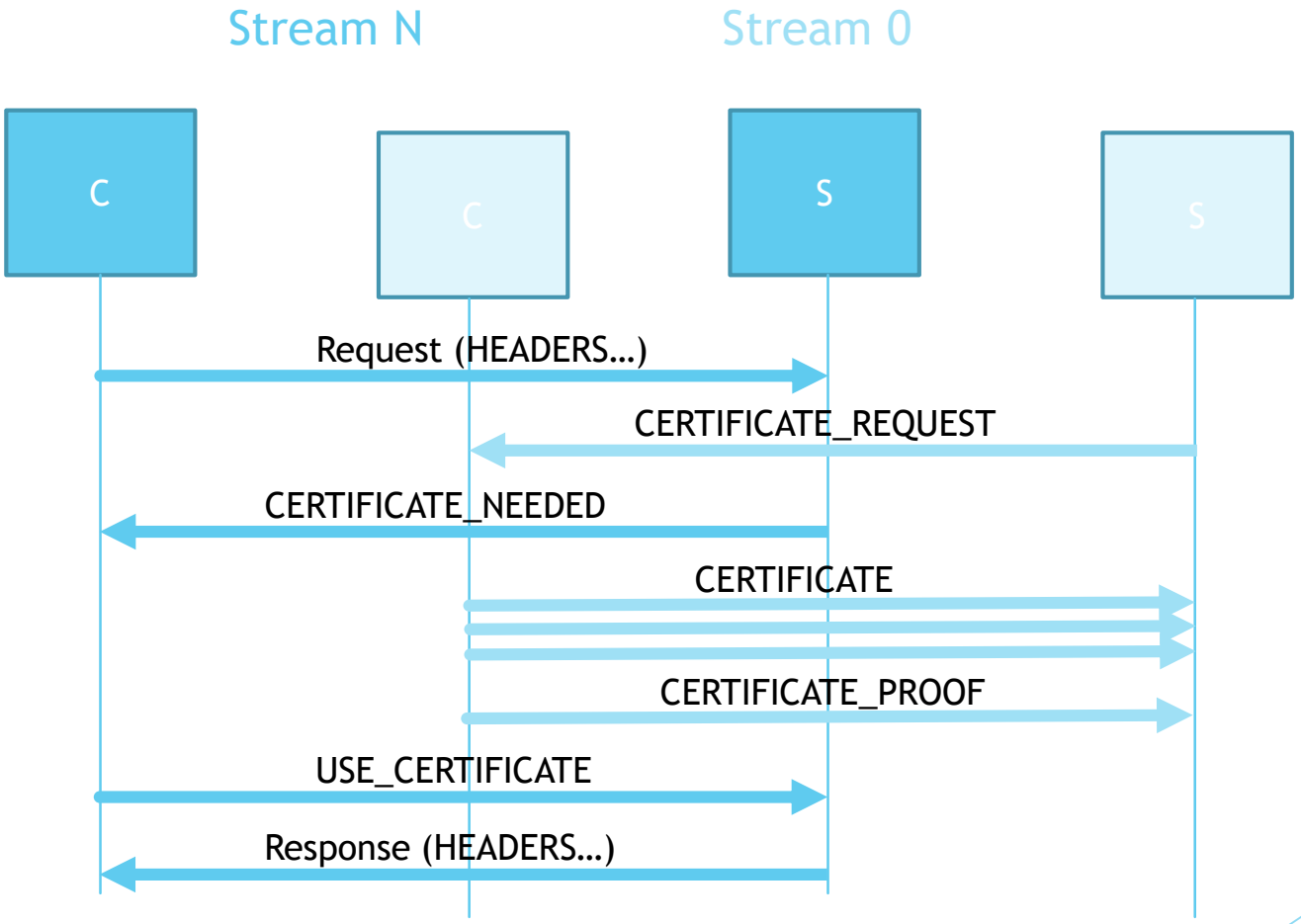
Secondary Certificates

Server Certificate



Client Certificate



Why do certs in HTTP?

- Multiplexing and TLS
  - TLS: One server identity, one client identity
    - Unless this changes…?
  - HTTP: Many requests, possibly distinct identities
- Multiplexing and client certs
  - HTTP/2 prohibits renegotiation
  - Even if it didn't, most TLS 1.2 implementations can't do renegotiation while application data flows
  - TLS 1.3 might improve this
    - Still have to bind HTTP requests and TLS CertificateRequests
- Multiplexing and server certs
  - HTTP/2 connection coalescing only works if the server cert has all possible names
  - Forces servers to use mega-certs

Changes since Buenos Aires

- Merged client and server drafts, per WG feedback
- Permit unsolicited offers of certificates
  - Helps the AUTOMATIC_USE case substantially
  - Requires declaring acceptable signature methods in SETTINGS
- Certificates can include “supporting data”
  - OCSP
  - Signed Certificate Timestamp
  - Possible future application: DNSSec for TLSA, A, AAAA, etc. records
- Call for Adoption

Key critiques

- Memory explosion – have to persist certificates forever!
  - Might be good to allow a peer to indicate it has "forgotten" a certificate
- Not everything is a cert!
  - PSK, etc.
  - Can be made to look cert-like, or could add a credential-type field
- Client/server symmetry is overkill!
- Insufficient binding of proof to certificate!
  - Defer to our crypto brethren to make this better
- Clients shouldn't have to pick between AUTOMATIC_USE and losing 1 RTT!
  - Allow unsolicited USE_CERTIFICATE?
  - Departs further from the TLS semantics

Biggest Critique

- Currently uses a 32-bit HTTP/2 SETTINGS value to convey signature methods and supplemental data types
  - 16-bit bitmask for each
- Missing way to convey other properties, like supported certificate types
- Severely constrains future expansion and experimentation
- Requires re-defining all currently-interesting values into a new registry

- **Why can't we just use the values TLS has already defined for such things?**

Because RFC 7540 said so!

EXTENDED_SETTINGS

Enough for everyone?

- Some uses need much more than 32 bits
  - Certificates would ideally use an array of HashAndSignatureAlgorithm values from the TLS registry
  - Also should convey acceptable certificate types
- Some uses need fewer than 32 bits, or none:
  - Is anyone *actually* using a 4GB HPACK header table?
  - SETTINGS_ENABLE_PUSH: “Any value other than 0 or 1 MUST be treated as a connection error of type PROTOCOL_ERROR.”
  - draft-kerwin-http2-encoded-data: “Any value other than 0 or 1 MUST be treated as a connection error of type PROTOCOL_ERROR.”
  - Others?
- Exactly 32 bits is too constrained

Payload layout

SETTINGS

EXTENDED_SETTINGS

EXTENDED_SETTINGS vs. vanilla SETTINGS

- Borrows heavily from RFC7540 SETTINGS text
- Values are length-prefixed blobs
  - Currently static 16-bit length; could do something variable if desired
- ACK works differently:
  - Sender of EXTENDED_SETTINGS sets flag if ACK is desired
  - Recipient sends back EXTENDED_SETTINGS_ACK listing the values which it understood from the EXTENDED_SETTINGS frame
    - If it received the frame, but didn't understand any of the values, the ACK is sent but empty
- Never-seen is a different value than zero
  - Implicitly true in SETTINGS as well; RFC 7540 defines some initial values which can't be expressed on the wire.
- Possible future optimization for Boolean values
  - Reserve a bit somewhere, use if length=0

Should we do this?

- Subjectively better than using the current bitmask approach
- Strictly better than defining a CERT_SETTINGS frame purely for the certificates draft
- Negligible improvement in chattiness for small things to migrate
  - Even worse if only 1-2 things ever use it and you're sending EXTENDED_SETTINGS only for one flag