Resetting and Closing Streams

Marten Seemann
IETF 116, Yokohama

Problem Statement

1. The WebTransport Use Case: "I really need to get the Session ID through"
2. The Relaying Proxy: "Oops, the upstream server died"
   a. But the proxy wants to send the bytes it has received and signal an error
   b. See https://github.com/quicwg/base-drafts/issues/3300

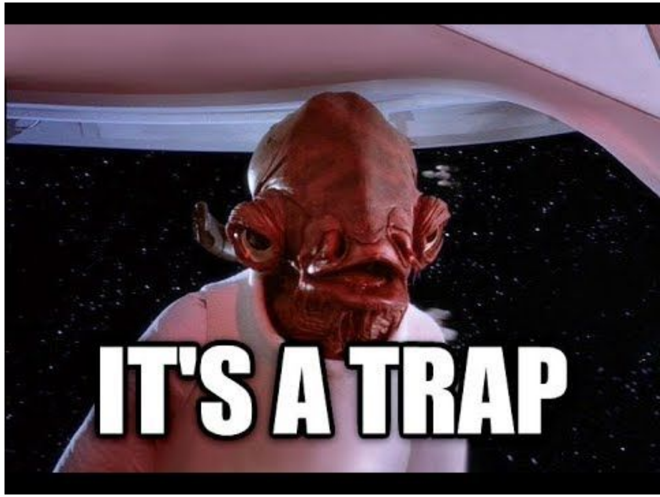
Victor's proposal: RESET_STREAM_WITH_PAYLOAD

- https://github.com/marten-seemann/draft-seemann-quic-reliable-stream-reset/pull/2
- basic idea: add some payload to a RESET_STREAM frame
  - doesn't need to correspond to any data sent on the stream
- requires changing both receiver and sender QUIC stream API
- cannot solve the relaying use case

```
RESET_STREAM_WITH_PAYLOAD Frame {
  Type (i) = 0x73,
  Stream ID (i),
  Application Protocol Error Code (i),
  Final Size (i),
  Application Protocol Payload Size (i),
  Application Protocol Payload (..),
}
```

RELIABLE_RESET_STREAM

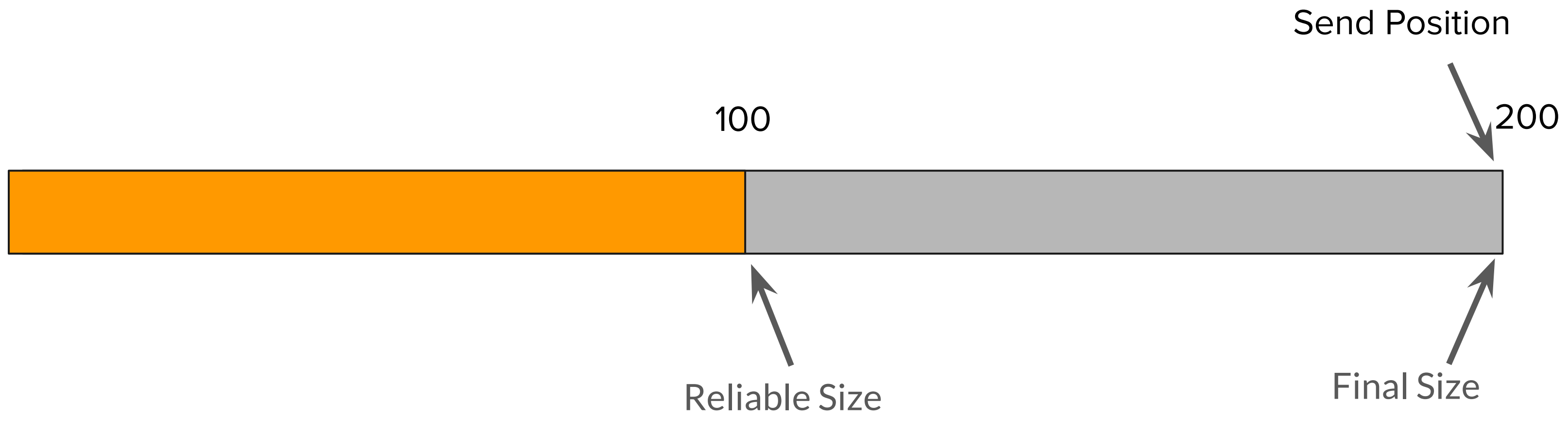
- RELIABLE_RESET_STREAM: the name is confusing



```
RESET_STREAM Frame {
  Type (i) = 0x04,
  Stream ID (i),
  Application Protocol Error Code (i),
  Final Size (i),
}
```

```
~~RELIABLE_RESET_STREAM~~ Frame {
  Type (i) = 0x72,
  Stream ID (i),
  Application Protocol Error Code (i),
  Final Size (i),
  Reliable Size (i),
}
```

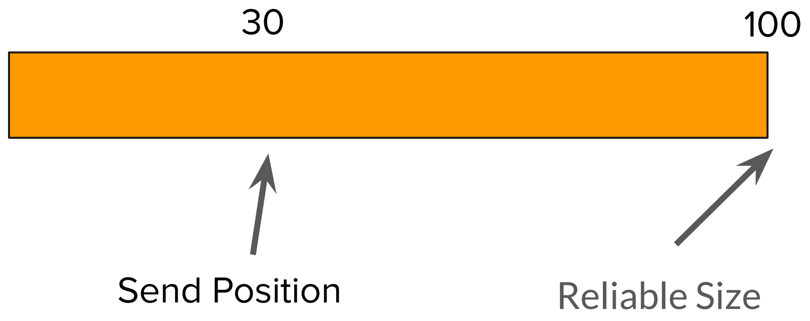
Sometimes it looks like a RESET!



close stream at

```
{
    Reliable Size: 100,
    Final Size: 200,
}
```

But it's actually more like a FIN!



Implementation Strategy:

don't send STREAM_CLOSE before having sent all the reliable bytes (like a FIN)

close stream at

```
{
    Reliable Size: 100,
    Final Size: 100,
}
```

Stream API (sending side)

```
class SendStream:
    ...
    def write():
        // write data to the stream buffer
    def commit():
        // commit to transmit all bytes written so far
```

Receiver Side

Receiver: don't act on the Reliable Size before having deliver all the reliable bytes

Just like you'd do for a FIN bit!

Stream API (receiving side)

~~RELIABLE_RESET_STREAM~~

Maybe STREAM_CLOSE?

Implementation Status

- quic-go: ~80 LOC for the stream state machine changes
- quicly: ~50 LOC for the stream state machine changes
- quic-go and quicly successfully interop!

Next Steps

Adoption?