Ending Connections

QUIC Interim 2017-06, Paris

What do we want?

A connection ends when either **end**point decides to stop

Ideally, both endpoints agree that a connection has ended

... at more or less the same time

What signaling do we need to support this?

Enter the path signals problem

What do we say to a path?

Just like with Stateless Reset, we can hide any signals, or

... allow signals to leak

Explicit signals could use separate mechanisms

Taxonomy

Sudden loss of connectivity (no remaining viable paths)

Endpoint crash (see Stateless Reset discussion)

Fatal errors

Idle timeout

Graceful shutdown

Total Loss of Connectivity

Can't signal anything by definition

HTTP talks about this scenario in some depth

Might include advice on

- ... detecting the condition (tricky, could involve timers)
- ... what to do with application state if this happens

Might be some discussion of spoofing detection

- ... e.g., the implications of ICMP unreachable

Endpoint Crash

This is a little like loss of connectivity for the surviving peer

No signal possible

Recovery is most relevant

- Stateless Reset (see previous) covers server recovery
- Client is unlikely to see packets once it recovers

Some assumptions here; improvements welcome

Fatal Errors

One or other peer can't continue for ***reasons***

This might be protocol violations by a peer, or DoS

The terminating endpoint needs a signal

That signal should be reliable

Currently, CONNECTION_CLOSE

- … suggestion open to use Stateless Reset (see earlier)
- … alternatively to restrict this to errors only, read on…

Graceful Shutdown

Proposal Overview

Application protocol manages its own shutdown

Application tells the transport to close

Transport goes straight to a TIME_WAIT analogue

No explicit transport-level signal

Idle timeouts catch everything else

Idle Timeout

X seconds pass without sending or receiving a packet

- Note: streams could still be open

The connection is gone

Allows indecisiveness in connection management

- Can’t decide if you might need this again, no worries!
- Saves radio resources

Failsafe method for detecting loss of connectivity

Why Force Applications to Manage Shutdown?

The transport doesn't know the application

e.g., PUSH_PROMISE "uses" stream IDs before opening the associated streams

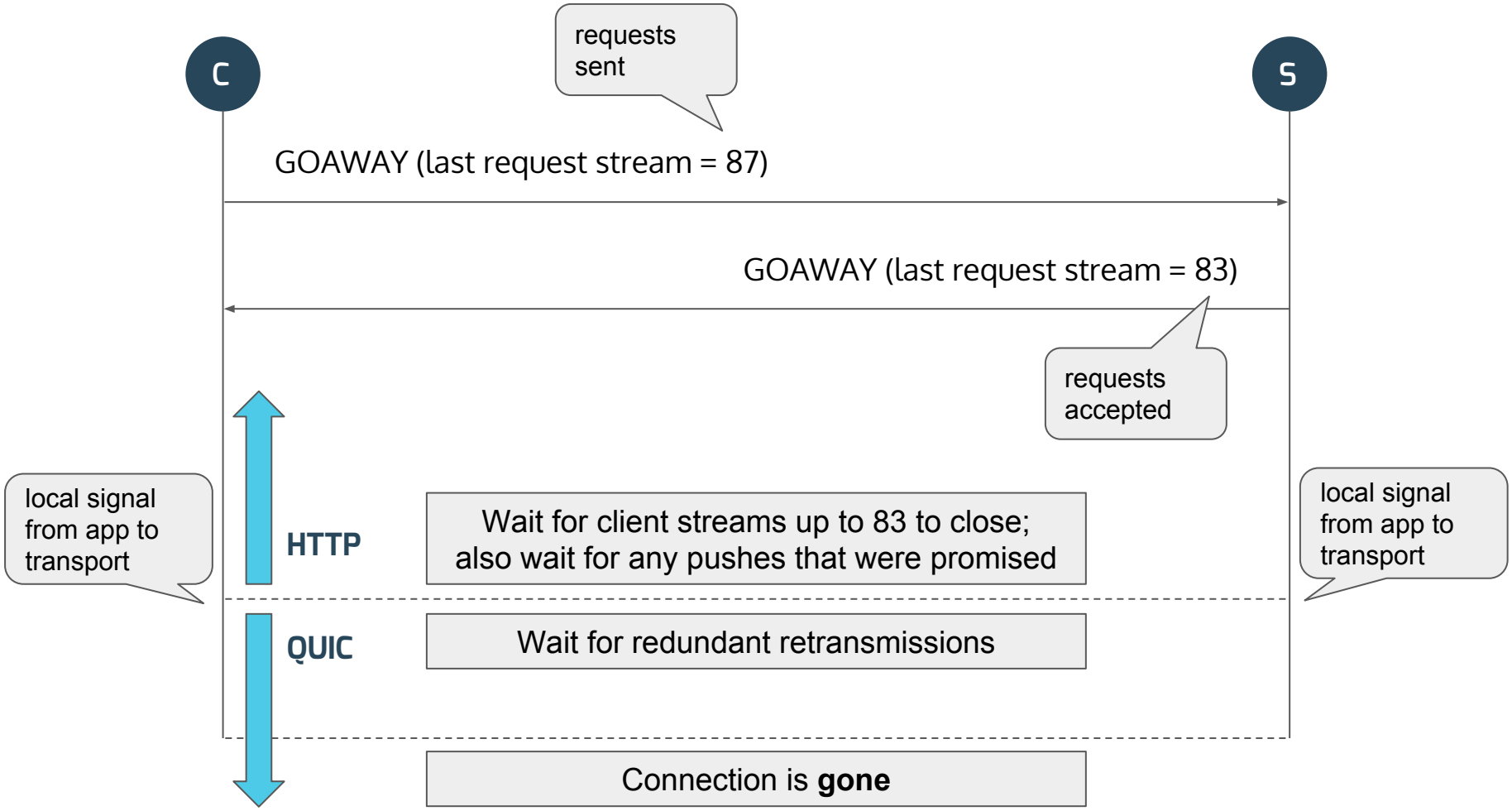
Applications have different requirements for shutdown

e.g., HTTP cares about what can be retried safely

e.g., RTP cares about how many participants are present

How HTTP Might Shut Down

WARNING! STRAWMAN



Alternative: Generalized Design

Application decides on the last frame number

Application tells QUIC, QUIC sends a GOAWAY frame

QUIC waits for all streams up to that number to close

To deal with applications that don't know the last stream ID

- ... like HTTP, with server push
- allow multiple GOAWAY frames
- start at STREAM_MAX and reduce when you know better