Transport Services (TAPS)

Modern Interfaces to Future Transports

Brian Trammell, IETF 109 ~~Bangkok~~
Friday 20 Nov 2020 — 07:30 UTC

the case for TAPS in three observations

Transport **deployment has been slowed** by the hard binding between transport behaviors and concrete protocols (SCTP).

Runtime binding would allow **dynamic transport selection** based on OS support and interface/network conditions (NEAT).

Dynamic selection is only half the battle: applications have to sit atop an **abstraction over *all* supported transport behavior** (Post Sockets, Network.framework) or be constrained to the least common denominator behavior (SOCK_STREAM)

the story so far

2014: chartered for the dynamic transport selection problem

2014-17: RFC 8095 catalogs transport layer behaviors to understand basis of dynamic selection

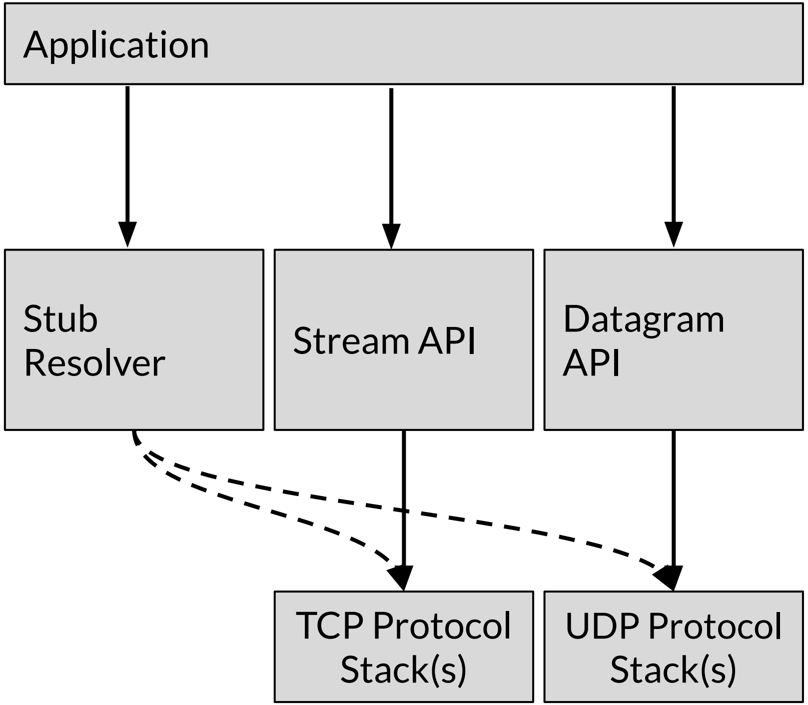
> During discussion of this document, it becomes clear that dynamic protocol selection needs an abstract API.

2016-2018: groundwork for abstract API: RFCs 8303, 8304, 8923

2018-present: Architecture, Interface, and Implementation documents to define an abstract API for future transports.

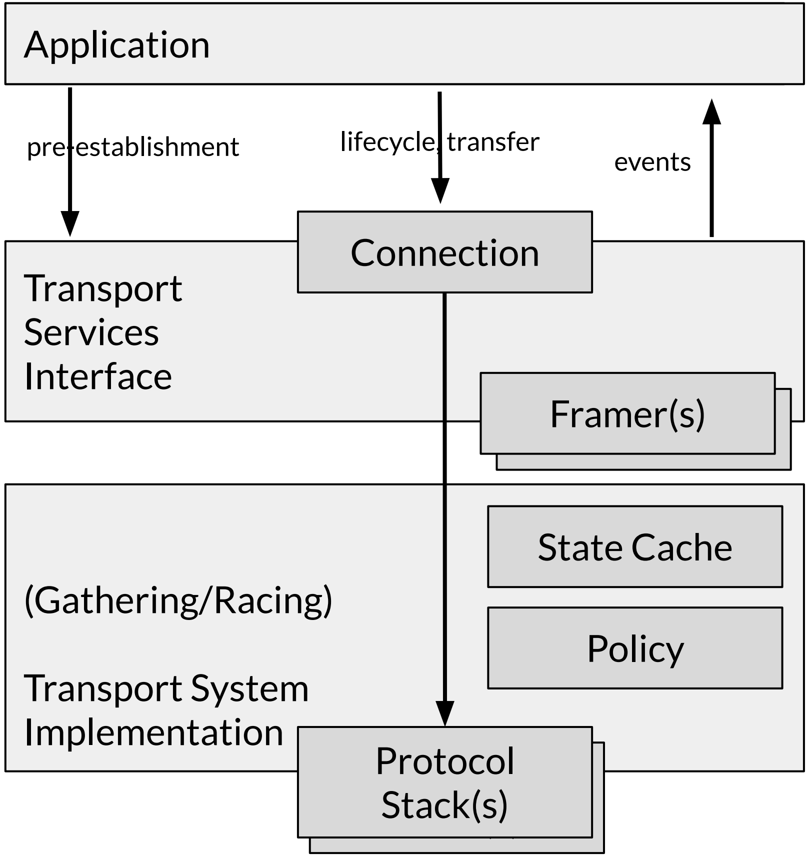
(2018, Network.framework, production TAPS-like API, announced at WWDC)

Sockets



Application
Stub Resolver
Stream API
Datagram API
TCP Protocol Stack(s)
UDP Protocol Stack(s)

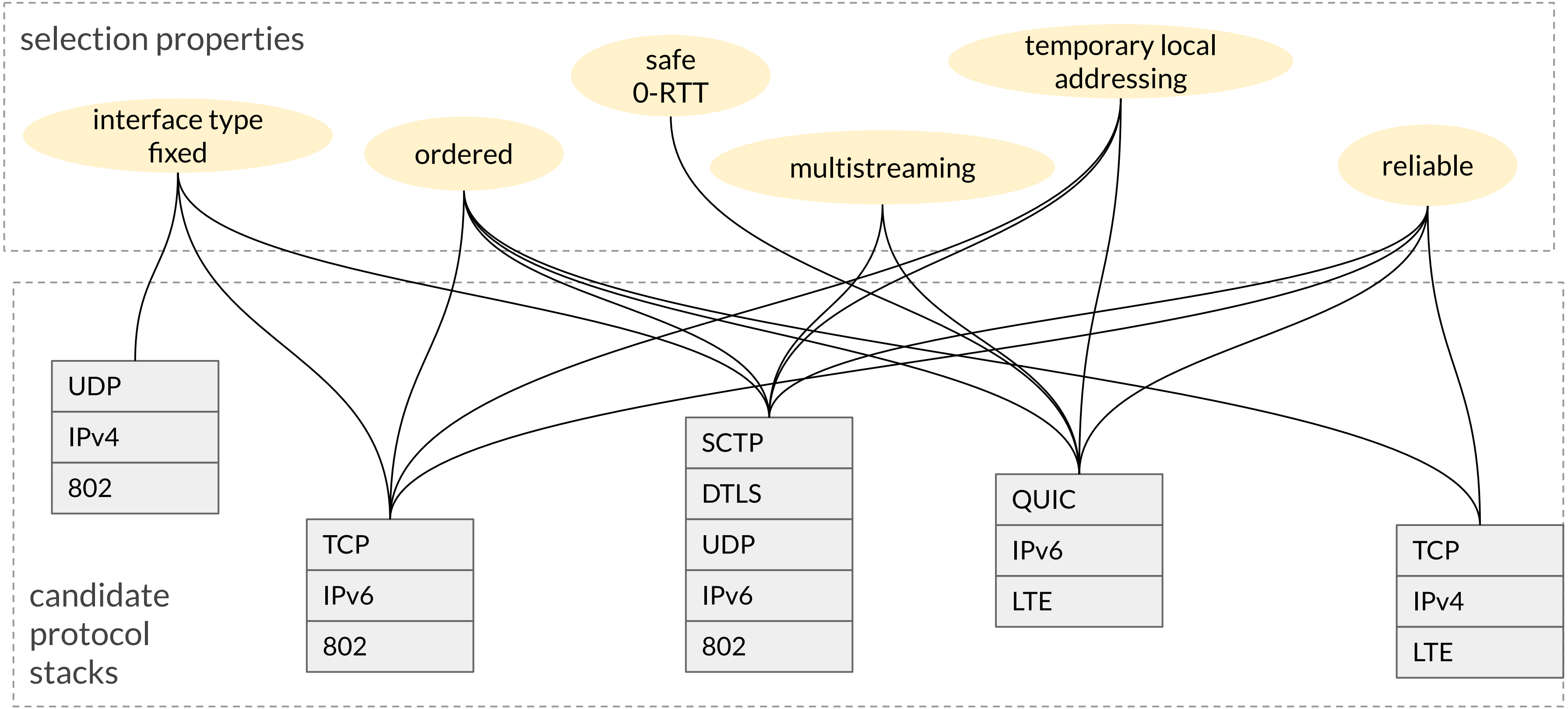
The Transport Services Architecture



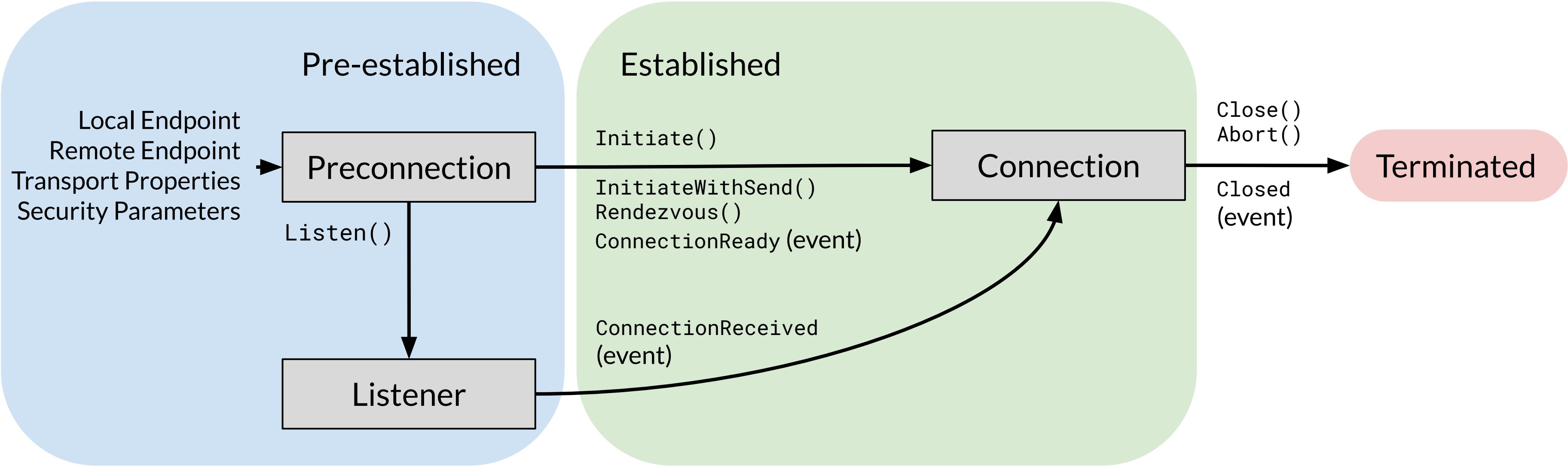
Some Abstract Interface Features

- Designed to allow the implementation considerable latitude for **optimization / parallelization:**
  - Preconnections represent the properties of connections (protocol stacks + interfaces) desired by the application, and can be raced to find the best acceptable connection.
  - Resolution and connection are not explicitly separate.
- Sending/receiving are **asynchronous/event driven**.
- Connection interface can **abstract multistreaming** / multipath protocols.
- Framers match impedance between **streams and message** oriented interactions.

Transport Stack Selection



Connection Lifecycle



Why an *abstract* interface?

- The "shape" of the interface to the transport layer is more important than the implementation details
  - IETF work unlikely to change programming language idioms
  - Swift vs Haskell vs Rust not much fun to discuss in a mic line
  - writing to the same/similar concepts is a big win, though
- Incentivize application development further down the stack to be l**ess synchronous and less imperative** about its interaction with the transport layer.
  - frees up transport layer innovation without future application rewrites.

Transport layer?

I thought this was the QUIC area.

- QUIC is a modern, deployable, extensible transport...
  - It'll do everything every other transport can do, if all proposed features are defined and implemented.
  - Why transport protocol agility, if everything will be QUIC?

- ...but it doesn't have a proper interface.
  - focus on the Web → implementations often tied to H3, interface between application and transport is blurry and internal.
- (note: this is seen as a feature)

We're nearly done...

https://github.com/ietf-taps/api-drafts:

- Architecture + Interface nearing completion
- Implementation follows close behind
    - notes on implementation based on experience and experimentation.

Implementations:

- Apple's Network.framework
- https://github.com/fg-inet/python-asyncio-taps