University of Glasgow
School of Computing Science

IETF®

An Architecture for Transport Services

Colin Perkins

Reporting on joint work with: Anna Brunstrom, Theresa Enghardt, Gorry Fairhurst, Karl-Johan Grinnemo, Tom Jones, Mirja Kühlewind, Tommy Pauly, Philipp Tiesel, Brian Trammell, Michael Welzl, & Chris Wood

https://tools.ietf.org/html/draft-ietf-taps-arch-00
https://tools.ietf.org/html/draft-ietf-taps-interface-00
https://tools.ietf.org/html/draft-brunstrom-taps-impl-00

Goals of the Transport Services Framework

- Ongoing transport innovation and network development – QUIC – not easily realised in applications

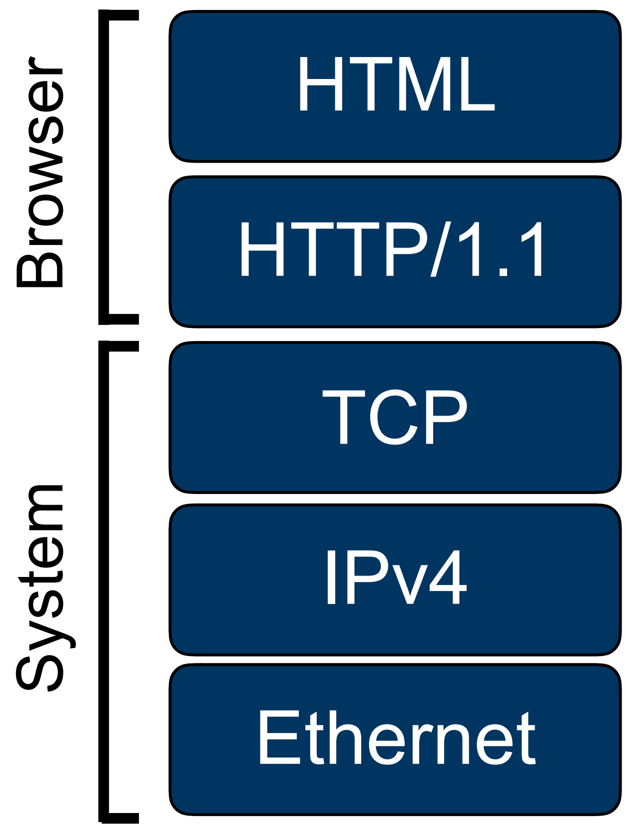
- Raise semantic level of network transport API to ease future evolution
  - Allow transport evolution independent of the application
  - Provide richer services to support application needs
  - Replace the Berkeley Sockets API as the basis for implementing networked systems

```
SOCKET(2)                    BSD System Calls Manual

NAME
     socket -- create an endpoint for communication

SYNOPSIS
     #include <sys/socket.h>

     int
     socket(int domain, int type, int protocol);

DESCRIPTION
```

What’s wrong with Sockets?

What’s Wrong With Sockets?

BSD Sockets ubiquitous since the 1980s – now showing their age



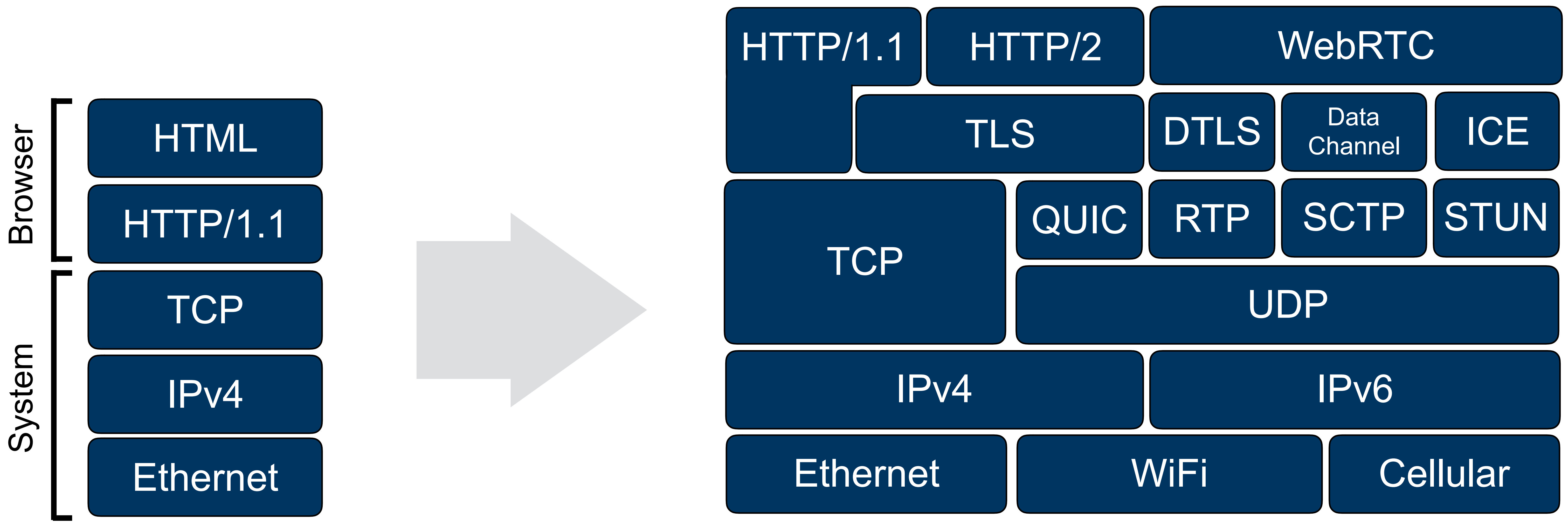
The protocol stack used to be straight-forward

Clear division between browser and system
Limited choice at the link layer

Sockets well suited to static system with limited options

What's Wrong With Sockets?

BSD Sockets ubiquitous since the 1980s – now showing their age



The stack is no longer simple – API must evolve to match

- Asynchronous and message oriented
- Rich notion of streams, objects, timing, and reliability
- Modern security and 0-RTT connection resumption
- Path discovery, connection racing, and NAT traversal
- Application, network, operator policies
- Flexibility in choice of transport

Implications for the Network API

- Asynchronous and message oriented
- Rich notion of streams, objects, timing, and reliability
- Modern security and 0-RTT connection resumption
- Path discovery, connection racing, and NAT traversal
- Application, network, operator policies
- Flexibility in choice of transport

Protocols deliver structured, typed, messages, not byte streams – e.g., HTTP objects, not byte arrays

Message arrivals and connectivity changes are fundamentally asynchronous

API should reflect this reality

- Asynchronous and message oriented
- Rich notion of streams, objects, timing, and reliability
- Modern security and 0-RTT connection resumption
- Path discovery, connection racing, and NAT traversal
- Application, network, operator policies
- Flexibility in choice of transport

HTTP/2, QUIC, SCTP support native multi-streaming; essential for low latency by avoiding HoL blocking

Real-time applications increasingly care about timing, managing partial reliability to control latency – message metadata

Transport cannot optimise delivery unless given application requirements, semantically meaningful messages

- Asynchronous and message oriented
- Rich notion of streams, objects, timing, and reliability
- Modern security and 0-RTT connection resumption
- Path discovery, connection racing, and NAT traversal
- Application, network, operator policies
- Flexibility in choice of transport

Security is essential, but difficult to implement

0-RTT connection resumption requires transport support and ability to signal idempotent data

Some features must be delegated to the application, but much can – and should – be generalised

- Asynchronous and message oriented
- Rich notion of streams, objects, timing, and reliability
- Modern security and 0-RTT connection resumption
- Path discovery, connection racing, and NAT traversal
- Application, network, operator policies
- Flexibility in choice of transport

Connectivity cannot be assumed – must probe protocols and paths to understand what works

IPv4/IPv6, TCP/QUIC but application just wants HTTP – let stack find the optimal transport for destination

NAT traversal and path discovery – some aspects can be generalised

- Asynchronous and message oriented
- Rich notion of streams, objects, timing, and reliability
- Modern security and 0-RTT connection resumption
- Path discovery, connection racing, and NAT traversal
- Application, network, operator policies
- Flexibility in choice of transport

Increasingly important to respect constraints on path usage – cost, latency, bandwidth, privacy, etc.

Complex issue, but clear policies must be expressible to the protocol stack

- Asynchronous and message oriented
- Rich notion of streams, objects, timing, and reliability
- Modern security and 0-RTT connection resumption
- Path discovery, connection racing, and NAT traversal
- Application, network, operator policies
- Flexibility in choice of transport

Applications shouldn’t over-specify transport

Indicate *transport services* they need, let the system race alternatives and find most suitable transport

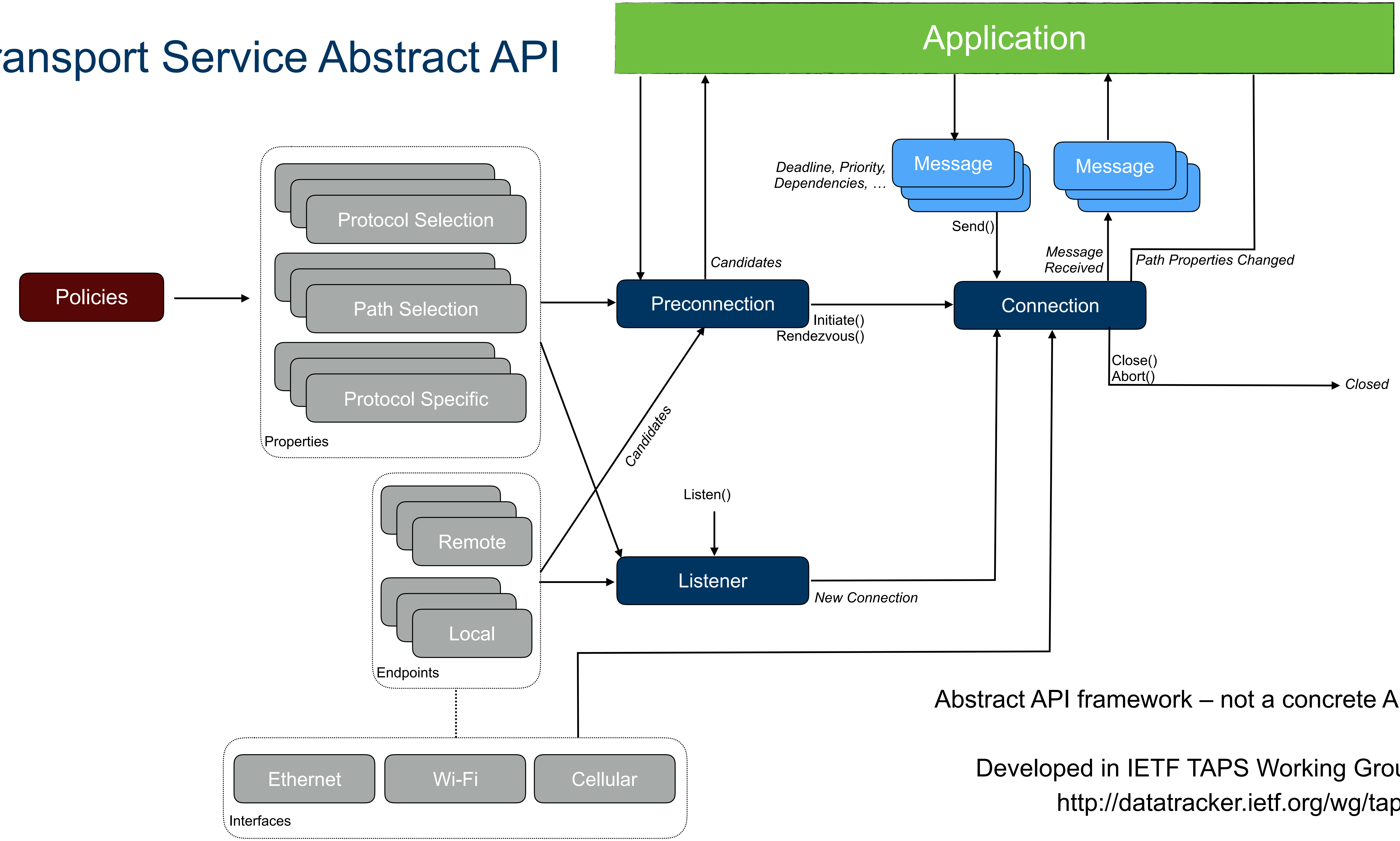
Flexibility to change the transport limits ossification

Proposed IETF Transport Services Framework

Defining an abstract API for transport services, to support new applications, enable innovation in transport protocols, and avoid ossification

- Asynchronous and message oriented
- Rich notion of streams, objects, timing, and reliability
- Modern security and 0-RTT connection resumption
- Path discovery, connection racing, and NAT traversal
- Application, network, operator policies
- Flexibility in choice of transport

Transport Service Abstract API



Abstract API framework – not a concrete API

Developed in IETF TAPS Working Group
http://datatracker.ietf.org/wg/taps/

Transport Services, the Web, and 5G

- Ongoing changes to applications, transport protocols, and networks

- Browser internal APIs must change to support QUIC, WebRTC, HTTP/2 – generalise rather than change to new protocol-specific APIs
  - Raise abstraction level – specify what, not how
  - A conceptual model for future transport APIs
- Let the web benefit from transport and path layer evolution – optimise for the changing network environment, independent of application code