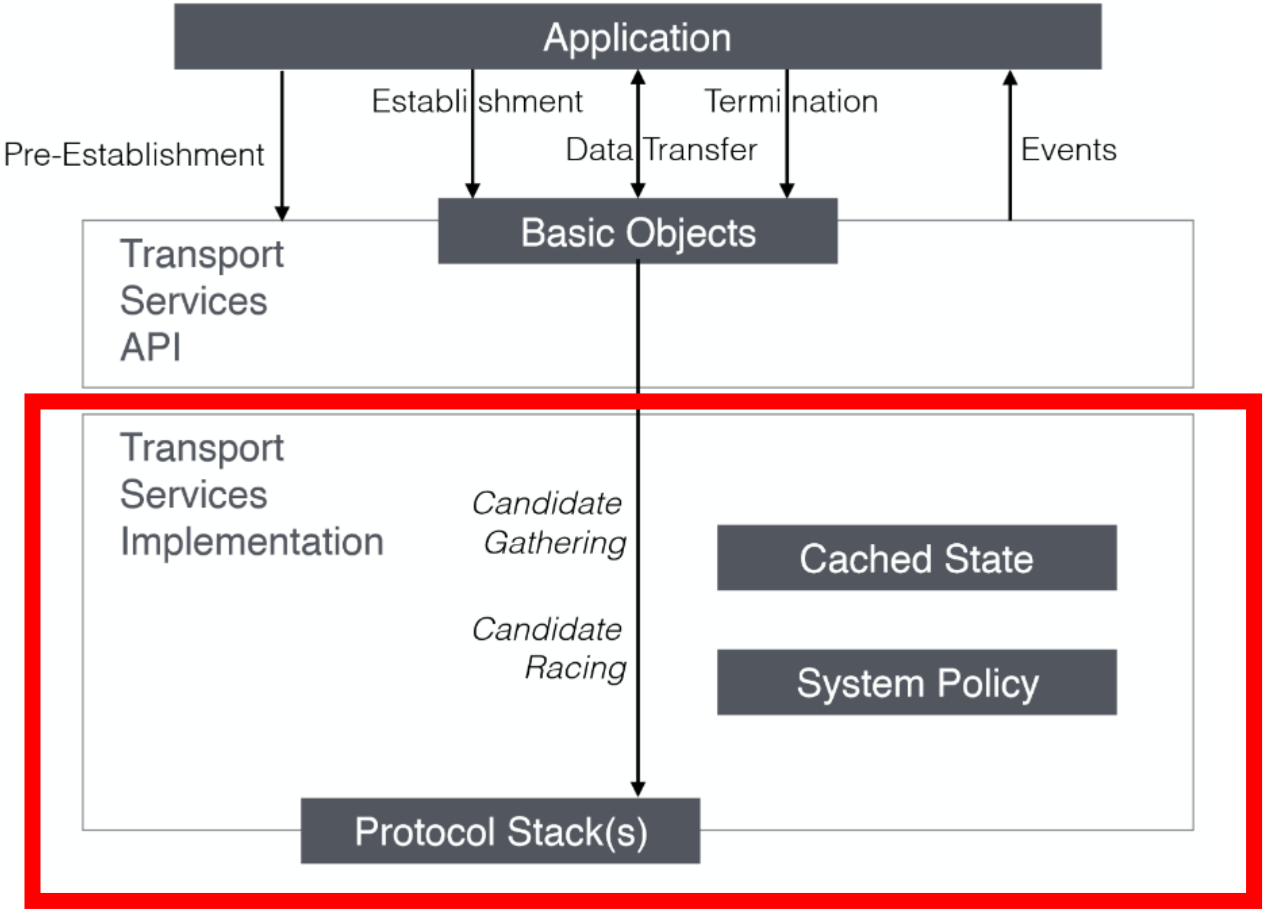
Implementing Interfaces to Transport Services

draft-brunstrom-taps-impl-00

Anna Brunstrom
TAPS
IETF 101, 21 March 2018, London

Scope

- Serve as a guide to implementation on how to build a system that provides a Transport Services API
- Complements architecture and API drafts



Implementing Basic Objects

- Preconnection: bundle of properties that describes the application constraints on the transport
- Connection: represents a flow of data in either direction between the Local and Remote Endpoints
- Listener: a passive waiting object that delivers new Connections
- The implementation should ensure that the copy of the properties held by the Connection or Listener is immutable

Implementing Pre-Establishment

- Application specifies Endpoints and its preferences regarding Protocol and Path Selection
  - Implementation stores these objects and properties as part of the Preconnection object
- Default values specified in the Transport Services API must be used for Properties not provided by the application
- Early failure detection should be done during pre-establishment
  - Protocol Properties include requirements or prohibitions that cannot be satisfied
  - Requested Protocol Properties are in conflict with each other

Role of system policy

- Implementation combines and reconciles several different sources of preferences when establishing Connections
  1. Application preferences specified during the pre-establishment
  2. Dynamic system policy compiled from internally and externally acquired information
  3. Default implementation policy, predefined policy by OS or application
- Any protocol or path used for a connection must conform to all three sources of constraints

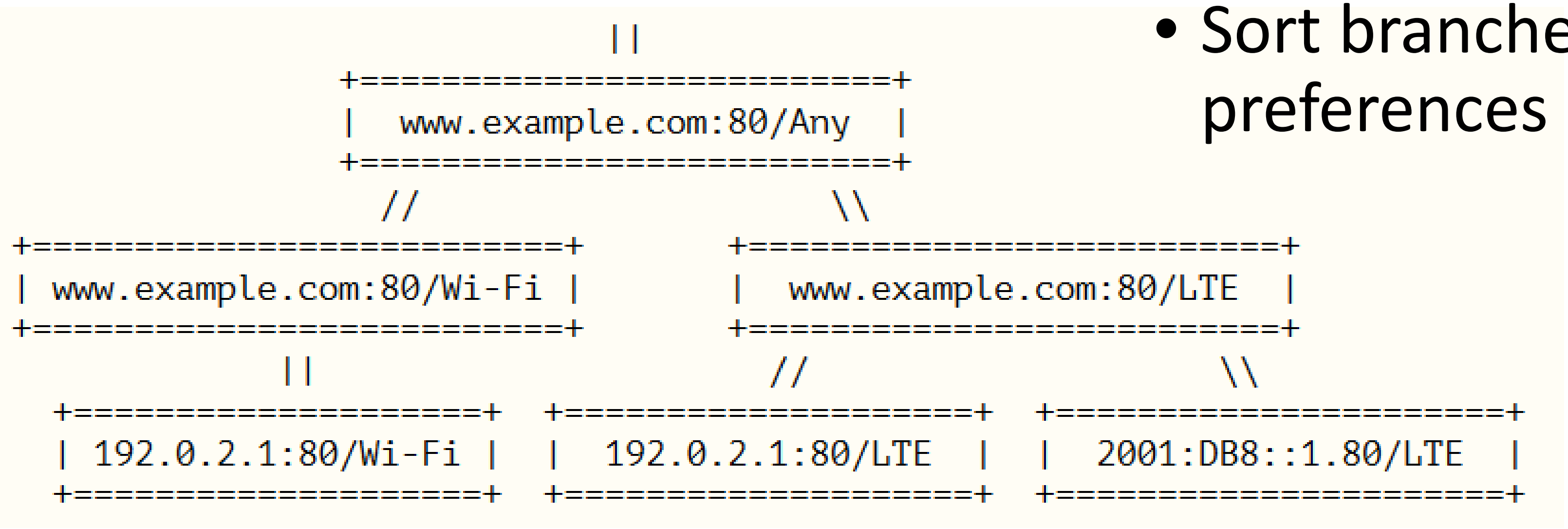
Implementing Connection Establishment

Two main steps:

- Candidate Gathering, identifying the paths, protocols, and endpoints that can be used
- Candidate Racing, in which the necessary protocol handshakes are conducted in order to select which set to use

Candidate Gathering

- Candidates can be described by [Endpoint, Path, Protocol]
- Available candidates can be structured as a tree



```
                              ||
                 +==========================+
                 |  www.example.com:80/Any  |
                 +==========================+
                   //                    \\
+==========================+       +==========================+
| www.example.com:80/Wi-Fi |       |  www.example.com:80/LTE  |
+==========================+       +==========================+
               ||                     //                    \\
 +====================+  +====================+  +======================+
 | 192.0.2.1:80/Wi-Fi |  |  192.0.2.1:80/LTE  |  |  2001:DB8::1.80/LTE  |
 +====================+  +====================+  +======================+
```

- Branching Order-of-Operations
  1. Alternate Paths (e.g. Wi-Fi then LTE)
  2. Protocol Options (e.g. QUIC then HTTP/2)
  3. Derived Endpoints (e.g. IPv6 then IPv4)
- Sort branches based on application preferences and policy

Candidate Racing

- Racing approaches: Immediate (avoid as default), Delayed, Failover
- Completes when one candidate has successfully established a connection, or all candidates have failed to connect
- Determining Successful Establishment
  - TCP – established when TCP handshake completes
  - Multiplexed connection – immediately established, no handshake needed
    - Initiate may not result in a ConnectionReceived event at the peer
  - UDP - established as soon as a local route to the peer endpoint is confirmed

Implementing listeners

- Listener object should register for incoming traffic on all eligible network interfaces or paths
    - Implementation should monitor network path changes and register and de-register the Listener across all usable paths
- Listener object should register across all eligible protocols for each path
    - Inbound Connections delivered by the implementation may have heterogeneous protocol stacks

Data Transfer - Sending message

- Depends on the top-level protocol in the established Protocol Stack
- Support for the different send parameters (Lifetime, Niceness, Ordered, Idempotent, Corruption Protection Length, Immediate Acknowledgement, Instantaneous Capacity Profile)
- 0-RTT data needs to be provided before the process of connection establishment has begun
- Implementation should keep a copy of this data and provide it to each 0-RTT protocol started during racing

Data Transfer - Receiving message

- Depends on the top-level protocol in the established Protocol Stack
- Size and boundaries of the Message are not known beforehand
    - Application can communicate the parameters for the Message

Implementing Termination

- Application not able to read any more data after calling Close
  - No half-closed connections
- A Close may not always provoke a Finished event at peer
  - Connection may be mapped to a stream of an underlying multi-streaming protocol
- Similarly an Abort may not always provoke a ConnectionError event at peer

Other parts covered in draft

- Implementing Maintenance
  - Changing Protocol Properties and Handling Path Changes
- Cached State
  - Protocol state caches and performance caches
- Specific Transport Protocol Considerations
  - TCP, UDP, SCTP, TLS, HTTP, QUIC, HTTP/2
- Rendezvous and Environment Discovery
  - Connection establishment process in peer-to-peer Rendezvous scenarios

?