An Abstract Application Layer Interface to Transport Services

draft-trammell-taps-interface-01

Brian Trammell

TAPS — IETF 101 — Montreal — Tue 17 July 2018

Interface Design Principles (§3) (a review)

We set out to define a ***single interface*** to a variety of transport protocols to be used in a variety of application design patterns, to enable applications written to a single API to make use of multiple transport protocols in terms of the features they provide, providing:

- explicit support for ***security properties*** as first-order transport features;
- ***asynchronous*** connection, transmission, and reception;
- support for ***multistreaming and multipath*** transport protocols; and
- ***atomic transmission of data***, using application-assisted framing and deframing where necessary.

Interface Diagram (as of -01)

Properties (related to Send() properties)

`Require()` `Prefer()` `Ignore()` `Avoid()` `Prohibit()`

Security parameters

Preconnection

`Clone()`

`Initiate() → Ready<>`
`Listen() → CReceived<>`
`Rendezvous() → RDone<>`
`Stop() → Stopped<>`

Endpoints

Local

Remote

Connection

`Clone()` → Connection Group

`Send(MCtx,EOM) →`
`Sent<>, Expired<>`
`Receive() →`
`Received<Data/Metadata>`
`ReceivedPartial<>`

`Close() → Closed<>`
`Abort() → Aborted<>`

(non-editorial) changes since -00

- #201 Restructure Transport Properties
- #200 Rework Partial Sends and Receives
- #198 Message Receive Metadata
- #195 Ordering of API Events
- #181 Rework Interface Types
- #171 Batching Sends

(non-editorial) changes since -00

- #201 Restructure Transport Properties
- #200 Rework Partial Sends and Receives
- #198 Message Receive Metadata
- #195 Ordering of API Events
- #181 Rework Interface Types
- #171 Batching Sends

#201 Transport Parameters Rework

- All of the various ways to configure stacks (pre-selection, post-selection, and per-send) are related, but were spread throughout the document
- New approach: group all (non-security) parameters into into Properties (new §12), attempt to reclassify them.
    - Note: the authors ***do not think we have this right yet***, but we do think it's less intentionally confusing than it was.
    - Definitely needs reordering (order is kind of random)
    - May need new / renamed axes / classifications.
- Preferences still expressed using `Require()`, `Prefer()`, `Avoid()`, `Prohibit()`; send properties are bound to `MessageContext` passed on `Send()`.

#201: current property "axes"

- Data type

  Boolean / Enumeration / Integer / Preference

- Scope

  Preconnection / Connection / Message

- Classification

Affected Aspects	Path & Protocol Selection	Protocol Operation	Control Flow
Level of Abstraction Immediate	Selection Property	Protocol Property	Control Property
Interpreted	Intent		

Properties (1/3)

	Type	Dep.	Select	protocol/ control prop. post-Select	protocol/ control prop. per-Send
12.3.1. Final	bool				✓/?
12.3.2. Reliable Message Transfer	pref		✓		
12.3.3. Configure Reliability Per Message	pref		✓		
12.3.4. Reliable Transfer (Message)	bool	↑			✓
12.3.5. Preservation of Data Ordering	pref		✓		
12.3.6. Ordered	bool	↑			✓
12.3.7. Direction of communication	enum		?	?	
12.3.8. 0-RTT Establishment w/Idem.	pref		✓		
12.3.9. Idempotent	bool	↑			✓
12.3.10. Multistream in Group	pref		✓		
12.3.11. Excessive RTX Notification	pref		✓		
12.3.12. Exc. RTX Notification Threshold	int	↑	✓	✓	

Properties (2/3)

			protocol/ control prop.		
	Type	Dep.	Select	post-Select	per-Send
12.3.13. Soft Error Notification	pref		✓		
12.3.14. Checksum Coverage Control	pref		✓		
12.3.15. Checksum Coverage Length	int	↑			✓
12.3.16. Recv Checksum Requirement	int		✓	✓	
12.3.17. Interface Instance / Type	(enum,pref)		✓		
12.3.18. PvD Instance / Type	(enum,pref)		✓		
12.3.19. Capacity Profile (intent)	enum		✓	✓	✓
12.3.20. Congestion Control	pref		✓		
12.3.21. Niceness	int			✓	✓
12.3.22. Abort Timeout	int		✓		
12.3.23. Connection Group TX Scheduler	enum		✓	✓	

Properties (3/3)

				protocol/ control prop.	
	Type	**Dep.**	**Select**	**post-Select**	**per-Send**
12.3.24. Max Idempotent Send Size	int			r/o	
12.3.25. Max No-Frag Send Size	int			r/o	
12.3.26. Max (non-partial?) Send Size	int			r/o	
12.3.27. Max (non-partial?) Recv Size	int			r/o	
12.3.28. PR Send Lifetime	int	12.3.3.			✓

Some Observations from the Editor (+discussion)

- Calling these axes is a little misleading: they're not orthogonal
- We have only six distinct kinds of thing:
  - Preference used for selection, scoped to preconnection, read-only after connection.
  - Property used to control how messages are sent, scoped to message (boolean or integer, usually linked to selection preference).
  - Property used to control protocol operation, scoped to preconnection + connection (usually integer, e.g. sizes/timeouts), possibly also usable for selection.
  - Property used to inspect protocol operation, scoped to connection, read-only (usually integer, e.g. buffer size).
  - Enumeration/preference tuples for selecting interface/PvD.
  - Intents, which can influence selection, configuration, scheduling, etc. at a higher level.

#200 Partial Send and Receive

- API is organized around atomic write/read of messages
  - (using application-supplied deframing when the underlying transport doesn't do framing, see §8.4)
- But sometimes you have a message (or a real stream) that won't fit into a buffer.
  - Solution: partial read/write
    - Introduce optional `EOM` parameter to `Send();` calls with `EOM = false` → still writing to a partial message identified by a given `MessageContext`.
    - `ReceivedPartial<>` event fires when a partial message is received.
  - Partial read/write boundaries are not preserved.

Open issue: API for idempotent Send on establishment (#112 / #124)

- How does the application tell the stack that it wants to send some 0RTT data?
  - Some tradeoffs here, but mainly a bikeshed.

- Option 1: as in #124, hold any data sent until an explicit `Connection.Start()` call.
  - `Send()` before `Start()` is 0RTT if idempotent.
  - `Start()` is always required, even if you don't know what 0RTT is.
- Option 3: 0RTT behavior is implied by 0RTT selection properties.
  - When `Initiate()` is called and selects a 0RTT-capable stack, the actual initiation is delayed slightly to wait for the first `Send()`, which is 0RTT if idempotent.
  - Note this makes racing 0RTT-capable and 0RTT-incapable stacks impossible.
- Option 3.5: as 3, but with a `Preconnection.InitiateNow()` to override the wait-for-`Send()` behavior (e.g. for application protocols where the server sends first)
- Option 5: Add `Preconnection.Send()`, which initiates with 0RTT data.

Next steps

There are still some open issues:
github.com/taps-api/drafts/issues