Transport-Independent Path Layer State Management

Brian Trammell
TSV Area Meeting - IETF 98 Chicago - 27 March 2017
The Problem

- Lots and lots of state-keeping devices on path…
  - … that assume TCP semantics
  - … won’t work with non-TCP transports
- UDP-based transports need:
  - frequent keepalives
  - explicit directional rules, port mapping
  - other nasty hacks
  - or fall back to TCP
- These devices will do *something* with UDP transports anyway
  - Let’s define something sane for them to do.
TCP state modeling at middleboxes

- closed
  - SYN → SYN/ACK
  - RST
  - ACK → FIN
- half-closed
  - FIN → ACK
- SYN
  - → SYN/ACK
- SYN/ACK
  - ACK → established
- established
A generic state machine

- **zero**
  - to idle
  - to stop
  - stop-confirm
  - y → x

- **stopping**
  - to stop
  - stop-confirm
  - y → x

- **stop-wait**
  - stop
  - x → y

- **uniflow**
  - associate
  - b → a

- **associating**
  - confirm
  - a → b

- **associated**

- **to idle**

- **a → b**
One-way flows & one-sided devices

Zero

Stop

Stop-confirms

Stop waiting

Stop associated

Stop idle

Stop uniflow

Stop associate

Stop confirm

Stop x → y

Stop b → a

Stop a → b
Ensure intent & return-routeability

Diagram:

- **zero**
  - to idle
  - to stop
  - stop-confirm
- **stopping**
  - to stop
  - stop-confirm
- **stop-wait**
  - stop
- **uniflow**
  - associate
  - confirm
- **associated**
  - associate
  - confirm

Transitions:

- a → b
- b → a
- x → y
- y → x
Two-way stop
The draft

• As input to protocol design: consider which signals are made publicly available by your protocol, and how these will be used to maintain transport state on-path.

• As guidance for middlebox design: separate the extraction of signals from headers from the semantic treatment of those signals for state maintenance.

• Next steps?