Post Sockets

the story so far

Post Sockets Secret Cabal Meeting, Zürich
13 February 2017

Brian Trammell, ETH Zürich

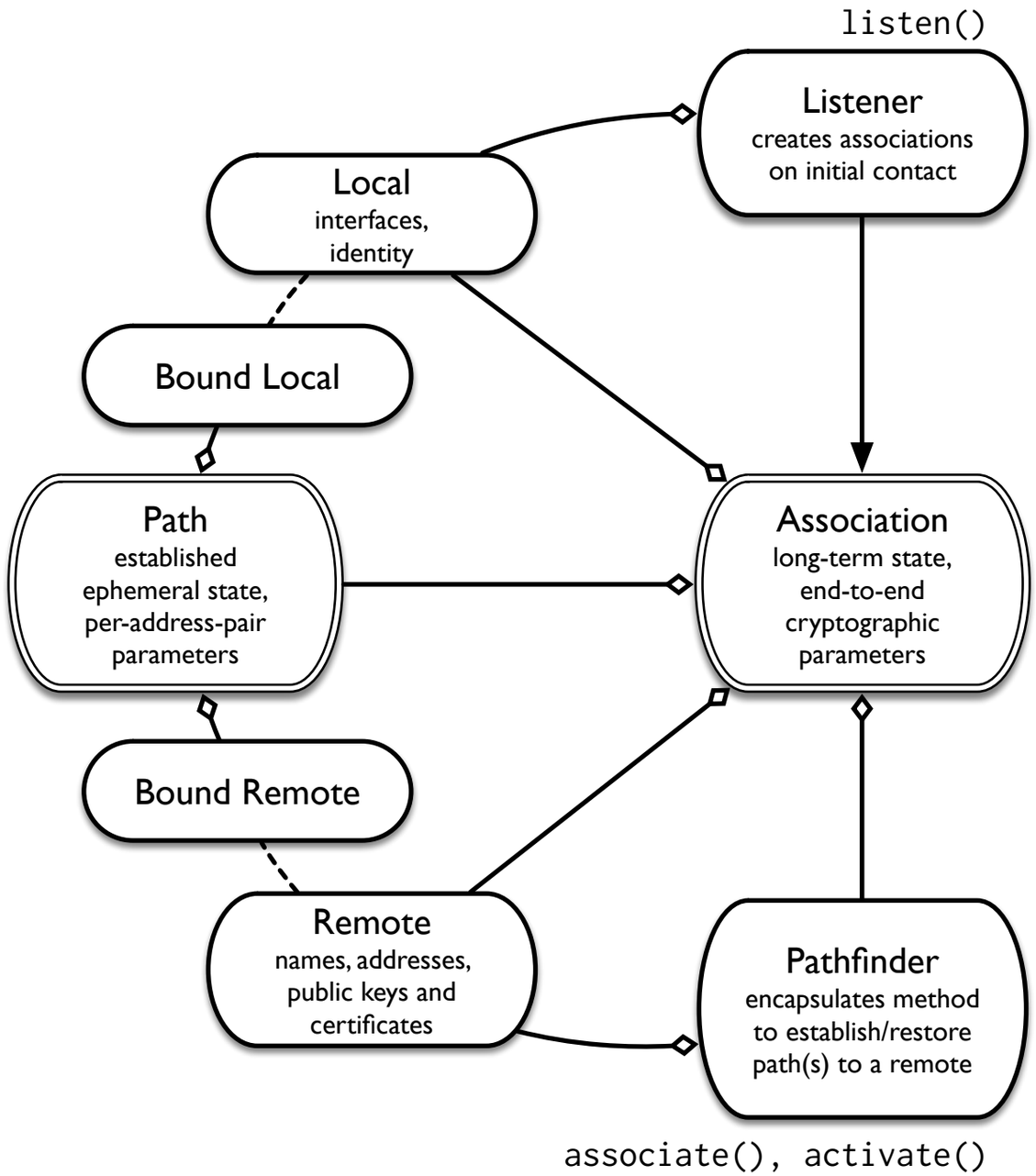
mami
measurement and architecture for a middleboxed internet

ETH zürich

a few insights
about network applications

- ***Applications deal in objects*** (messages) of arbitrary size
  - Files, assets, media frames, etc. *depend* on each other, have no strict ordering.
  - “Real” streams exist too: unknown data source length, not easily divisible.
    - Most “streams” are the result of applying sequential-file logic to networking
- The network of the future is ***explicitly multipath***.
  - Started with dual-stack, continues with mobile+802.11 at terminal+CPE
  - Applications (as well as transports) must have access to path properties.
- Future transports must ***guarantee security properties***.
- Message reception is ***inherently asynchronous***.
  - Present scalable programming models enable async IO.
  - (Core kernel/user interface may remain `select()`-based for performance.)

Abstract Programming Interface
Classes



- **Local** specifies information about a local endpoint.
- **Remote** encapsulates information about a remote endpoint.
- **Path** encapsulates ephemeral transport state and properties about a single (addressable) endpoint pair,
  - Roughly analogous to an open socket. May need renaming.
- **Association** encapsulates long-term state about an (identified) endpoint pair
  - Associations are persistent, cacheable, and can be serialized and migrated.
  - Encapsulates all cached parameters for transport and security protocols
  - Associations have zero (‘inactive’) or more (‘active’) current paths.
- **Listener** provides functions for passive establishment and activation of associations
- **Pathfinder** provides functions for active establishment and activation of associations

Abstract Programming Interface
Establishing and Activating Associations

- `associate(local, remote, receive_handler) → Association`
  creates an Association between two endpoints to encapsulate long-term state.
  - Requires an active Listener or simultaneous `associate()` call at remote.
  - Associates a `receive_handler` at association establishment.
- `listen(local, accept_handler) → Listener`
  creates a Listener to handle association and activation.
  - `accept_handler(listener, local, remote) -> boolean`
    is called for new association requests, returns True if accepted; calls
  - `Listener.accept(local, remote, receive_handler) -> Association`
    to finalize association setup.

Abstract Programming Interface
Object and Stream properties

- Objects and streams have a **niceness**
  - Nicer `send()`s/`write()`s yield to less nice
- Objects have a **deadline**
  - An object will be cancelled if it cannot be realistically received before this deadline
  - Infinite-deadline objects are fully reliable
- Objects may have **antecedents…**
  - other objects which should be sent before
- …or be associated with a **sequence**
  - objects within which depend on each other sequentially
  - better mapping to multistreaming transport protocols

Abstract Programming Interface
Sending and Receiving Objects

- `Association.activate()`
  ensures that an association has at least one active path.
- `Association.send(msg)`
  sends a message to the remote via the active path(s) on the association.
- `Association.send(msg, lifetime, niceness, oid, antecedent_oids, ack_handler, expiry_handler)`
  `Association.send(msg, lifetime, niceness, group, ack_handler, expiry_handler)`
  wider API for partially-reliable and prioritized messages.
  - `ack_handler` called after the message is fully ACKed by the remote
  - `expiry_handler` called if the message expires before it is sent
  - missing: method to constrain messages to certain paths
- `receive_handler(association, msg)`
  called for each message received

Abstract Programming Interface
Handling Association Events

- `Association.receive_handler = receive_handler(association, msg)`
  set the handler to call for incoming messages
- `Association.path_up_handler = path_up_handler(association, path)`
  set the handler to call on new path available
- `Association.path_down_handler = path_down_handler(association, path)`
  set the handler to call on path no longer available
- `Association.dormant_handler = dormant_handler(association)`
  set the handler to call on no paths available (i.e., `Association.activate()` must be called to send)

Abstract Programming Interface

…sometimes, you really need a stream…

- `Association.open_stream(stream_id)` → `stream_like_thing`

  Create a stream over an active association, with a given stream ID for multistreaming protocols.

  - Returns a stream as idiomatic for the implementing platform.
  - Stream shutdown is platform-specific.

Transport Independence

- Only one requirement on the wire: framing for objects
    - …lack thereof is a design flaw in `SOCK_STREAM`…
    - Implies lack of interop between Post and non-Post nodes over TCP fallback.
- Assumption that the transport protocol provides encryption for payload confidentiality and public header integrity protection.
- Can make use of other transport features on demand:
    - Multipath load balancing and migration
    - Multistreaming for objects and streams
- Object properties (niceness, deadline, dependencies) are sender-side only; path properties can be derived locally too.

Why now?

- We've known most of this for years.
  - API ~~stolen from~~ heavily inspired by SCTP, Minion.
  - Dual-stack networks are multipath too, and happy-eyeballs is a path probing technique.
- But the world is a little different now:
  - New transports over UDP (e.g. QUIC) make it easier to deploy new userspace libraries than new kernel interfaces.
  - Ubiquitous multihoming of user terminals over heterogeneous link layers makes application/transport involvement in path selection more important.
  - Ubiquitous multiprocessing means asynchronous I/O is more than a mere syntactic win.