Post Sockets

A generic API for multipath-cooperative communication

Brian Trammell, Networked Systems and Network Security Groups
with Laurent Chuat and Jason Lee, Network Security Group
and Mirja Kühlewind, Networked Systems Group

mami

measurement and architecture for a middleboxed internet

ETH zürich

measurement architecture experimentation

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688421.The opinions expressed and arguments employed reflect only the authors' view. The European Commission is not responsible for any use that may be made of that information.

Supported by the Swiss State Secretariat for Education, Research and Innovation under contract number 15.0268. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the Swiss Government.

SOCK_STREAM: yesterday's interface

- Synchronous
- Unicast
- No framing support
- Single-stream
- Single-path
- No path abstraction
- No security
- Implicit measurability

- But it makes the network look like a file. Simplicity wins!

SOCK_STREAM: **yesterday’s interface, today**

- Synchronous
- ~~Unicast~~ (nobody cares, multicast routing, security too hard)
- ~~No framing support~~ (nobody cares, apps do this anyway)
- ~~Single-stream~~ (just open multiple flows)
- ~~Single-path~~ (MPTCP for failover and balancing, might deploy…)
- ~~No security~~ (TLS/OpenSSL solves all our problems, right?)
- No path abstraction

- ***Can we do better than this?***

`SOCK_SEQPACKET`: tomorrow’s interface, yesterday

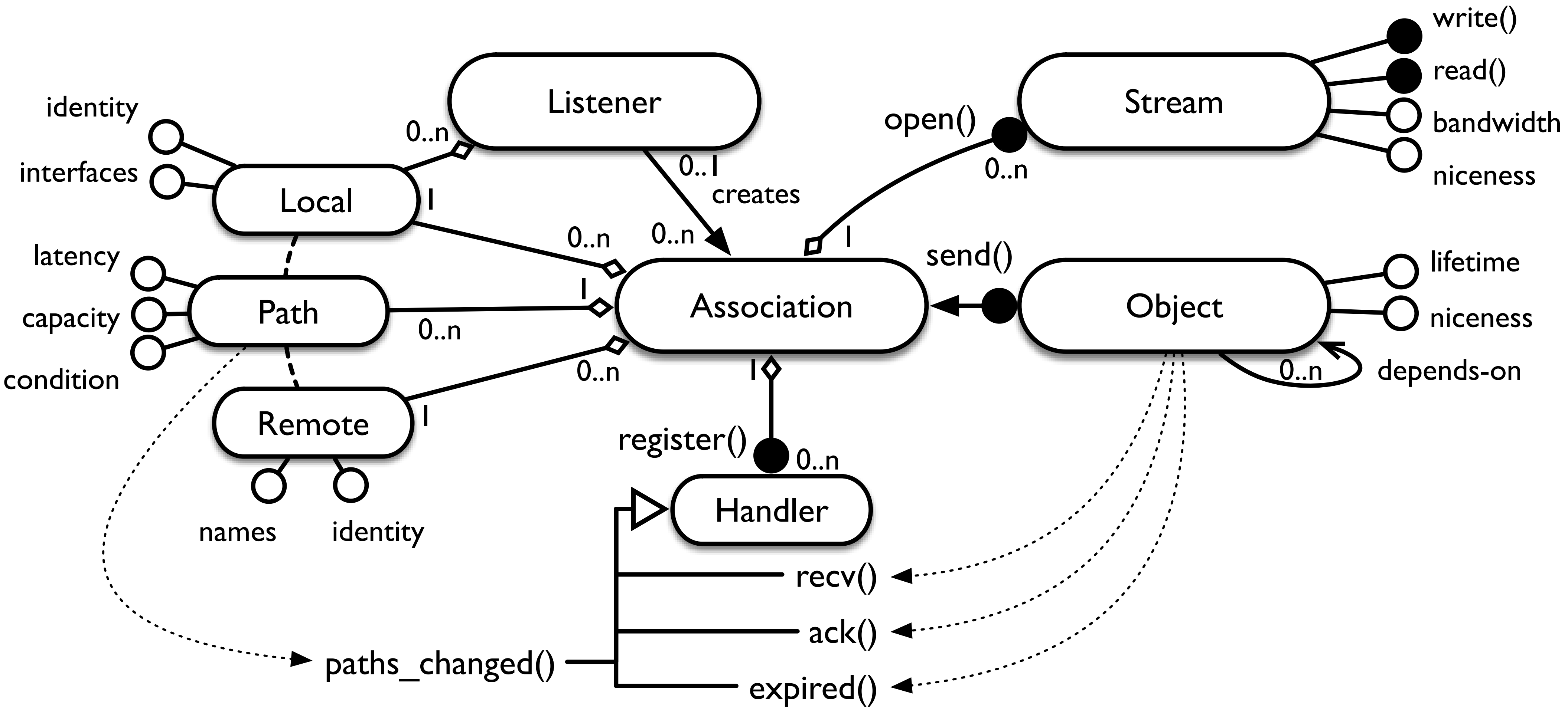
- Synchronous (with async event notification!)
- Unicast or multicast!
- Framing support!
- Single- or multiple-stream!
- Multipath! (for failover)
- No security
- No path abstraction
- Bound to Stream Control Transmission Protocol (SCTP), largely undeployable in the open Internet today.

- ***Let’s go back to the interface…***

Post Sockets: Insights and Principles

- ***Applications deal in objects*** (messages) of arbitrary size
  - May depend on each other, but don’t have a strict stream ordering
  - Let the transport layer solve the optimization problem!
- The network of the future is ***explicitly multipath***.
  - Applications must have access to path properties.
- Future transports must ***guarantee security properties.***
  - “Bolted-on” security (TLS) adds complexity, latency.
  - Path elements must not be able to see transport-layer metadata.
- Message reception is ***inherently asynchronous.***
  - Present scalable programming models enable (and require!) async IO.

Post Sockets: Abstractions



Abstractions

- ***Associations*** represent communication state among a group (pair) of network-connected processes:
    - ***Remote*** and ***Local*** Public key and certificate information
    - Session and cryptographic state for fast resume
    - Currently available ***Paths*** (or interface addresses)
    - Callbacks for association events (object receive, etc)
- ***Listeners*** allow for passive opening of Associations
- ***Objects*** given to one end of an association appear at the other, subject to priority, lifetime, and dependency constraints.
    - Objects may require multiple segments to transport.
    - Object boundaries guaranteed to be preserved.
- ***Streams*** over Associations allow bandwidth reservations for nonmaterialized, streaming data to coexist with Objects.

Entry Points and Events

- Associations created with `associate()`, given Local, Remote.
- Most calls are conceptual methods on Association:
  - `.send()`: send an object
    - object properties include `lifetime, niceness, antecedents`
  - `.open()`: get a new stream compatible with platform's stream IO API
    - stream properties include `bandwidth, niceness`
  - `.register()`: register a handler for a given event
    - event types include `recv, ack, expired, paths-changed`
- Listener (created with `listen()`): rump Association with a single event, `accept.`
- Local and Remote API are architecture-dependent.

Implementation

- API is designed to be transport-, architeture- and platform-neutral
- Different implementations will have different feature tradeoffs:

Implementation/ Features	over TCP	over SCTP (or SCTP over UDP over DTLS)	native transport over UDP userland/ MAMI MCP	native transport over UDP in-kernel/ MAMI MCP	SCION socket/ UDP userland
Async Receive	coroutines in userland	coroutines in userland	coroutines in userland	zero copy w/ coroutines	coroutines in userland
Object Framing and Interleaving	object header in TCP stream (can deadlock)	provided by SCTP	object header with native segmentation	object header with native segmentation	object header in SCION stream
Object Lifetime and Reliability	sender-side-only expiry	sender-side-only expiry, provided by SCTP-PR	expiry at sender, receiver, and on-path	expiry at sender, receiver, and on-path	expiry at sender, receiver, and on-path
Multistreaming	multiple TCP sockets	multiple SCTP streams, single association	via object interleaving	via object interleaving	via object interleaving
Path Primacy	interface only no path info MPTCP?	interface only no path info SCTP path failover	interface only path info via MCP	interface only path info via MCP	PCFS routing path info via PCB
Security	using TLS	using DTLS	using DTLS 1.3	using DTLS 1.3	integrated with SCION trust root

Post Sockets and MAMI MCP

- Object properties (lifetime and niceness) exposed to the path via the MCP; lifetime can be implemented by MCP-aware bottleneck devices.
- Path properties derived via MCP and measurement facilities.
- Post Sockets implemented as native transport atop the MCP, with some headers public and some private.

What’s next?

- Further refinement of the interface.
- Pilot implementation atop TCP, SCION.
- Is this something we want to pursue for the Flexible Transport Layer (FTL)?
- Is this something that would be useful for NEAT/TAPS?