Verifying Reliable Sessions Over an Unreliable Network in Distributed Separation Logic

Léon Gondelman, Jonas Kastberg Hinrichsen,
Mário Pereira, Amin Timany, Lars Birkedal

Cambium, Paris

January 16, 2023

Communicating processes

- Network communication & message-passing concurrency:

  —> coordination is done via exchanging messages (not via shared memory)

  —> communication protocols and ownership transfer play central role.

- One can expect that specification and reasoning about network and concurrency should exhibit common patterns and similar program logics.

Session types

high-level typing pattern to show safety for message-passing style concurrency

Example program:

$$\texttt{let}\,(c, c') := \texttt{new\_chan}\,()\ \texttt{in}$$
$$\texttt{fork}\ \{\texttt{let}\,x := \texttt{recv}\ c'\ \texttt{in}\,\texttt{send}\ c'\ (x+2)\}\,; \quad \text{// Service thread}$$
$$\texttt{send}\ c\ 40;\ \texttt{recv}\ c \quad \text{// Client thread}$$

Session types:

$$c \ :\ \texttt{chan}\ (\mathbf{!Z}.\,\mathbf{?Z}.\,\texttt{end}) \qquad \text{and}$$
$$c' \ :\ \texttt{chan}\ (\mathbf{?Z}.\,\mathbf{!Z}.\,\texttt{end})$$

Properties obtained:

- ☑ Program does not crash
- ☒ Program is correct (returns 42)

$$\Gamma \vdash \lambda c.\,\texttt{let}\,x := \texttt{recv}\ c\ \texttt{in}\ \texttt{send}\ c\ (x+2) : \texttt{chan}\ (\mathbf{?Z}.\,\mathbf{!Z}.\,\texttt{end}) \multimap \mathbf{1} \dashv \Gamma$$

Actris Framework

high-level specification pattern to reason about for reliable message-passing communication [Hinrichsen et al. 2020]

- **Dependent Separation Protocols:**

$$prot \in \text{iProto} ::= \,!\,\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,prot \mid ?\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,prot \mid \textbf{end}$$

$$\texttt{echo\_prot} \triangleq \mu rec.\ ?(s : \text{String})\,\langle s\rangle.\,!\,(n : \mathbb{N})\,\langle n\rangle\{n = |s|\}.\,rec$$

- **Specifications for message-passing concurrency:**

$$\{c \rightarrowtail \,!\,\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,prot * P[\vec{t}/\vec{x}]\}$$
$$\texttt{send}\ c\ (v[\vec{t}/\vec{x}])$$
$$\{c \rightarrowtail prot[\vec{t}/\vec{x}]\}$$

$$\{c \rightarrowtail ?\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,prot\}$$
$$\texttt{recv}\ c$$
$$\{w.\,\exists(\vec{y} : \vec{\tau}).\,(w = v[\vec{y}/\vec{x}]) * P[\vec{y}/\vec{x}] * c \rightarrowtail prot[\vec{y}/\vec{x}]\}$$

Network Communication

Actris Session Type-based Reasoning

- provides a high-level model of reliable communication (Actris Ghost Theory)
- has been applied so far only to reason about message-passing concurrency, where the communication layer itself is reliable.

Network communication is fundamentally unreliable and asynchronous

- messages are lost, arrive out of order, got duplicated, or forged by adversary
- network partitions make it impossible to distinguish, in a finite amount of time, between delayed messages and lost messages (e.g. due to remote's crash)

Research Question

How can we design a program logic
for reliable network communication
using session-typed based reasoning
as high-level specification pattern?

Aneris Project

Aneris: A Mechanised Logic for Modular Reasoning about Distributed Systems

Morten Krogh-Jespersen, Amin Timany*, Marit Edna Ohlenbusch, Simon Oddershede Gregersen, and Lars Birkedal

Aarhus University, Aarhus, Denmark

Abstract. Building network-connected programs and distributed systems is a powerful way to provide scalability and availability in a digital, always-connected era. However, with great power comes great complexity. Reasoning about distributed systems is well-known to be difficult.
In this paper we present Aneris, a novel framework based on separation logic supporting modular, node-local reasoning about concurrent and distributed systems. The logic is higher-order, concurrent, with higher-order store and network sockets, and is fully mechanized in the Coq proof assistant. We use our framework to verify an implementation of a load balancer that uses multi-threading to distribute load amongst multiple servers and an implementation of the *two-phase-commit* protocol with a replicated logging service as a client. The two examples certify that Aneris is well-suited for both horizontal and vertical modular reasoning.

Keywords: Distributed systems · Separation logic · Higher-order logic · Concurrency · Formal verification

1 Introduction

Reasoning about distributed systems is notoriously difficult due to their sheer complexity. This is largely the reason why previous work has traditionally focused on verification of protocols of core network components. In particular, in the context of model checking, where safety and liveness assertions [29] are considered, tools such as SPIN [9], TLA+ [23], and Mace [17] have been developed. More recently, significant contributions have been made in the field of formal proofs of *implementations* of challenging protocols, such as two-phase-commit, lease-based key-value stores, Paxos, and Raft [7, 25, 30, 35, 40]. All of these developments define domain specific languages (DSLs) specialized for distributed systems verification. Protocols and modules proven correct can be compiled to an executable, often relying on some trusted code-base.

Formal reasoning about distributed systems has often been carried out by giving an abstract model in the form of a *state transition system* or *flow-chart* in the tradition of Floyd [5], Lamport [21, 22]. A state is normally taken to be a

* This research was carried out while Amin Timany was at KU Leuven, working as a postdoctoral fellow of the Flemish research fund (FWO).

(ESOP 20)

Original Aneris Paper

AnerisLang, an OCaml-like language with

- UDP sockets primitives (msgs can be dropped, reordered or duplicated)
- Well-defined formal operational semantics
- Compiler from a subset of OCaml

Aneris Program Logic, a logic with

- All features from the Iris Framework (on top of which it is built in Coq)
- Proof rules to reason about node-local concurrency
- Proof rules to reason about UDP network communication

Hoare Logic → Higher-Order Concurrent Separation Logic → Distributed Separation Logic

Key contribution

We connect of the dependent session protocols of Actris to *distributed systems*, without extending the trusted code base of Aneris or Actris.

We achieve this

(1) by developing reliable communication library on top of Aneris' basic unreliable network primitives

(2) by proving the high-level Actris-like specifications of this library in Aneris, which involved coming up with a *session escrow pattern*

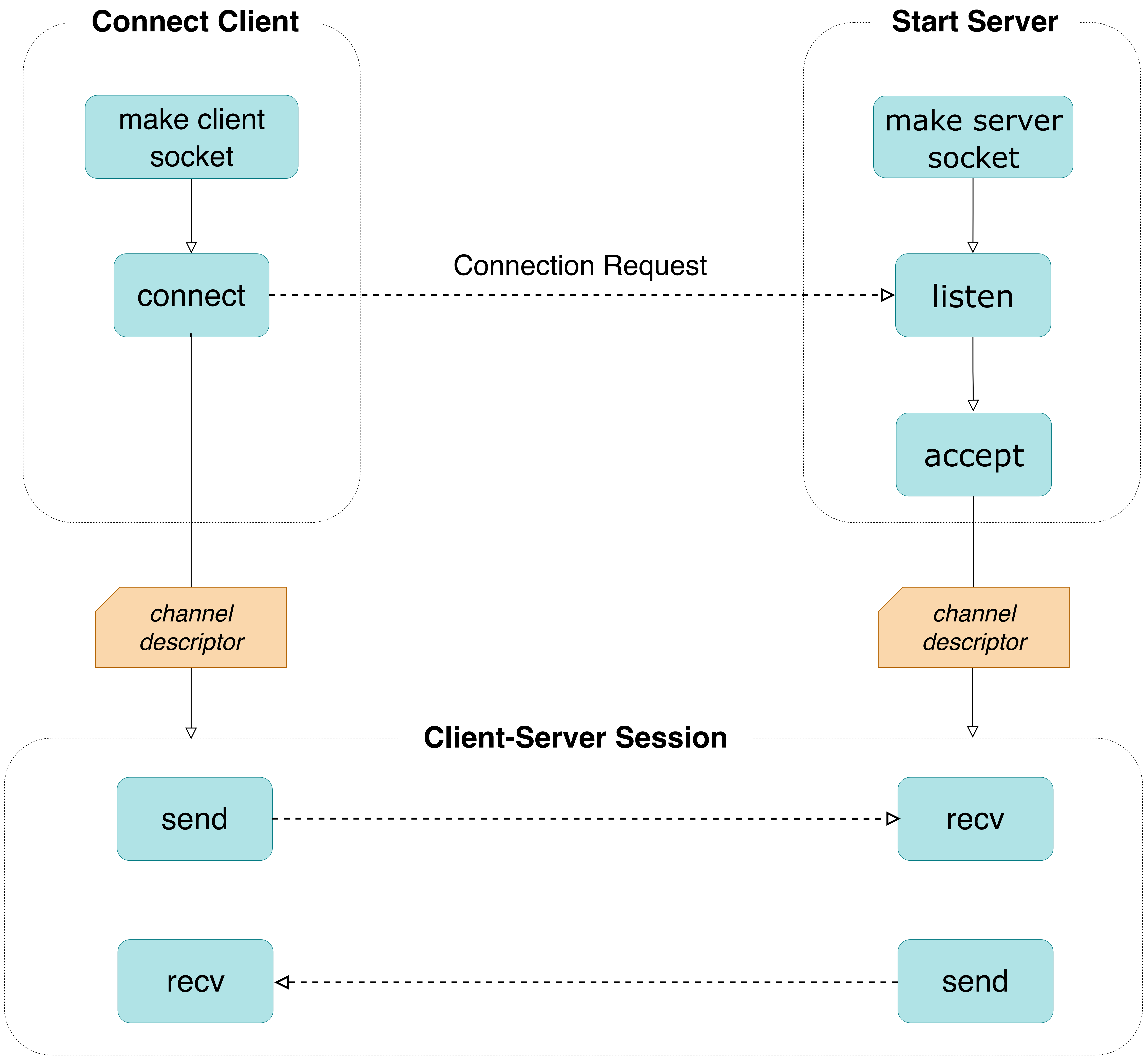
I. The API of the library

Our Library

- BSD sockets-like primitives
- 4-handshake connection
- buffered bidirectional channels
- sequence-ids/acknowledgments/ retransmission mechanisms
- ~ 350 lines of OCaml

Some design choices:

- distinction between active/passive sockets and channels
- data transfer of serialisable values



OCaml API

Explicit distinction between *active/passive socket* and *channel descriptor* datatypes

```
open Ast

  type ('a, 'b) client_skt
  type ('a, 'b) server_skt
  type ('a, 'b) chan_descr
  val make_client_skt : 'a serializer -> 'b serializer -> saddr -> ('a, 'b) client_skt
  val make_server_skt : 'a serializer -> 'b serializer -> saddr -> ('a, 'b) server_skt
  val server_listen : ('a, 'b) server_skt -> unit
  val accept : ('a, 'b) server_skt -> ('a, 'b) chan_descr * saddr
  val connect : ('a, 'b) client_skt -> saddr -> ('a, 'b) chan_descr
  val send : ('a, 'b) chan_descr -> 'a -> unit
  val try_recv : ('a, 'b) chan_descr -> 'b option
  val recv : ('a, 'b) chan_descr -> 'b
```

OCaml API

How **client** serialises values to be send to the **server**

How **server** deserialises values received from the **client**

```
open Ast

  type ('a, 'b) client_skt
  type ('a, 'b) server_skt
  type ('a, 'b) chan_descr
  val make_client_skt : 'a serializer -> 'b serializer -> saddr -> ('a, 'b) client_skt
  val make_server_skt : 'a serializer -> 'b serializer -> saddr -> ('a, 'b) server_skt
  val server_listen : ('a, 'b) server_skt -> unit
  val accept : ('a, 'b) server_skt -> ('a, 'b) chan_descr * saddr
  val connect : ('a, 'b) client_skt -> saddr -> ('a, 'b) chan_descr
  val send : ('a, 'b) chan_descr -> 'a -> unit
  val try_recv : ('a, 'b) chan_descr -> 'b option
  val recv : ('a, 'b) chan_descr -> 'b
```

OCaml API

How **server** serialises values to be send to the **client**

How **client** deserialises values received from the **server**

```
open Ast

  type ('a, 'b) client_skt
  type ('a, 'b) server_skt
  type ('a, 'b) chan_descr
  val make_client_skt : 'a serializer -> 'b serializer -> saddr -> ('a, 'b) client_skt
  val make_server_skt : 'a serializer -> 'b serializer -> saddr -> ('a, 'b) server_skt
  val server_listen : ('a, 'b) server_skt -> unit
  val accept : ('a, 'b) server_skt -> ('a, 'b) chan_descr * saddr
  val connect : ('a, 'b) client_skt -> saddr -> ('a, 'b) chan_descr
  val send : ('a, 'b) chan_descr -> 'a -> unit
  val try_recv : ('a, 'b) chan_descr -> 'b option
  val recv : ('a, 'b) chan_descr -> 'b
```

Example: echo server

```
open Ast
open Serialization_code
open Client_server_code

let int_s = int_serializer
let str_s = string_serializer

let rec echo_loop c =
  let req = recv c in
  send c (strlen req);
  echo_loop c

let accept_loop s =
  let rec loop () =
    let c = fst (accept s) in
    fork echo_loop c; loop ()
  in loop ()

let server srv =
  let s = make_server_skt int_s str_s srv in
  server_listen s;
  fork accept_loop s
```

```
let client clt srv s1 s2 =
  let s = make_client_skt str_s int_s clt in
  let c = connect s srv in
  send c s1; send c s2;
  let m1 = recv c in
  let m2 = recv c in
  assert (m1 = strlen s1 && m2 = strlen s2)

let client_0 clt srv =
  client clt srv "carpe" "diem"
```

II. Specification

Spec 1/4 : Params & Resources

Our specification of the API primitives is dependent on

- the **user parameters** provided by the user

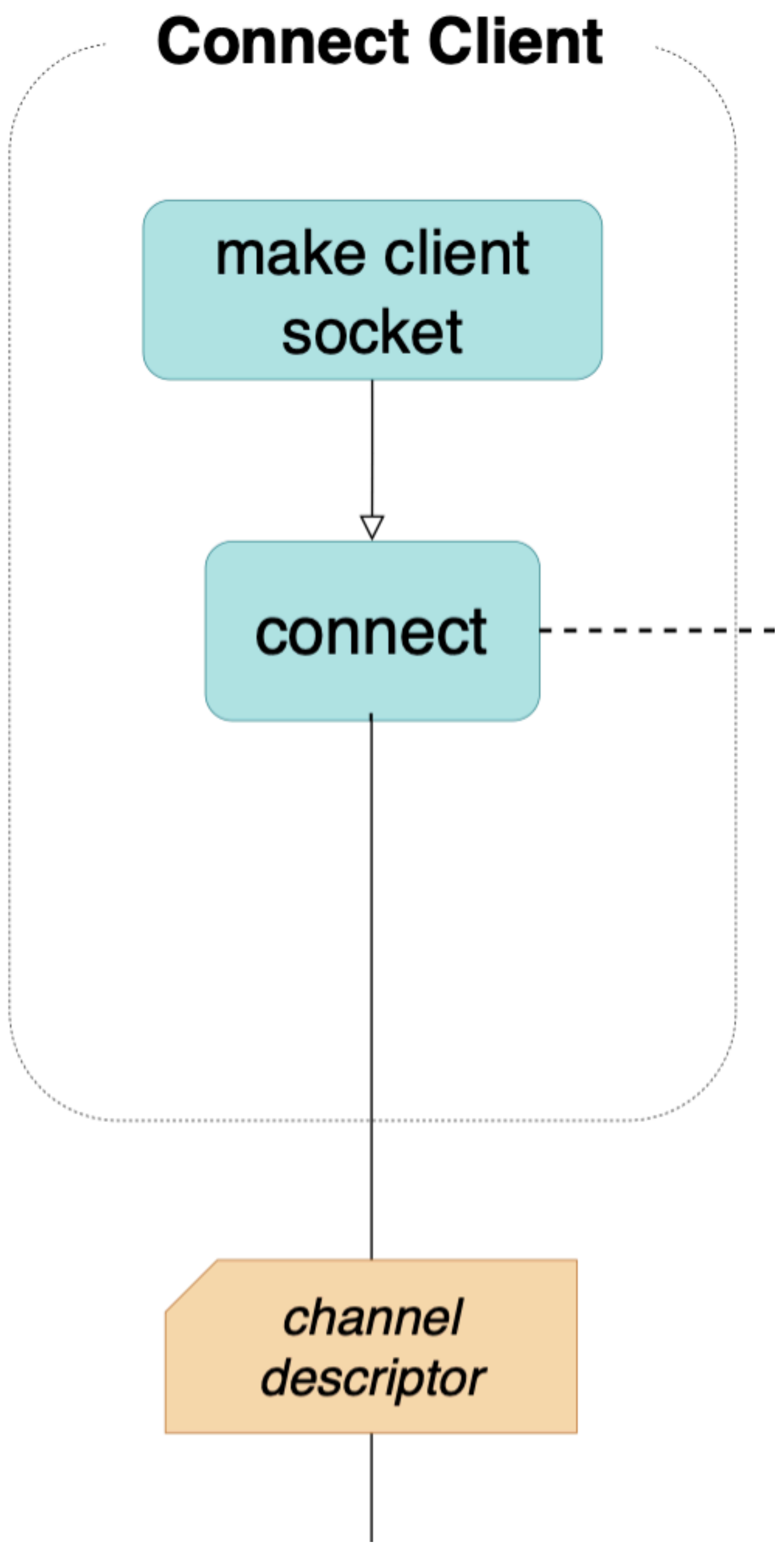
$$UP \in \mathrm{RC\_UserParams} \triangleq \{\mathtt{srv} : \mathrm{Address}; \quad \mathtt{prot} : \mathrm{iProto}; \quad \mathtt{ss} : \mathrm{Serializer}; \quad \mathtt{cs} : \mathrm{Serializer}\}$$

- and the **abstract specification resources** provided by the library

$$S \in \mathrm{RC\_Resources}\ (UP : \mathrm{RC\_UserParams}) \triangleq \left\{\begin{array}{ll} \mathtt{SrvCanInit} : \mathrm{iProp}; & \mathtt{CltCanInit} : \mathrm{Address} \to \mathrm{iProp}; \\ \mathtt{CanListen} : \mathrm{Socket} \to \mathrm{iProp}; & \mathtt{CanConnect} : \mathrm{Ip} \to \mathrm{Socket} \to \mathrm{iProp} \\ \mathtt{Listens} : \mathrm{Socket} \to \mathrm{iProp}; & \end{array}\right\}$$

$$\mathit{Notations} : \quad \mathsf{S} := \mathsf{SessionResources}(\mathsf{UP}),\ \mathsf{S.srv} := \mathsf{UP.srv}$$

Spec 2/4 : Client Setup



$\textsc{Ht-make-client-socket}$ [S]

$$\{S.\texttt{CltCanInit}\ sa\}$$
$$\langle sa.\mathsf{ip};\ \mathsf{mk\_clt\_skt}\ S.\mathsf{ss}\ S.\mathsf{cs}\ sa\rangle$$
$$\{w.\ \exists skt.\ w = skt * S.\mathsf{CanConnect}\ sa.\mathsf{ip}\ skt\}$$

$\textsc{Ht-connect}$ [S]

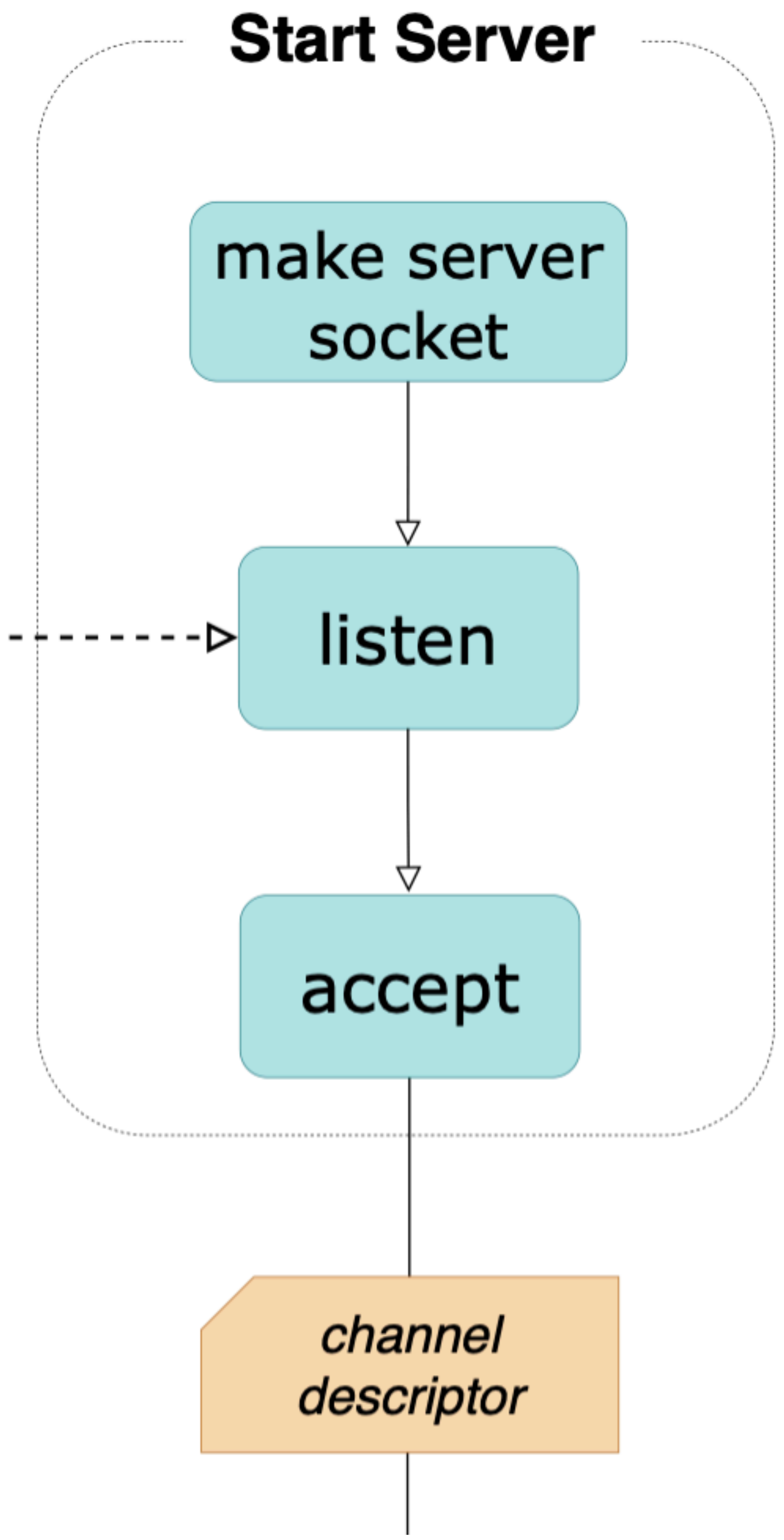
$$\{S.\mathsf{CanConnect}\ ip\ skt\}$$
$$\langle ip;\ \mathsf{connect}\ skt\ S.\mathsf{srv}\rangle$$
$$\{w.\ \exists c.\ w = c * c \xrightarrow[S.\mathsf{cs}]{sa.\mathsf{ip}} S.\mathsf{prot}\}$$

channel endpoint ownership

Spec 3/4 : Server Setup

$$\textsc{Ht-make-server-socket}\ [S]$$
$$\{S.\mathtt{SrvCanInit}\}$$
$$\langle S.\mathsf{srv.ip};\ \mathsf{mk\_srv\_skt}\ S.\mathsf{ss}\ S.\mathsf{cs}\ S.\mathsf{srv}\rangle$$
$$\{w.\,\exists skt.\ w = skt * S.\mathtt{CanListen}\ skt\}$$

$$\textsc{Ht-listen}\ [S]$$
$$\{S.\mathtt{CanListen}\ skt\}$$
$$\langle S.\mathsf{srv.ip};\ \mathsf{listen}\ skt\rangle$$
$$\{S.\mathtt{Listens}\ skt\}$$



$$\textsc{Ht-Accept}\ [S]$$
$$\{S.\mathtt{Listens}\ skt\}$$
$$\langle S.\mathsf{srv.ip};\ \mathsf{accept}\ skt\rangle$$
$$\{w.\,\exists c, sa.\ w = (c, sa) * S.\mathtt{Listens}\ skt\ * c \overset{S.\mathsf{srv.ip}}{\underset{S.\mathsf{ss}}{\rightarrowtail}} \overline{S.\mathsf{prot}}\}$$

channel endpoint ownership

Spec 4/4 : Send and Receive

HT-RELIABLE-SEND

$$\{c \xrightarrow[ser]{ip} !\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,prot * P[\vec{t}/\vec{x}] * \mathsf{Ser}\ ser\ (v[\vec{t}/\vec{x}])\}$$
$$\langle ip;\ \mathsf{send}\ c\ (v[\vec{t}/\vec{x}])\rangle$$
$$\{c \xrightarrow[ser]{ip} prot[\vec{t}/\vec{x}]\}$$

HT-RELIABLE-RECV

$$\{c \xrightarrow[ser]{ip} ?\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,prot\}$$
$$\langle ip;\ \mathsf{recv}\ c\rangle$$
$$\{w.\,\exists\vec{y}.\,w = v[\vec{y}/\vec{x}] * c \xrightarrow[ser]{ip} prot[\vec{y}/\vec{x}] * P[\vec{y}/\vec{x}]\}$$

These specs are similar to the Actris specs for message-passing concurrency and they are the same for both channel endpoints.

Workflow

- (Step 1) Writing the program(s) in the OCaml subset (done by user)
- (Step 2) Translating the programs to AnerisLang (done by compiler)
- (Step 3) Defining a Dependent Separation Protocol (done by user)
- (Step 4) Verifying each node individually (done by user)
- (Step 5) Applying the adequacy theorem to obtain a closed proof, i.e., a proof in Coq independent of Iris and Aneris, (done by user).

Echo Server Proof (1/3)

Step1: Write OCaml sources.

```
let rec echo_loop c =
  let req = recv c in
  send c (strlen req);
  echo_loop c
```

Step 2. Generate Coq definition

```
Definition echo_loop : val :=
  rec: "echo_loop" "c" :=
  let: "req" := recv "c" in
  send "c" (strlen "req");;
  "echo_loop" "c".
```

Step 3: Define the dependent separation protocol.

```
Definition prot_aux (rec : iProto Σ) : iProto Σ :=
  (<! (s : string)> MSG #s ;
   <? (n : ℕ) > MSG #n {{ ⌜String.length s = n⌝ }};
   rec)%proto.
```

Echo Server Proof (2/3)

Step 4. Instantiate the following class for echo server…

```
Class Reliable_communication_init := {
    Reliable_communication_init_setup
      E (UP : Reliable_communication_service_params):
    ↑RCParams_srv_N ⊆ E →
    ⊢ |={E}⇒
      ∃ ( _ : Chan_mapsto_resource),
      ∃ (SnRes : SessionResources UP),
        SrvInit ∗
        ⌜make_client_skt_spec UP SnRes⌝ ∗
        ⌜make_server_skt_spec UP SnRes⌝ ∗
        ⌜connect_spec UP SnRes⌝ ∗
        ⌜server_listen_spec UP SnRes⌝ ∗
        ⌜accept_spec UP SnRes⌝ ∗
        ⌜send_spec⌝ ∗
        ⌜send_spec_tele⌝ ∗
        ⌜try_recv_spec⌝ ∗
        ⌜recv_spec⌝
  }.
```

Echo Server Proof (3/3)

…and verify each node separately (modular proof).

```
Lemma wp_echo_loop c :
  {{{ c ↣{S.srv_saddr_ip, S.srv_ser} iProto_dual S.protocol }}}
      echo_loop c @[S.srv_saddr_ip]
  {{{ v, RET v ; ⊥ }}}.
Proof.
  iIntros (Φ) "Hci HΦ". iLöb as "IH". wp_lam.
  wp_recv (s₁) as "_". wp_send with "[//]".
  wp_seq.by iApply ("IH" with "[$Hci]").
Qed.
```

Step 5. Apply the adequacy theorem to obtain a closed proof, i.e., a proof in Coq independent of Iris and Aneris.

Case study: Remote Procedure Call

So far :

from Aneris rules to reason about UDP
to the logical rules for Client-Server Sessions

Distributed components :

from rules for Client Server Sessions
to the Remote Procedure Call (RPC) library

The RPC abstraction specification allows to reason about distributed applications (e.g. key-value store) without any reasoning about network-level communication at all.

RPC Spec (1/4)

RPC API:

```
type ('a, 'b) rpc
val rpc_start : 'b serializer → 'a serializer → saddr → ('a → 'b) → unit
val rpc_connect : 'a serializer → 'b serializer → saddr → saddr → ('a, 'b) rpc
val rpc_make_request : ('a, 'b) rpc → 'a → 'b
```

- The API exposes just one service handler, but in which the types of request and response are polymorphic and higher-order.
- instantiating those types with sum-types $\tau_r^1 + \tau_r^2$ (for requests), and $\tau_r^1 + \tau_r^2$ (for responses) allows us to encode an RPC service that handles multiple procedures calls e.g., as a pair of procedures of type $\tau_q^1 \to \tau_r^1$ and $\tau_q^2 \to \tau_r^2$.

RPC Spec (2/4)

As before, we use the dependent specification pattern, starting with user's parameters and library's abstract resources:

RPC User Parameters and Resources:

$$UP \in \mathsf{RPC\_UserParams} \triangleq \begin{Bmatrix} \mathsf{srv} : \mathsf{Address}; & \mathsf{ReqData} : \mathsf{Type}; \quad \mathsf{RepData} : \mathsf{Type}; \\ \mathsf{qs} : \mathsf{Serializer}; & \mathsf{pre} : \mathsf{Val} \to \mathsf{ReqData} \to \mathsf{iProp}; \\ \mathsf{rs} : \mathsf{Serializer}; & \mathsf{post} : \mathsf{Val} \to \mathsf{ReqData} \to \mathsf{RepData} \to \mathsf{iProp} \end{Bmatrix}$$

$$S \in \mathsf{RPC\_Resources}\ (UP : \mathsf{RPC\_UserParams}) \triangleq \{\mathsf{CanStart} : \mathsf{iProp}; \quad \mathsf{CanConnect} : \mathsf{Address} \to \mathsf{iProp}; \quad \mathsf{CanRequest} : \mathsf{Ip} \to \mathsf{Val} \to \mathsf{iProp}\}$$

RPC Spec (3/4)

Client-side | **Server-side**

HT-RPC-CONNECT [S]

$\{S.\mathsf{CanConnect}\ sa\}$
$\quad\langle sa.\mathsf{ip};\ \mathsf{rpc\_connect}\ S.\mathsf{qs}\ S.\mathsf{rs}\ sa\ S.\mathsf{srv}\rangle$
$\{rpc.\ S.\mathsf{CanRequest}\ sa.\mathsf{ip}\ rpc\}$

HT-RPC-REQUEST [S]

$\left\{\begin{array}{l} S.\mathsf{CanRequest}\ ip\ rpc\ * \\ S.\mathsf{pre}\ qv\ qd * \mathsf{Ser}\ S.\mathsf{qs}\ qv \end{array}\right\}$
$\quad\langle ip;\ \mathsf{rpc\_make\_request}\ rpc\ qv\rangle$
$\{rv.\ S.\mathsf{CanRequest}\ ip\ rpc * \exists rd.\ S.\mathsf{post}\ rv\ qd\ rd\}$

$\mathsf{rpc\_process\_spec}\ S\ proc \triangleq \forall qv, qd.$
$\quad\{S.\mathsf{pre}\ qv\ qd\}$
$\quad\quad\langle S.\mathsf{srv.ip};\ proc\ qv\rangle$
$\quad\{rv.\ \exists rd.\ \mathsf{Ser}\ S.\mathsf{rs}\ rv * S.\mathsf{post}\ rv\ qd\ rd\}$

HT-RPC-START [S]

$\{S.\mathsf{CanStart} * \mathsf{rpc\_process\_spec}\ S\ proc\}$
$\quad\langle S.\mathsf{srv.ip};\ \mathsf{rpc\_start}\ S.\mathsf{rs}\ S.\mathsf{qs}\ S.\mathsf{srv}\ proc\rangle$
$\{\mathsf{True}\}$

RPC Spec (4/4)

Client-side	Server-side

HT-RPC-CONNECT [S]

$\{S.\mathsf{CanConnect}\ sa\}$
$\quad\langle sa.\mathsf{ip};\ \mathsf{rpc\_connect}\ S.\mathsf{qs}\ S.\mathsf{rs}\ sa\ S.\mathsf{srv}\rangle$
$\{rpc.\ S.\mathsf{CanRequest}\ sa.\mathsf{ip}\ rpc\}$

HT-RPC-REQUEST [S]

$\left\{\begin{array}{l} S.\mathsf{CanRequest}\ ip\ rpc\ * \\ S.\mathsf{pre}\ qv\ qd * \mathsf{Ser}\ S.\mathsf{qs}\ qv \end{array}\right\}$
$\quad\langle ip;\ \mathsf{rpc\_make\_request}\ rpc\ qv\rangle$
$\{rv.\ S.\mathsf{CanRequest}\ ip\ rpc * \exists rd.\ S.\mathsf{post}\ rv\ qd\ rd\}$

$\mathsf{rpc\_process\_spec}\ S\ proc \triangleq \forall qv, qd.$
$\quad\{S.\mathsf{pre}\ qv\ qd\}$
$\quad\quad\langle S.\mathsf{srv.ip};\ proc\ qv\rangle$
$\quad\{rv.\ \exists rd.\ \mathsf{Ser}\ S.\mathsf{rs}\ rv * S.\mathsf{post}\ rv\ qd\ rd\}$

HT-RPC-START [S]

$\{S.\mathsf{CanStart} * \mathsf{rpc\_process\_spec}\ S\ proc\}$
$\quad\langle S.\mathsf{srv.ip};\ \mathsf{rpc\_start}\ S.\mathsf{rs}\ S.\mathsf{qs}\ S.\mathsf{srv}\ proc\rangle$
$\{\mathsf{True}\}$

RPC Verification (1/3)

```
let service_loop c (request_handler : 'req -> 'rep) () : unit =
  let rec loop () =
    let req = recv c in
    let rep = request_handler req in
    send c rep; loop ()
  in loop ()

let accept_new_connections_loop skt request_handler () : unit =
  let rec loop () =
    let new_conn = accept skt in
    let (c, _a) = new_conn in
    fork (service_loop c request_handler) (); loop ()
  in loop ()

let run_server
    (ser[@metavar] : 'repl serializer) (deser[@metavar] : 'req serializer) addr
    (request_handler : 'req -> 'rep) : unit  =
  let (skt :  ('repl, 'req) server_skt) = make_server_skt ser deser addr in
  server_listen skt;
  fork (accept_new_connections_loop skt request_handler) ()
```

RPC Verification (2/3)

Dependent Separation Protocol:

$$\texttt{rpc\_prot}\ (S : \texttt{RPC\_Resources}\ UP) \triangleq \mu rec.\,!\,(qv : \mathsf{Val})(qd : S.\texttt{ReqData})\ \langle qv \rangle \{S.\texttt{pre}\ qv\ qd\}.\ ?(rv : \mathsf{Val})(rd : S.\texttt{RepData})\ \langle rv \rangle \{S.\texttt{post}\ rv\ qd\ rd\}.\ rec$$

RPC Verification (3/3)

Client

$$\left\{\begin{array}{l} S.\mathsf{CanRequest}\ ip\ rpc\ * \\ S.\mathsf{pre}\ qv\ qd * \mathsf{Ser}\ S.\mathsf{qs}\ qv \end{array}\right\}$$

$$\left\{\begin{array}{l} rpc \xrightarrow[S.\mathsf{qs}]{ip} \mathsf{rpc\_prot}\ S\ * \\ S.\mathsf{pre}\ qv\ qd * \mathsf{Ser}\ S.\mathsf{qs}\ qv \end{array}\right\}$$

$\mathsf{send}\ rpc\ qv;$

$\{rpc \xrightarrow[S.\mathsf{qs}]{ip} \_\}$

$\{rpc \xrightarrow[S.\mathsf{qs}]{ip} \_\}$

$\mathsf{recv}\ rpc$

$$\left\{\begin{array}{l} rv.\ rpc \xrightarrow[S.\mathsf{qs}]{ip} \mathsf{rpc\_prot}\ S\ * \\ \exists rd.\ S.\mathsf{post}\ rv\ qd\ rd \end{array}\right\}$$

$$\left\{\begin{array}{l} rv.\ S.\mathsf{CanRequest}\ ip\ rpc\ * \\ \exists rd.\ S.\mathsf{post}\ rv\ qd\ rd \end{array}\right\}$$

Protocol

$!qv\ qd\langle qv\rangle\{S.\mathsf{pre}\ qv\ qd\}.$

$?rv\ rd\langle rv\rangle\{S.\mathsf{post}\ rv\ qd\ rd\}.$

$\mathsf{rpc\_prot}\ S$

Server

$\{c \xrightarrow[S.\mathsf{rs}]{ip} \overline{\mathsf{rpc\_prot}\ S}\}$

$\mathsf{let}\ qv = \mathsf{recv}\ c\ \mathsf{in}$

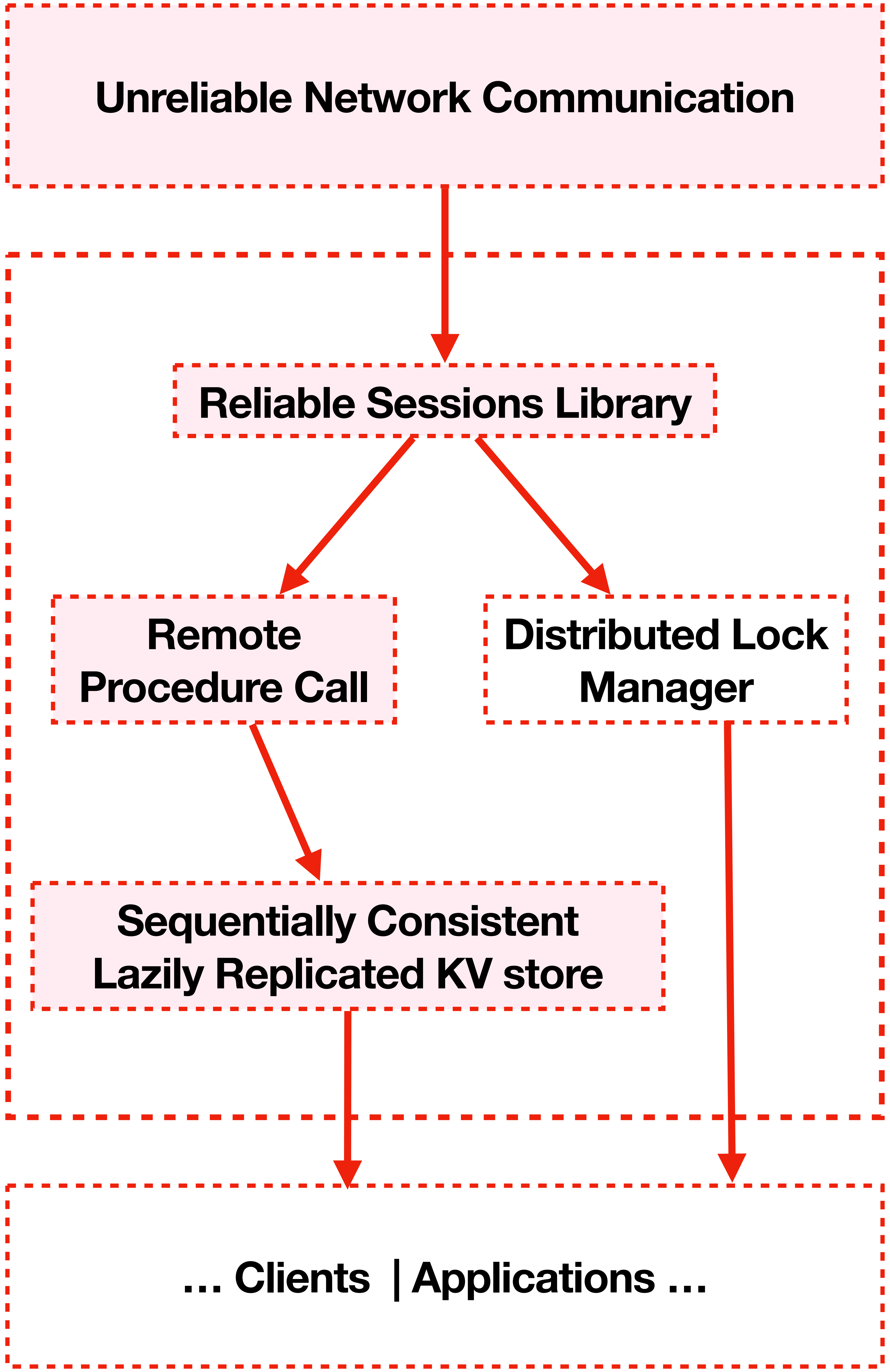
$\{c \xrightarrow[S.\mathsf{rs}]{ip} \_ * S.\mathsf{pre}\ qv\ qd\}$

$\mathsf{let}\ rv = proc\ qv\ \mathsf{in}$

$\{c \xrightarrow[S.\mathsf{rs}]{ip} \_ * S.\mathsf{post}\ rv\ qd\ rd\}$

$\mathsf{send}\ c\ rv$

$\{c \xrightarrow[S.\mathsf{rs}]{ip} \overline{\mathsf{rpc\_prot}\ S}\}$



Modular reasoning about distributed applications

LEADER-ONLY-READ-SPEC

$$\{k \mapsto^{\mathsf{ldr}}_{q} vo\}\ \langle ip;\ \mathit{read}\ k\rangle\ \{x.\ k \mapsto^{\mathsf{ldr}}_{q} vo * x = vo\}$$

LEADER-ONLY-WRITE-SPEC

$$\{k \mapsto^{\mathsf{ldr}} vo\}\ \langle ip;\ \mathit{write}\ k\ v\rangle\ \{x.\ k \mapsto^{\mathsf{ldr}} \mathsf{Some}\ v * x = ()\}$$

(Distributed Key-Value Store with Leader-Followers)

III. Verification

Verification (of established sessions)

To understand what is the **crux of the verification** (for the code when session is established), we need to take a look on

1. how **resources are transferred** for unreliable communication in Aneris Logic
2. how the **reliable transfer is modelled** in Actris Ghost Theory

The proof then proceeds in two steps:

1. connecting Actris Ghost Theory & Aneris Logic **(Session Escrow Pattern)**
2. verifying the implementation **(API send/receive and internal procedures)**

Resource Transfer in Aneris

In Aneris, safe transfer of spatial resources (associated with a sent message) over the unreliable network is achieved by

- **storing the spatial resources** in a shared logical context (Iris invariant),
- and then **sending a duplicable witness** over the network

This (**escrow pattern**) enables retransmission (as the witness is duplicable), and safe transfer (as the spatial resources can only be taken out once).

However, it **does not allow dependencies** between the resources stored in the shared logical context (indeed, there might be several resources in transit).

Actris Ghost Theory (Fragment)

Reliable transfer is modelled using logical buffers $\vec{v}_1, \vec{v}_2$ which

- describe **symmetrically** for each direction the **messages in transit**
- are governed (inside an Iris invariant) **by the shared resource** $\text{prot\_ctx}\ \chi\ \vec{v}_1\ \vec{v}_2$

$$\text{True} \Rrightarrow \exists \chi.\, \text{prot\_ctx}\ \chi\ \epsilon\ \epsilon * \text{prot\_own}_{\mathsf{l}}\ \chi\ \mathit{prot} * \text{prot\_own}_{\mathsf{r}}\ \chi\ \overline{\mathit{prot}} \qquad (\text{PROTO-ALLOC})$$

$$\begin{aligned}&\text{prot\_ctx}\ \chi\ \vec{v}_1\ \vec{v}_2 * \text{prot\_own}_{\mathsf{l}}\ \chi\ (!\,\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,\mathit{prot}) * P[\vec{t}/\vec{x}] \Rrightarrow \\ &\quad \left(\triangleright^{|\vec{v}_2|}\ \text{prot\_ctx}\ \chi\ (\vec{v}_1 \cdot [v[\vec{t}/\vec{x}]])\ \vec{v}_2\right) * \text{prot\_own}_{\mathsf{l}}\ \chi\ (\mathit{prot}[\vec{t}/\vec{x}])\end{aligned} \qquad (\text{PROTO-SEND-L})$$

$$\begin{aligned}&\text{prot\_ctx}\ \chi\ \vec{v}_1\ ([w] \cdot \vec{v}_2) * \text{prot\_own}_{\mathsf{l}}\ \chi\ (?\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\,\mathit{prot}) \Rrightarrow \\ &\quad \triangleright \exists \vec{y}.\, (w = v[\vec{y}/\vec{x}]) * P[\vec{y}/\vec{x}] * \text{prot\_ctx}\ \chi\ \vec{v}_1\ \vec{v}_2 * \text{prot\_own}_{\mathsf{l}}\ \chi\ \mathit{prot}[\vec{y}/\vec{x}]\end{aligned} \qquad (\text{PROTO-RECV-L})$$

Session Escrow Pattern

Actris Ghost Theory allows dependencies between the resources stored in the shared logical context

However,

- as such **it does not use an escrow pattern**, which is needed to connect Actris logical state with the **spatial transfer using duplicable witnesses**

- the duplicable witnesses **must appropriately reflect** the Actris logical state so that resources can be acquired in accordance to their dependence.

Message Histories

- We introduce **additional logical buffers** Tl, Rl, Tr, Rr *as a glue.*

  *(Tl, Tr) describe the **history of sent** messages;*
  *(Rl, Rr) describe the **history of received** messages (by the application)**.***

- *Various **relations** hold between Actris, glue, and physical buffers:*

  - Rr is prefix of Tl and Rl is prefix of Tr *(Internal-Coh)*
  - v1 = Tl – Rr and v2 = Tr – Rl *(Actris-Coh)*
  - sbufl is suffix of Tl and sbufr is suffix of Tr *(SBuf-Coh)*
  - rbufl is prefix of (Tr – Rl) and rbufr is prefix of (Tl – Rr) *(Rbuf-Coh)*

Session Escrow Pattern

The monotonic list ghost theory :

AUTH-LIST-ALLOC
$$\mathsf{True} \Rrightarrow \exists \gamma.\, \mathsf{auth\_list}\ \gamma\ \epsilon * \mathsf{list\_len}\ \gamma\ 0$$

AUTH-LIST-EXTEND
$$\mathsf{auth\_list}\ \gamma\ \vec{x} * \mathsf{list\_len}\ \gamma\ n \Rrightarrow \mathsf{auth\_list}\ \gamma\ (x \cdot [\vec{x}]) * \mathsf{list\_len}\ \gamma\ (n+1) * \mathsf{frag\_list}\ \gamma\ n\ x$$

AUTH-LIST-AGREE
$$\frac{\mathsf{auth\_list}\ \gamma\ \vec{x} \quad \mathsf{frag\_list}\ \gamma\ i\ x}{\vec{x}_i = x}$$

AUTH-LIST-LENGTH
$$\frac{\mathsf{auth\_list}\ \gamma\ \vec{x} \quad \mathsf{list\_len}\ \gamma\ n}{|\vec{x}| = n}$$

FRAG-LIST-DUP
$$\frac{\mathsf{frag\_list}\ \gamma\ i\ x}{\mathsf{frag\_list}\ \gamma\ i\ x * \mathsf{frag\_list}\ \gamma\ i\ x}$$

Shared logical context (Iris invariant):

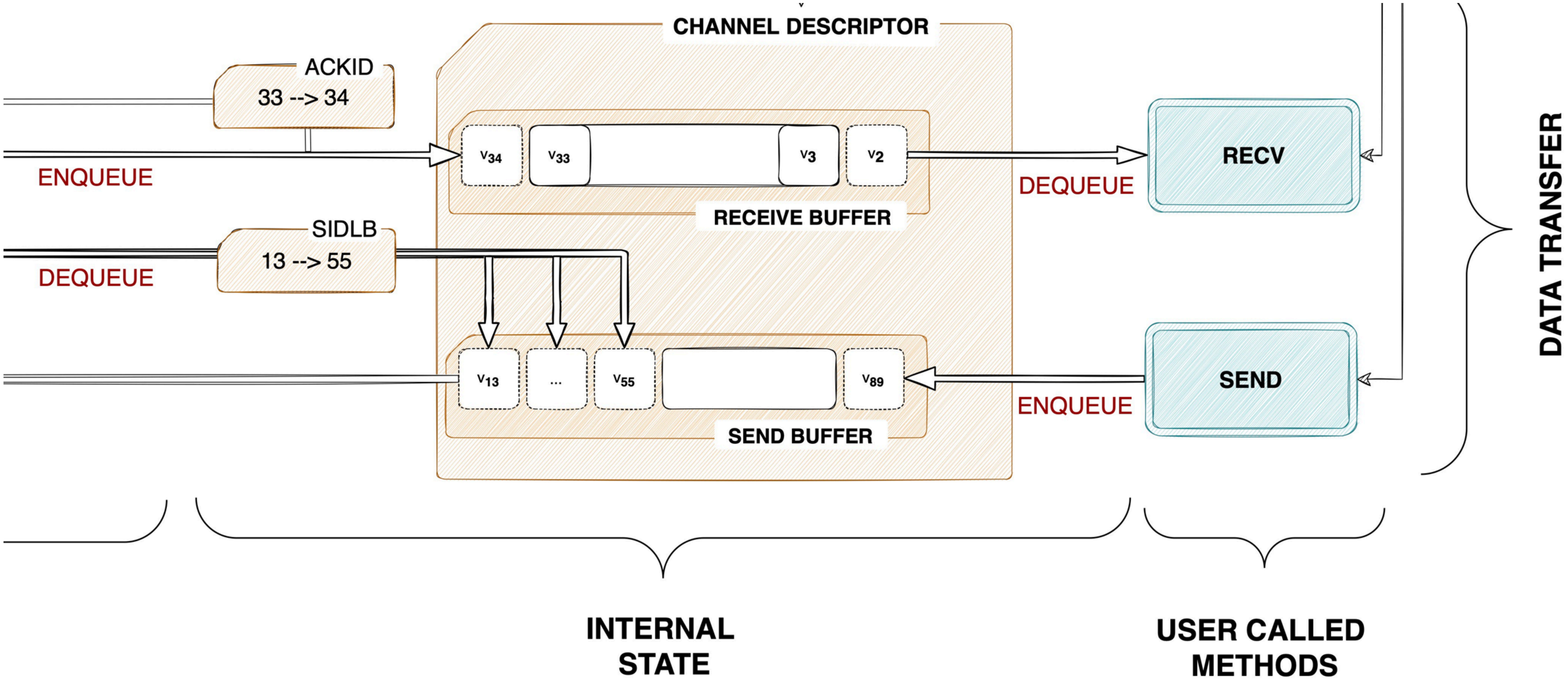
$$\exists Tl, Tr, Rl, Rr.\ \mathsf{auth\_list}\ \chi_{\mathsf{Tl}}\ Tl * \mathsf{auth\_list}\ \chi_{\mathsf{Tr}}\ Tr * \mathsf{auth\_list}\ \chi_{\mathsf{Rl}}\ Rl * \mathsf{auth\_list}\ \chi_{\mathsf{Rr}}\ Rr * \mathsf{prot\_ctx}\ \chi_{\mathsf{chan}}\ (Tl - Rr)\ (Tr - Rl) * Rr \leq_{\mathsf{p}} Tl * Rl \leq_{\mathsf{p}} Tr * ⧗\, |Tl| * ⧗\, |Tr|$$

Duplicable witnesses: $\mathsf{frag\_list}\ \chi_{Tl}\ n\ v$, $\mathsf{frag\_list}\ \chi_{Tr}\ i\ v$

Session Escrow Pattern

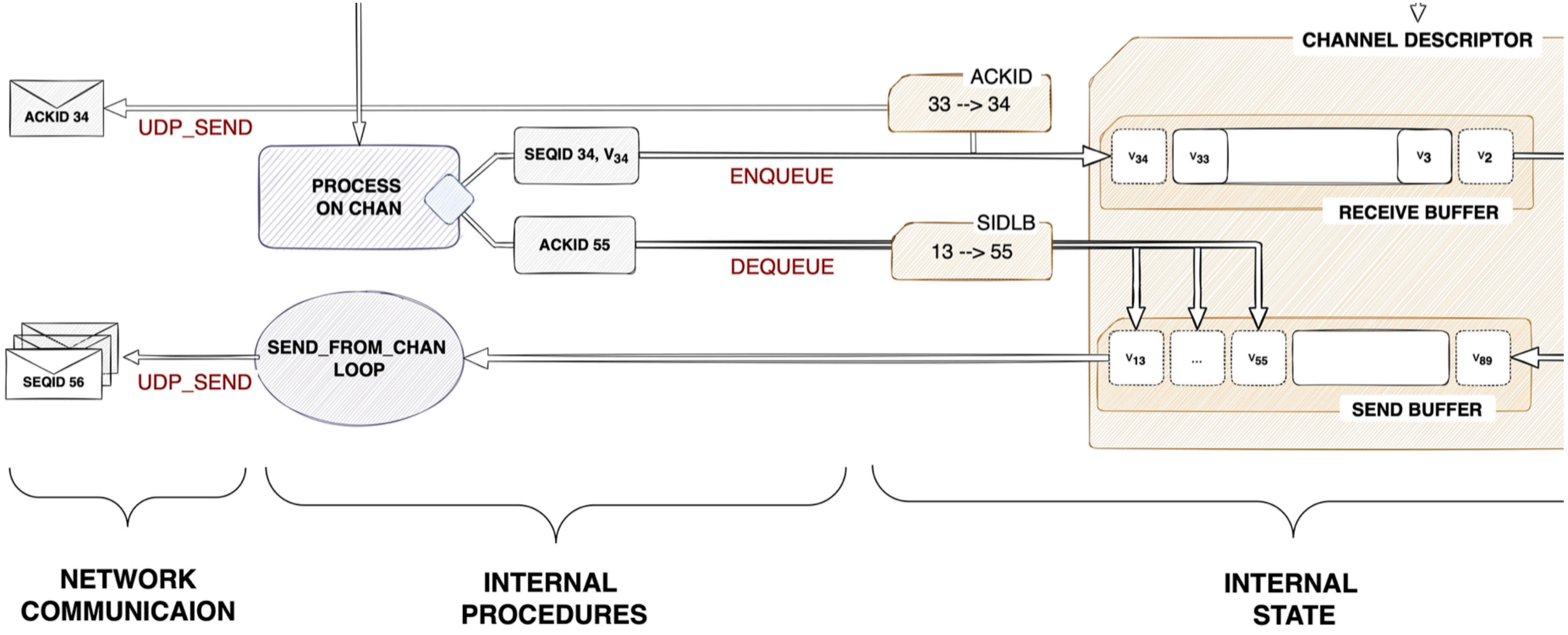
```
// Session, omitting fragments about right side
session γTl γTlc γRr ≜ ∃ Tl n, prot_ctx (drop n Tl) _ ∗ auth_list γTl Tl ∗ auth_count γTlc |Tl| ∗ auth_count
γRr n

// Session Escrow Rule for Send
session γTl γTlc γRr ⊢ prot_own_l (! xs <v> { Q } . p) ∗ frag_count γTlc n ∗     Q ==>
                       prot_own_l p ∗                        frag_count γTlc (S n) ∗ frag_list γTl n v

// Session Escrow Rule for Recv
session γTl γTlc γRr ⊢ prot_own_r (? xs <v> { Q } . p) ∗ frag_count γRr n     ∗ frag_list γTl n v ==>
                       prot_own_r p ∗                        frag_count γRr (S n) ∗ Q
```

Verification



Other Observations (1/3)

- The internal procedures that enforce the fault-tolerance are **also (mostly) the same** for clients and servers, and **so are our proofs.**

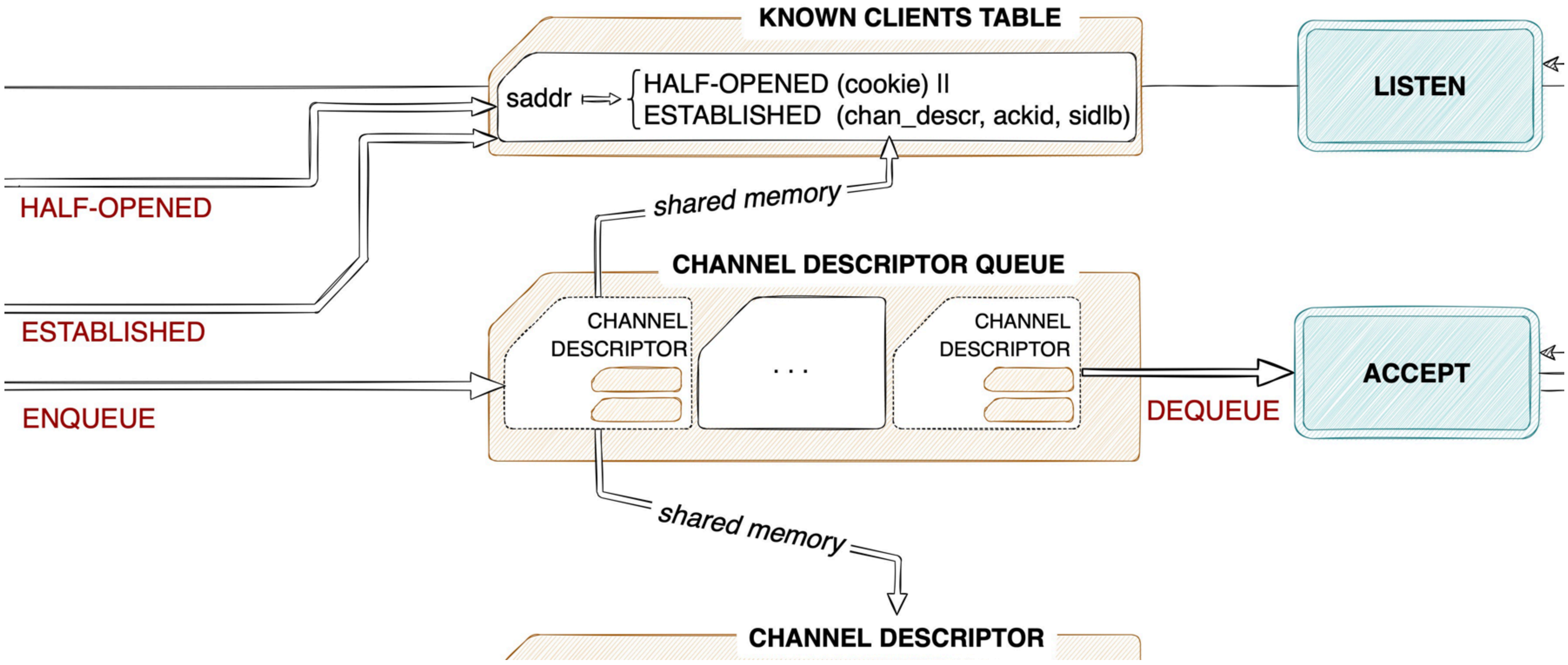


Other Observations (2/3)

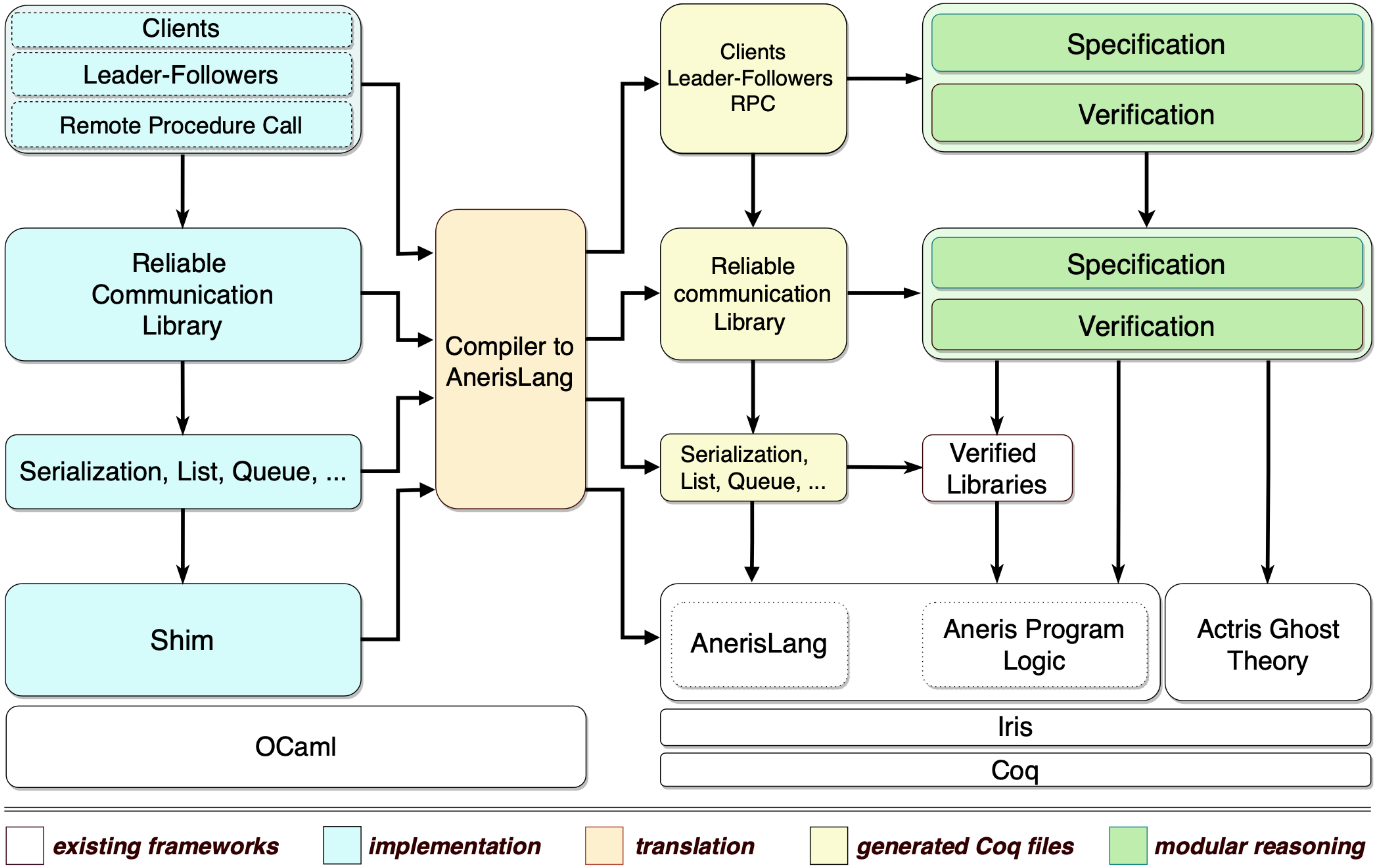
- The 4-handshake **is different for each side** and requires some effort in verification as it encodes an STS with several edge and absurd cases.

Other Observations (3/3)

- The implementation/verification of server side is more difficult, because the server must maintain **a table of known clients with their connection state** and a **channel description queue** for the established connections.



V. Conclusion & Future Directions


Clients
Leader-Followers
Remote Procedure Call
Reliable Communication Library
Serialization, List, Queue, ...
Shim
OCaml
Compiler to AnerisLang
Clients Leader-Followers RPC
Reliable communication Library
Serialization, List, Queue, ...
Specification
Verification
Specification
Verification
Verified Libraries
AnerisLang
Aneris Program Logic
Actris Ghost Theory
Iris
Coq
existing frameworks
implementation
translation
generated Coq files
modular reasoning

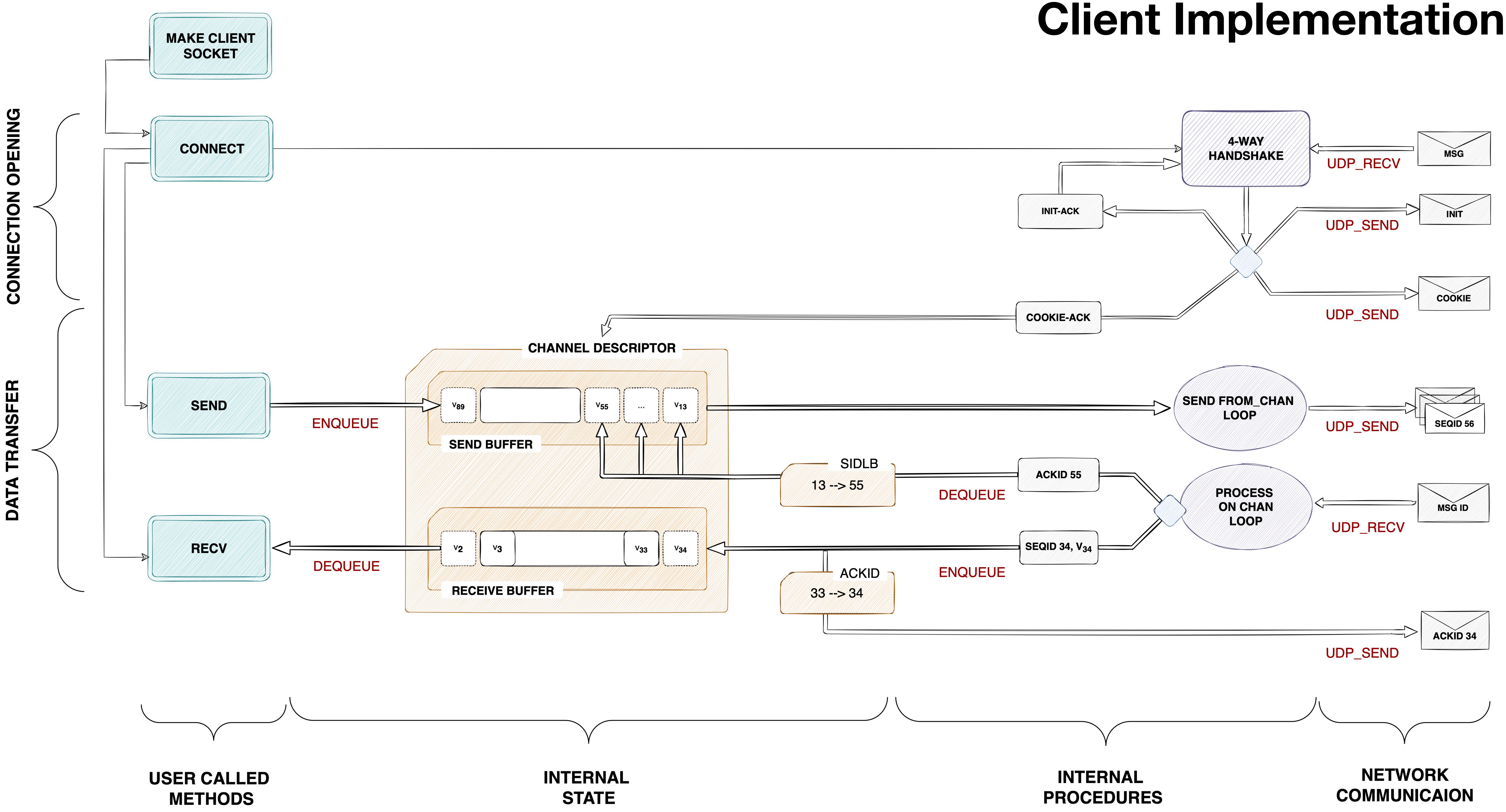
Possible Future Directions

- **Graceful/Abrupt session ending** : *detectable connection failures, reconnection*
- **Cryptography/Security**: *4-way handshake procedure / authentification / QUIC*
- **Network Partitions** : group membership/consensus built on top of our library
- **Group Communication** : *client-serv**ice** communication*
- **Transparency** : *verified libs for distributed/multithreaded programs (e.g. Functory)*
- (and maybe your insights/ideas !)

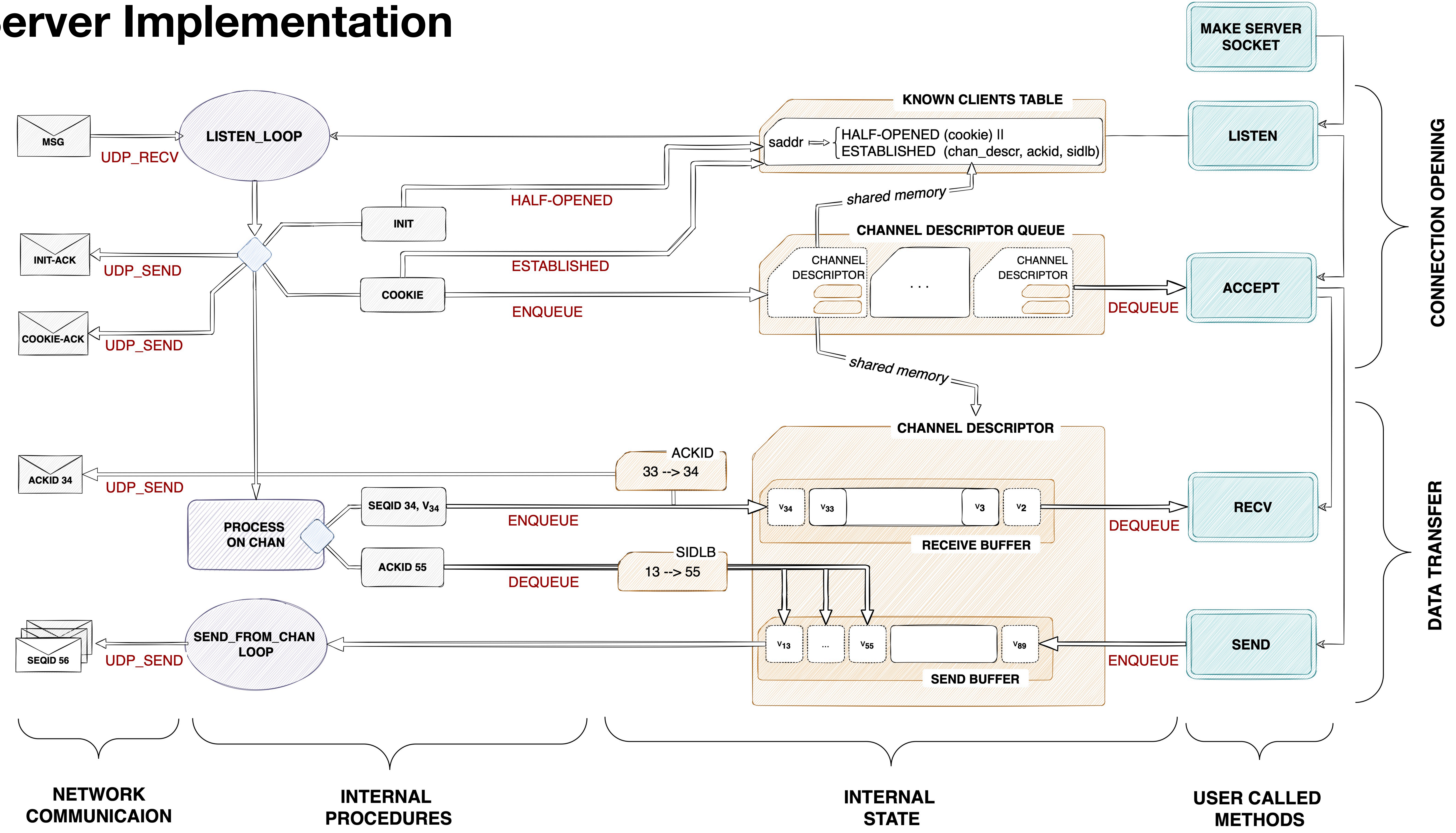
Thank you !

Backup slides

Client Implementation



Server Implementation



Protocol of the destination

SendTo

$$\left\{\begin{array}{l} a.\mathrm{ip} = ip * z \hookrightarrow_{ip} \mathrm{Some}(a) * \\ a \rightsquigarrow (R, T) * to \Rrightarrow \Phi * \Phi(m) * \\ m = \{\mathrm{body} = s; \mathrm{orig} = a; \mathrm{dest} = to\} \end{array}\right\}$$

$$\langle ip;\ \texttt{sendto}\ z\ s\ to \rangle$$

$$\left\{\begin{array}{l} v.\ v = |m| * z \hookrightarrow_{ip} \mathrm{Some}(a) * \\ \quad a \rightsquigarrow (R, T \cup \{m\}) \end{array}\right\}$$

msg must satisfy the protocol

msg has been sent

ReceiveFrom

$$\left\{\begin{array}{l} a.\mathrm{ip} = ip * z \hookrightarrow_{ip} \mathrm{Some}(a) * \\ a \rightsquigarrow (R, T) * a \Rrightarrow \Phi \end{array}\right\}$$

$$\langle ip;\ \texttt{receivefrom}\ z \rangle$$

$$\left\{\begin{array}{l} v.\ \exists m.\ v = \texttt{Some}\ (m.\mathrm{body}, m.\mathrm{orig}) * \\ \quad z \hookrightarrow_{ip} \mathrm{Some}(a) * a \rightsquigarrow (R \cup \{m\}, T) * \\ \quad m.\mathrm{dest} = a * (m \in R \lor (m \notin R * \Phi(m))) \end{array}\right\}$$

msg has been received

msg is a duplicate

msg is new, in which case it satisfies the protocol

SendToDup

$$\left\{\begin{array}{l} a.\mathrm{ip} = ip * z \hookrightarrow_{ip} \mathrm{Some}(a) * \\ a \rightsquigarrow (R, T) * m \in T * \\ m = \{\mathrm{body} = s; \mathrm{orig} = a; \mathrm{dest} = to\} \end{array}\right\}$$

$$\langle ip;\ \texttt{sendto}\ z\ s\ to \rangle$$

$$\left\{\begin{array}{l} v.\ v = |m| * z \hookrightarrow_{ip} \mathrm{Some}(a) * \\ \quad a \rightsquigarrow (R, T \cup \{m\}) \end{array}\right\}$$

if msg has been sent, then a copy can be sent for free

Remark: the proof rules for UDP primitives are low-level, but what we need is to achieve expressive specifications that abstract away most of low-level details!

Hoare Logic → Higher-Order Concurrent Separation Logic → **Distributed Separation Logic**

POSSIBLE SOLUTIONS	general-purpose solution	trusted code base	high-level specification
implement and verify reliability ad hoc for each application	✗	✓	✗
extend Aneris semantics and logics with reliable sessions primitives	✓	✗	✓
implement and verify a transport layer library on top of UDP	✓	✓	✓

Aneris Distributed Separation Logic

HT-SEND

$$\left\{ \begin{array}{l} sh \xhookrightarrow{m.\mathrm{src}_{\mathrm{ip}}} (\mathrm{Some}(m.\mathrm{src}), b) * m.\mathrm{dst} \Mapsto \Phi * \\ m.\mathrm{src} \rightsquigarrow (R, T) * (m \notin T \Rightarrow \Phi\ m) \end{array} \right\}$$

$$\langle m.\mathrm{src}_{\mathrm{ip}};\ \texttt{sendto}\ sh\ m.\mathrm{str}\ m.\mathrm{dst} \rangle$$

$$\left\{ \begin{array}{l} w.\ w = |m.\mathrm{src}| * m.\mathrm{src} \rightsquigarrow (R, T \cup \{m\}) * \\ \quad sh \xhookrightarrow{m.\mathrm{src}_{\mathrm{ip}}} (\mathrm{Some}(m.\mathrm{src}), b) \end{array} \right\}$$

HT-RECV

$$\left\{ sh \xhookrightarrow{sa_{\mathrm{ip}}} (\mathrm{Some}(sa), b) * sa \rightsquigarrow (R, T) * sa \Mapsto \Phi \right\}$$

$$\langle sa_{\mathrm{ip}};\ \texttt{receivefrom}\ sh \rangle$$

$$\left\{ \begin{array}{l} w.\ sh \xhookrightarrow{sa_{\mathrm{ip}}} (\mathrm{Some}(sa), b) * \\ \quad (b = \texttt{false} * w = \texttt{None} * sa \rightsquigarrow (R, T)) \lor \\ \quad (\exists m.\ w = \texttt{Some}\ (m.\mathrm{str}, m.\mathrm{src}) * m.\mathrm{dst} = sa * \\ \qquad sa \rightsquigarrow (R \cup \{m\}, T) * (m \notin R \Rightarrow \Phi\ m)) \end{array} \right\}$$

(a) socket handle resource $sh \xhookrightarrow{sa_{\mathrm{ip}}} (\mathrm{Some}(sa), b)$

Aneris Distributed Separation Logic

Ht-send

$$\left\{\begin{array}{l} sh \xhookrightarrow{m.\mathrm{src}_{\mathrm{ip}}} (\mathrm{Some}(m.\mathrm{src}), b) * m.\mathrm{dst} \Rrightarrow \Phi * \\ m.\mathrm{src} \rightsquigarrow (R, T) * (m \notin T \Rightarrow \Phi\ m) \end{array}\right\}$$

$$\langle m.\mathrm{src}_{\mathrm{ip}};\ \texttt{sendto}\ sh\ m.\mathrm{str}\ m.\mathrm{dst} \rangle$$

$$\left\{\begin{array}{l} w.\ w = |m.\mathrm{src}| * m.\mathrm{src} \rightsquigarrow (R, T \cup \{m\}) * \\ \quad sh \xhookrightarrow{m.\mathrm{src}_{\mathrm{ip}}} (\mathrm{Some}(m.\mathrm{src}), b) \end{array}\right\}$$

Ht-recv

$$\{sh \xhookrightarrow{sa_{\mathrm{ip}}} (\mathrm{Some}(sa), b) * sa \rightsquigarrow (R, T) * sa \Rrightarrow \Phi\}$$

$$\langle sa_{\mathrm{ip}};\ \texttt{receivefrom}\ sh \rangle$$

$$\left\{\begin{array}{l} w.\ sh \xhookrightarrow{sa_{\mathrm{ip}}} (\mathrm{Some}(sa), b) * \\ \quad (b = \texttt{false} * w = \texttt{None} * sa \rightsquigarrow (R, T)) \vee \\ \quad (\exists m.\ w = \texttt{Some}\ (m.\mathrm{str}, m.\mathrm{src}) * m.\mathrm{dst} = sa * \\ \qquad sa \rightsquigarrow (R \cup \{m\}, T) * (m \notin R \Rightarrow \Phi\ m)) \end{array}\right\}$$

(b) message history resources $sa \rightsquigarrow (R, T)$

Aneris Distributed Separation Logic

HT-SEND

$$\left\{\begin{array}{l} sh \xhookrightarrow{m.\mathrm{src}_{\mathrm{ip}}} (\mathrm{Some}(m.\mathrm{src}), b) * m.\mathrm{dst} \Mapsto \Phi * \\ m.\mathrm{src} \rightsquigarrow (R, T) * (m \notin T \Rightarrow \Phi\ m) \end{array}\right\}$$

$$\langle m.\mathrm{src}_{\mathrm{ip}};\ \texttt{sendto}\ sh\ m.\mathrm{str}\ m.\mathrm{dst}\rangle$$

$$\left\{\begin{array}{l} w.\ w = |m.\mathrm{src}| * m.\mathrm{src} \rightsquigarrow (R, T \cup \{m\}) * \\ \quad sh \xhookrightarrow{m.\mathrm{src}_{\mathrm{ip}}} (\mathrm{Some}(m.\mathrm{src}), b) \end{array}\right\}$$

HT-RECV

$$\{sh \xhookrightarrow{sa_{\mathrm{ip}}} (\mathrm{Some}(sa), b) * sa \rightsquigarrow (R, T) * sa \Mapsto \Phi\}$$

$$\langle sa_{\mathrm{ip}};\ \texttt{receivefrom}\ sh\rangle$$

$$\left\{\begin{array}{l} w.\ sh \xhookrightarrow{sa_{\mathrm{ip}}} (\mathrm{Some}(sa), b) * \\ \quad (b = \texttt{false} * w = \mathrm{None} * sa \rightsquigarrow (R, T)) \vee \\ \quad (\exists m.\ w = \mathrm{Some}\,(m.\mathrm{str}, m.\mathrm{src}) * m.\mathrm{dst} = sa * \\ \qquad sa \rightsquigarrow (R \cup \{m\}, T) * (m \notin R \Rightarrow \Phi\ m)) \end{array}\right\}$$

(c) socket protocol predicate $sa \Mapsto \Phi$