Verifying Reliable Sessions Over an Unreliable Network in Distributed Separation Logic

Léon Gondelman &
Jonas Kastberg Hinrichsen

The Second Iris Workshop

May 2, 2022

I. Reliable Communication in Distributed Systems

Communicating processes

- Network communication & message-passing concurrency:

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

  > communication protocols and resource transfer play central role

Fundamental Difference

- Communication over the network is fundamentally unreliable and asynchronous:

  > messages are lost, arrive out of order, got duplicated, or forged by adversary

  > messages arrive from one machine to another with a certain delay

  > 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)

Fault Tolerance

- Transport layer protocols such as TCP, SCTP and others provide some reliability guarantees (*at-most-once in-order delivery*).
- However, no protocol can guarantee that messages *will arrive in-order & without duplicates exactly once.*
- In the presence of network partitions/broken connections, TCP is no better than UDP: *in fine,* reliability is achieved at the application level.
- Many reasons to build fault-tolerance on top of UDP:
  > gaming community, Google QUIC (2013), Ensemble (Haiden 98)

Verification Perspective

- Two research directions:

  > **Assume** fault-tolerance **to reason** about high-level problems/algorithms:
  map-reduce, deadlock freedom, op-based CRDTs, …

  > **Model** network with faults **to build** fault-tolerance:
  consensus algorithms, reliable causal broadcast, client-server sessions.

- Longstanding goal: a unified framework where high-level abstractions meet realistic fault-tolerant implementations.

- The story of this work: one step towards this goal.

Key Observation (1/2)

- 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.

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

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

Key Observation (2/2)

- Aneris Distributed Separation Logic

  > provides rules to reason about unreliable unconnected communication;

  > had no native/library support for reliable/connected communication (i.e. each time reliability/sessions had to be built in ad-hoc way).

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

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

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

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

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

Key Observation (2/2)

- Aneris Distributed Separation Logic

  > provides rules to reason about unreliable unconnected communication;

  > had no native/library support for reliable/connected communication
  (i.e. each time reliability/sessions had to be built in ad-hoc way).

HT-SEND

$$\left\{\begin{array}{l} sh \xhookrightarrow{m.\mathrm{src}_{\mathrm{ip}}} (\mathrm{Some}(m.\mathrm{src}), b) * m.\mathrm{dst} \Rrightarrow \Phi * \\ m.\mathrm{src} \rightsquigarrow (R, T) * (m \notin T \Rightarrow \Phi\ m) \end{array}\right\}$$
$$\langle m.\mathrm{src}_{\mathrm{ip}};\ \mathtt{sendto}\ sh\ m.\mathrm{str}\ m.\mathrm{dst} \rangle$$
$$\left\{\begin{array}{l} w.\ w = |m.\mathrm{src}| * m.\mathrm{src} \rightsquigarrow (R, T \cup \{m\}) * \\ \quad sh \xhookrightarrow{m.\mathrm{src}_{\mathrm{ip}}} (\mathrm{Some}(m.\mathrm{src}), b) \end{array}\right\}$$

HT-RECV

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

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

Key Observation (2/2)

- Aneris Distributed Separation Logic

  > provides rules to reason about unreliable unconnected communication;

  > had no native/library support for reliable/connected communication (i.e. each time reliability/sessions had to be built in ad-hoc way).

HT-SEND

$$\left\{\begin{array}{l} sh \xhookrightarrow{m.\mathrm{src}_{\mathrm{ip}}} (\mathrm{Some}(m.\mathrm{src}), b) * m.\mathrm{dst} \Mapsto \Phi * \\ m.\mathrm{src} \rightsquigarrow (R, T) * (m \notin T \Rightarrow \Phi\ m) \end{array}\right\}$$
$$\langle m.\mathrm{src}_{\mathrm{ip}};\ \mathtt{sendto}\ sh\ m.\mathrm{str}\ m.\mathrm{dst} \rangle$$
$$\left\{\begin{array}{l} w.\ w = |m.\mathrm{src}| * m.\mathrm{src} \rightsquigarrow (R, T \cup \{m\}) * \\ \quad sh \xhookrightarrow{m.\mathrm{src}_{\mathrm{ip}}} (\mathrm{Some}(m.\mathrm{src}), b) \end{array}\right\}$$

HT-RECV

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

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

Our idea

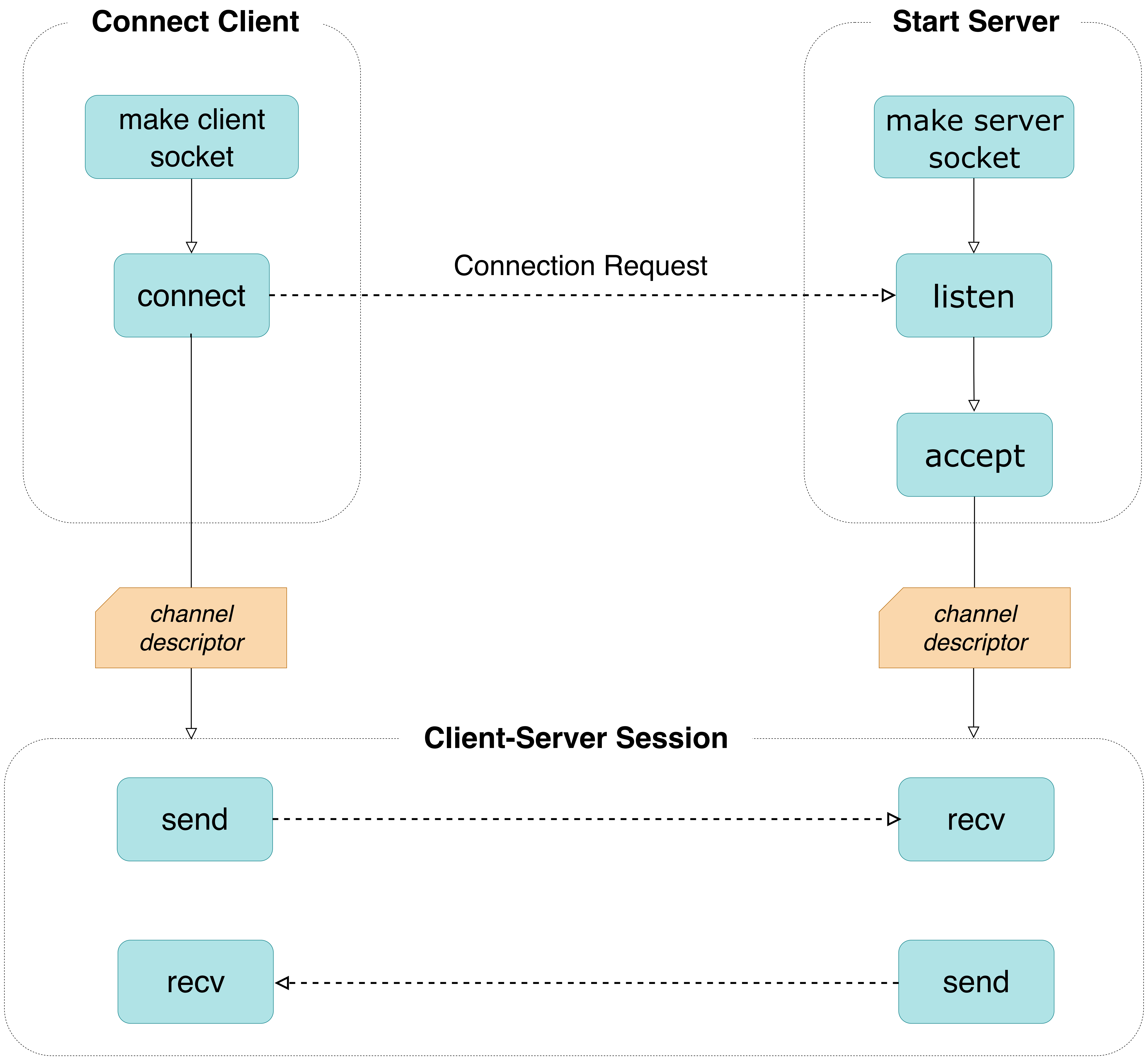
Let Aneris and Actris projects meet to enable reasoning about reliable network communication!

…The rendez-vous point is our verified client-server library.

II. 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
- 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"
```

III. Specification

Spec (1/4): Parameters & Resources

User Parameters:

$$\mathsf{UserParams} \triangleq \left\{\begin{array}{llll} \texttt{srv} & : \mathsf{Address}; & \texttt{srv\_ser} & : \mathsf{Serialization}; \\ \texttt{prot} & : \mathsf{iProto}; & \texttt{clt\_ser} & : \mathsf{Serialization}; \end{array}\right\}$$

Session Resources:

$$\mathsf{SessionResources}(\mathsf{UP} : \mathsf{UserParams}) \triangleq$$

$$\left\{\begin{array}{llll} \texttt{srv\_si} & : \mathsf{Message} \to \mathsf{iProp}; & \texttt{CanConnect} & : \mathsf{Val} \to \mathsf{Address}; \to \mathsf{iProp}; \\ \texttt{SrvInit} & : \mathsf{iProp}; & c \xrightarrow[ser]{ip} !\,\vec{x}:\vec{\tau}\,\langle v\rangle\{P\}.\ prot & (\textit{mapsto connective}); \\ \texttt{CanListen} & : \mathsf{Val} \to \mathsf{iProp}; & \textit{laws about those resources} & (\textit{e.g. subprotocols}) \\ \texttt{Listens} & : \mathsf{Val} \to \mathsf{iProp}; & & \end{array}\right\}$$

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

Spec (2/4): Client/Server Setup

Client Setup:

Ht-make-client-socket

$$\{\mathsf{FreeAddr}(clt) * clt \rightsquigarrow (\emptyset,\emptyset) * clt \neq S.\mathtt{srv}\}$$
$$\langle clt_{\mathtt{ip}};\ \mathsf{mk\_clt\_skt}\ (S.\mathtt{srv\_ser})\ (S.\mathtt{clt\_ser})\ clt\rangle$$
$$\{w.\ \exists skt.\ w = skt * S.\mathtt{CanConnect}\ clt\ skt\}$$

Ht-connect

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

Server Setup:

Ht-make-server-socket

$$\left\{\begin{array}{l}\mathsf{FreeAddr}(S.\mathtt{srv}) * S.\mathtt{srv} \rightsquigarrow (\emptyset,\emptyset) * \\ S.\mathtt{srv} \Mapsto S.\mathtt{srv\_si} * S.\mathtt{SrvInit}\end{array}\right\}$$
$$\langle S.\mathtt{srv}_{\mathtt{ip}};\ \mathsf{mk\_srv\_skt}\ S.\mathtt{srv\_ser}\ S.\mathtt{clt\_ser}\ S.\mathtt{srv}\rangle$$
$$\{w.\ \exists skt.\ w = skt * S.\mathtt{CanListen}\ skt\}$$

Ht-listen

$$\{S.\mathtt{CanListen}\ skt\}$$
$$\langle S.\mathtt{srv}_{\mathtt{ip}};\ \mathsf{listen}\ skt\rangle$$
$$\{S.\mathtt{Listens}\ skt\}$$

Ht-accept

$$\{S.\mathtt{Listens}\ skt\}\ \langle S.\mathtt{srv}_{\mathtt{ip}};\ \mathsf{accept}\ skt\rangle\ \{w.\ \exists c.\ w = (c, sa) * S.\mathtt{Listens}\ skt * c \xrightarrow[S.\mathtt{srv\_ser}]{S.\mathtt{srv}_{\mathtt{ip}}} \overline{S.\mathtt{prot}}\}$$

Spec (2/4): Client/Server Setup

Client Setup:

HT-MAKE-CLIENT-SOCKET

$$\{\mathrm{FreeAddr}(clt) * clt \rightsquigarrow (\emptyset, \emptyset) * clt \neq S.\mathtt{srv}\}$$
$$\langle clt_{\mathtt{ip}};\ \mathsf{mk\_clt\_skt}\ (S.\mathtt{srv\_ser})\ (S.\mathtt{clt\_ser})\ clt\rangle$$
$$\{w.\ \exists skt.\ w = skt * S.\mathtt{CanConnect}\ clt\ skt\}$$

HT-CONNECT

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

Server Setup:

HT-MAKE-SERVER-SOCKET

$$\left\{\begin{array}{l}\mathrm{FreeAddr}(S.\mathtt{srv}) * S.\mathtt{srv} \rightsquigarrow (\emptyset, \emptyset) * \\ S.\mathtt{srv} \Mapsto S.\mathtt{srv\_si} * S.\mathtt{SrvInit}\end{array}\right\}$$
$$\langle S.\mathtt{srv}_{\mathtt{ip}};\ \mathsf{mk\_srv\_skt}\ S.\mathtt{srv\_ser}\ S.\mathtt{clt\_ser}\ S.\mathtt{srv}\rangle$$
$$\{w.\ \exists skt.\ w = skt * S.\mathtt{CanListen}\ skt\}$$

HT-LISTEN

$$\{S.\mathtt{CanListen}\ skt\}$$
$$\langle S.\mathtt{srv}_{\mathtt{ip}};\ \mathsf{listen}\ skt\rangle$$
$$\{S.\mathtt{Listens}\ skt\}$$

HT-ACCEPT

$$\{S.\mathtt{Listens}\ skt\}\ \langle S.\mathtt{srv}_{\mathtt{ip}};\ \mathsf{accept}\ skt\rangle\ \{w.\ \exists c.\ w = (c, sa) * S.\mathtt{Listens}\ skt * c \xrightarrow[S.\mathtt{srv\_ser}]{S.\mathtt{srv}_{\mathtt{ip}}} \overline{S.\mathtt{prot}}\}$$

Spec (3/4): Reliable Data Transfer

Ht-reliable-recv

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

Ht-reliable-send

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

Spec (4/4) : Logical Setup

Init-setup

$$\mathsf{True} \Rrightarrow \exists\, S : \mathsf{SessionResources}(\mathsf{UP}).$$

$$S.\mathtt{SrvInit} \ast$$

$$(\forall sa, \textsc{Ht-make-client-socket}[S](sa)) \ast$$

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

$$(\forall skt\ sa, \textsc{Ht-connect}[S](skt, sa)) \ast$$

(*specs for listen, accept, send, recv, try_recv*)

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

OCaml function

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

Generated Coq definition

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

Protocol

```
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.
```

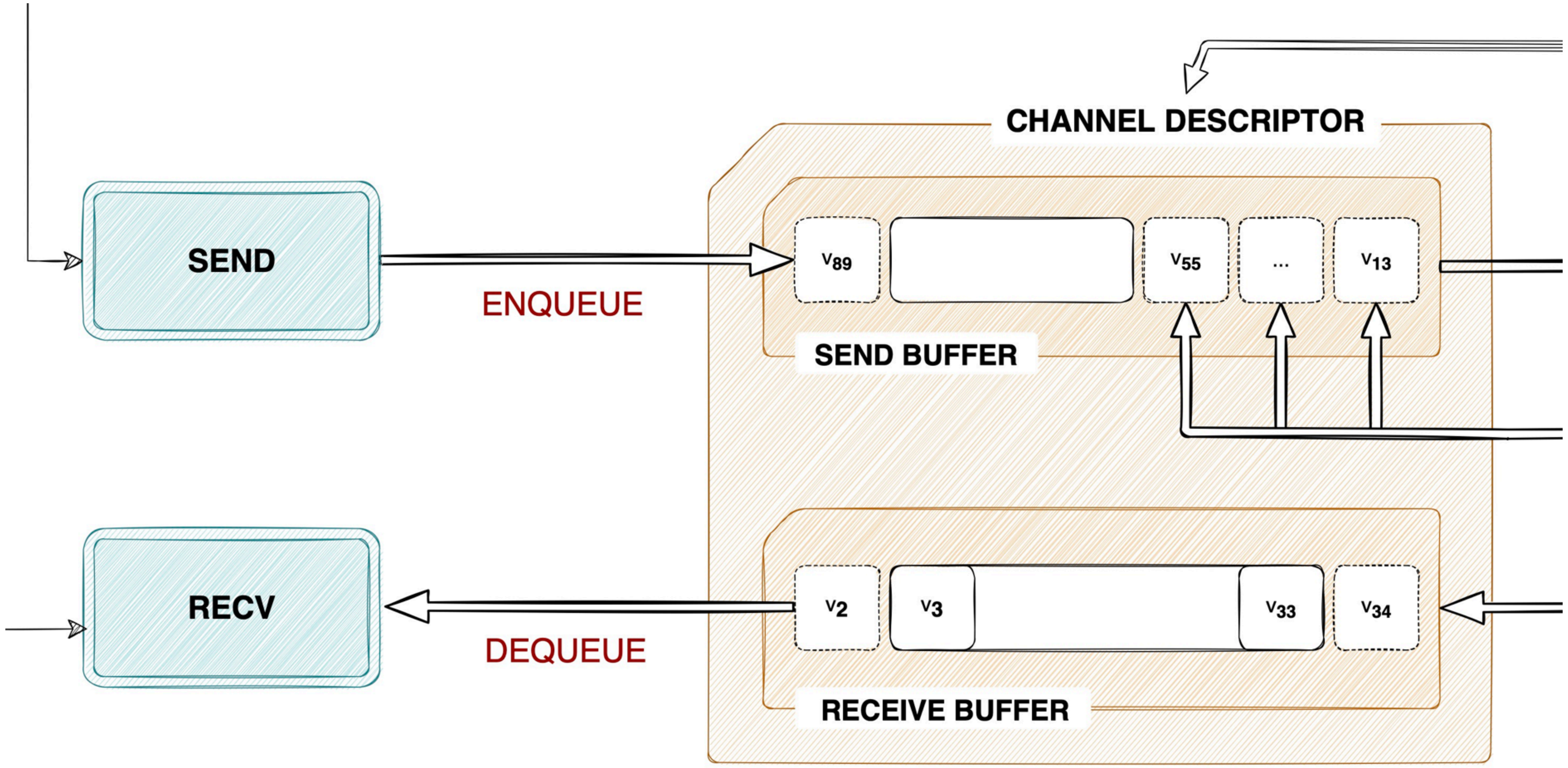
Proof of echo_loop

IV. Verification

Anatomy of send & recv

- The implementation of *send* and *recv* is the **same for client and server**. In fact, their implementation is also **agnostic of network**.

- This is possible because channels are using **in-** and **out- buffers** as indirection (calling *send* ***enqueues*** to the out-buffer, calling *recv* ***dequeues*** *from the in-buffer*)



The rendez-vous point

Crucially, this is also where the *connection* between **Actris Ghost Theory** and the **implementation** takes place. However, this connection is not immediate :

- **the two Actris logical buffers**
  > describe symmetrically for each direction the messages in transit
  > are governed (inside an Iris invariant) by the shared resource prot_ctx $\chi\ \vec{v}_1\ \vec{v}_2$
- **the four physical buffers**
  > play different role (out-buffer simply (re)transmits, in-buffer keeps data for delivery)
  > are local data of each node and are updated asynchronously

More buffers, seriously ?

- Our solution is to 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)**.***

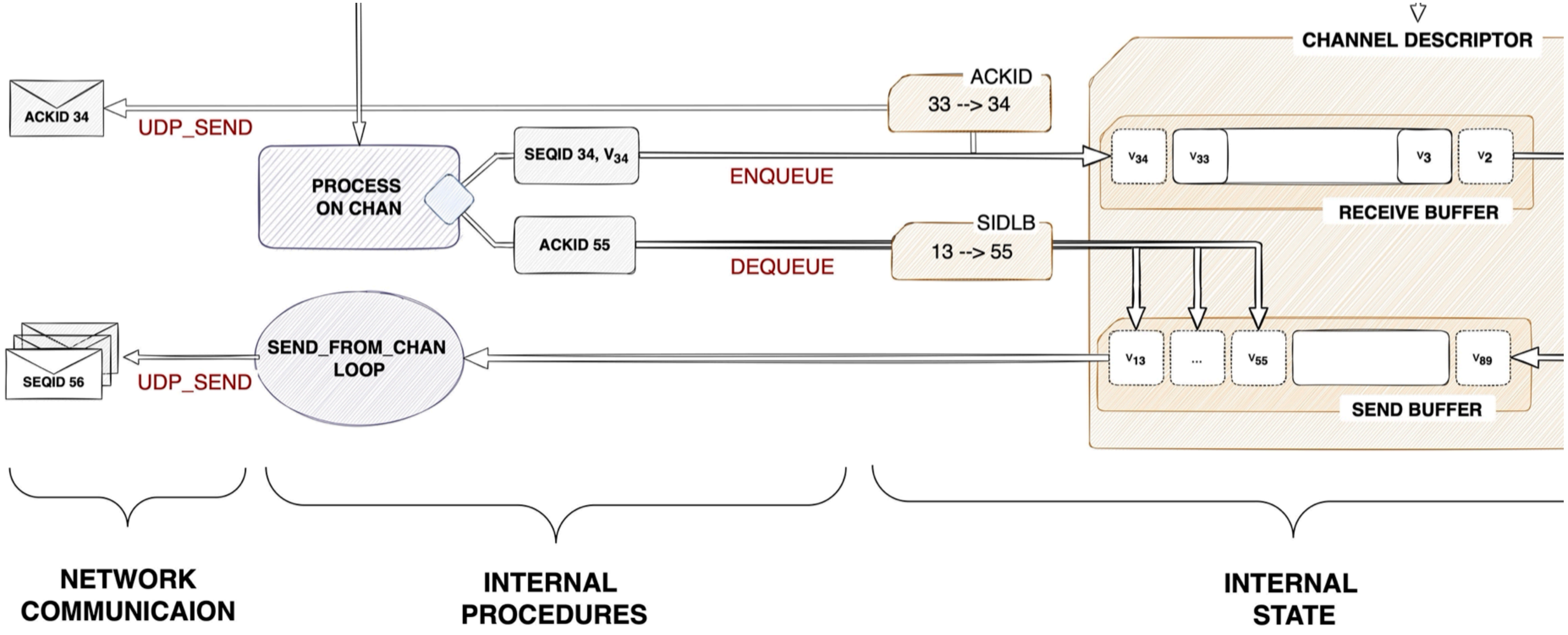
- *Various **relations** must hold between Actris, glue, and physical buffers:*

  - 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)*

- The **verification** is then primarily an effort in **preserving these relations**, in the presence of the concurrent accesses of the communication layer.

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.**

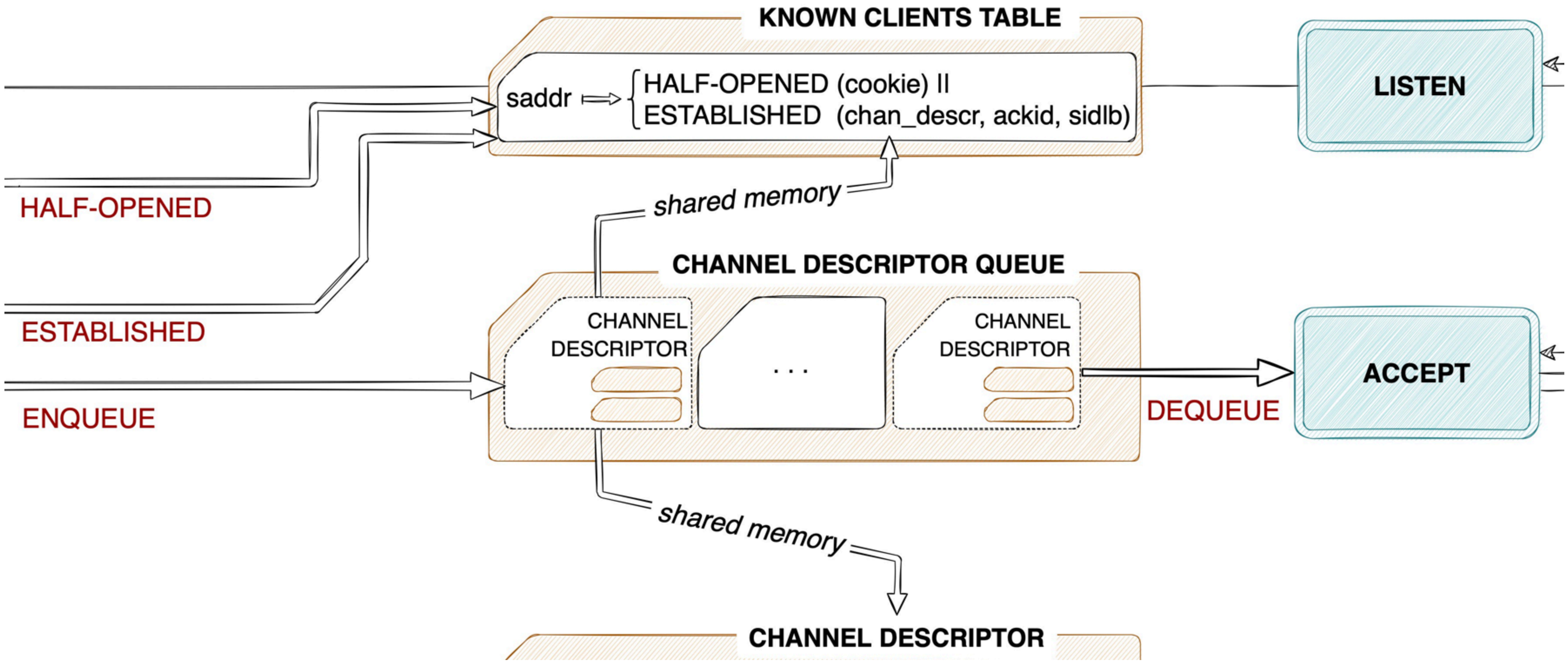


Other Observations (2/3)

- 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

Contributions

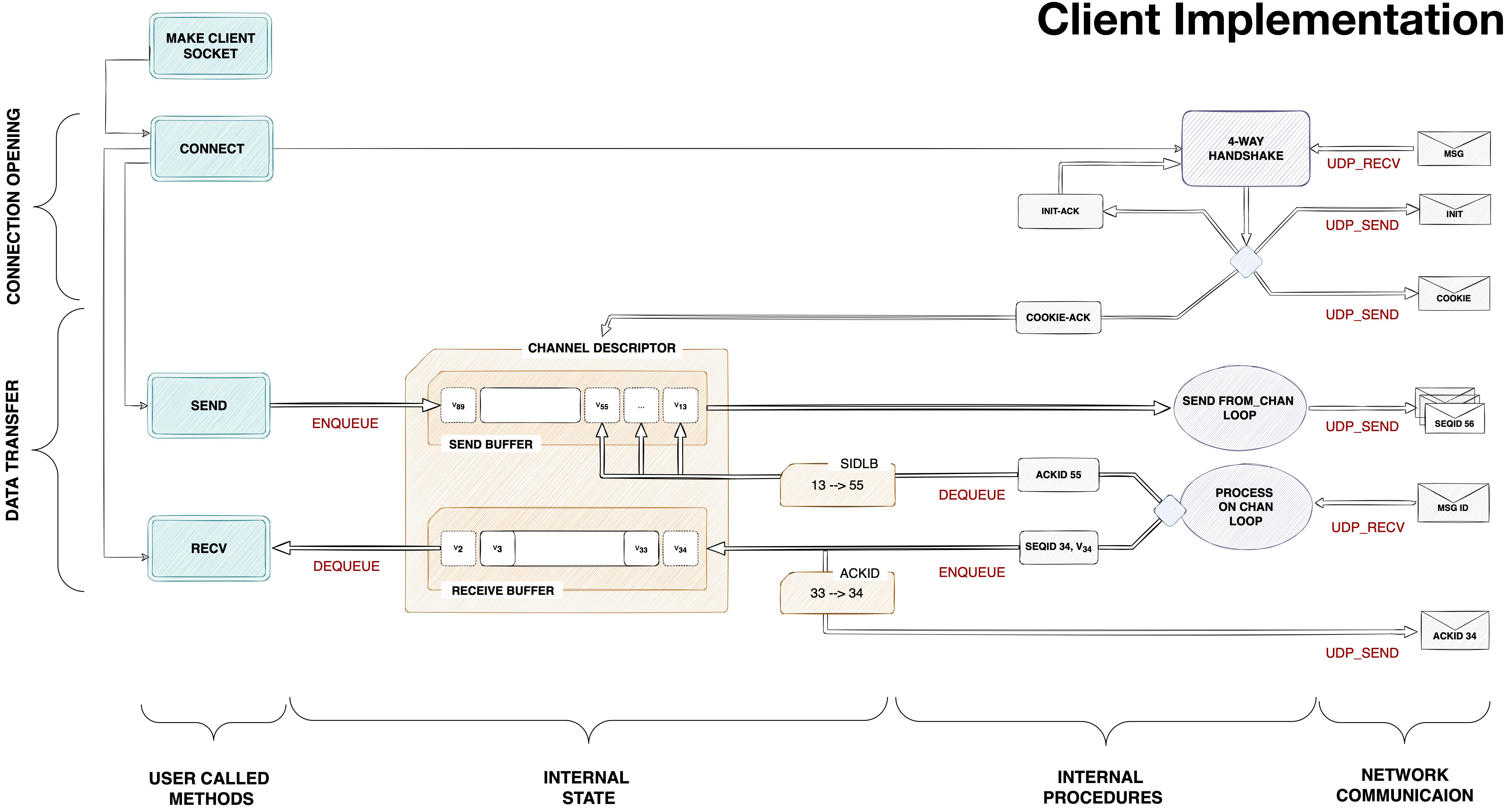
Possible Future Directions

- **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