

librice

Challenges with sans-io

2026 GStreamer conference

Matthew Waters

11 October 2026

Who Am I?

GStreamer maintainer for over a decade

WebRTC, Vulkan, OpenGL

Short overview of ICE

1. Gather candidates (Host, Server reflexive, Relayed)
2. Connectivity checks
3. Send/Receive data

Previous talks:

- 2023: [ICE: how to find your way through the internet.](#)
- 2024: [librice: a sans-io networking library](#)
- 2025: [librice: the TURNing point](#)

Current

- `librice` is successfully used by `webrtcsend/webrtcrecv` (`webrtcbin2`).
- Performance of UDP is great, TCP can be better.
- New features in the past year including:
 - Consent Freshness (RFC7675)
 - ICE-lite and ICE-restart
 - TURN support is complete
- CI now runs on Linux, macOS, and Windows

Consent Freshness (RFC 7675)

i.e. how to know when the peer is not listening anymore.

Thanks to Sanchayan for the implementation.

ICE-lite

- An ICE agent that is always publicly accessible and does not send any connectivity checks of its own.
- Multiple ICE-lite agents can run on the same UDP port.
- If no listening agent accepts the incoming data, an 'ignorable error' can be returned to the peer.

Other new functionality

Other new functionality

1. ICE-restart

Other new functionality

1. ICE-restart
2. `librice` support for choice of async runtime

Other new functionality

1. ICE-restart
2. `librice` support for choice of async runtime
3. TURN-UDP/TURN-TCP IPv4/IPv6 UDP/TCP-allocation with (D)TLS are all supported. More features than `libnice`

Challenges

Challenges

1. Time handling is complicated. e.g. `std::time::Instant` is underspecified for suspend. Handled instead by the [sans-io-time crate](#).

Challenges

1. Time handling is complicated. e.g. `std::time::Instant` is underspecified for suspend. Handled instead by the [sans-io-time crate](#).
2. Rust mutability constraints require some compromises in internal abstractions.

Challenges

1. Time handling is complicated. e.g. `std::time::Instant` is underspecified for suspend. Handled instead by the [sans-io-time crate](#).
2. Rust mutability constraints require some compromises in internal abstractions.
3. Efficiency of a sans-IO API requires considering many ways that IO is typically used.

Challenges - multiple &mut self

```
struct SansIO {  
    collection: Vec<Thing>,  
    list: Vec<Other>,  
}  
  
impl SansIO {  
    fn find_collection_item(&mut self, ...) -> Option<&mut Thing> { ... }  
  
    pub fn do_thing(&mut self) {  
        let Some(item) = self.find_collection_item() else { return; };  
        // This will fail as `self` cannot be borrowed mutably twice.  
        self.list.push(Other { ... });  
        item.do_other_thing();  
    }  
}
```

Solution - `&mut self`

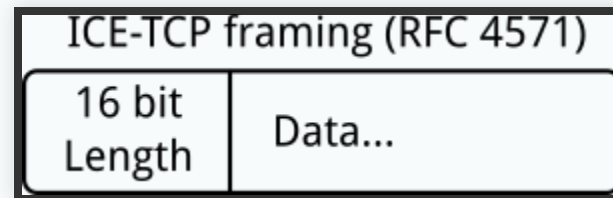
Need to somehow convince the Rust borrow checker to only partially borrow `self`.

1. Moving all `find_collection_item` invocations into the callers.
2. Re-lookup the relevant item.
3. Wrapper struct that handles lookup/insertion.
4. Future language support for partial borrows or view types.

Challenge - sans-io efficiency

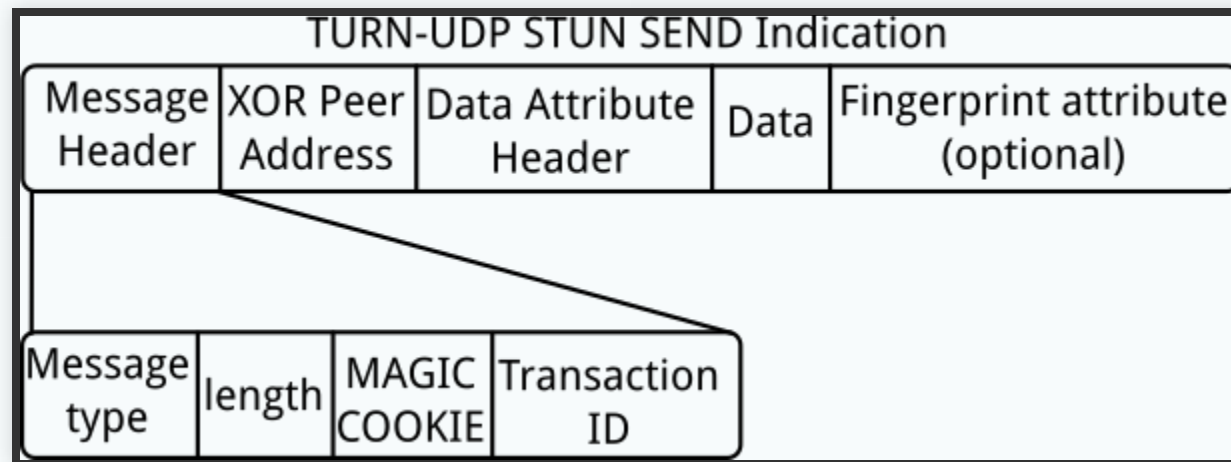
Data to be transferred can be contained within other packet structures.

e.g. TCP framing within ICE requires a 2 byte length header



Challenge - sans-io efficiency

Sending to a TURN server using UDP requires wrapping in a STUN indication, or prefixing a 4 byte header.



Solution 1 - sans-io efficiency

Wrap the provided data and build on demand

turn-client-proto/turn-server-proto currently uses this approach with [DelayedTransmitBuild](#). Currently memcpys on write.

Solution 2 - sans-io efficiency

Caller controlled buffers

- Requires caller to provide the prefix/suffix space for the data.
e.g. `DelayedTransmitBuild` would gain operations for only producing the prefix/suffix data.
- Two related questions across the layers are:
 - How much data to reserve?
 - Data alignment

Performance

Benchmarks are implemented for direct (no TURN) UDP/TCP data transfer.

TURN related benchmarks can be found in [turn-proto](#)

UDP Performance (sans-IO)

Bytes	Send Time	Send Throughput	Receive Time	Receive Throughput
32	~16ns	~2GiB/s	~37ns	~0.8GiB/s
1024	~16ns	~60GiB/s	~37ns	~25GiB/s
16000	~16ns	~919GiB/s	~37ns	~394GiB/s

Constant time UDP send/receive.

TCP Performance (sans-IO)

Bytes	Send Time	Send Throughput	Receive Time*	Receive Throughput
32	~20ns	~1.5GiB/s	~75ns	~0.4GiB/s
1024	~74ns	~13GiB/s	~75ns	~12GiB/s
16000	~604ns	~24GiB/s	~75ns	~200GiB/s

ICE-TCP has a 2 byte framing protocol (RFC 4571) that currently requires `memcpy()` of the data.

*These numbers are the happy case where blocks of received data are the same as the framing protocol.

Future expansion

Future expansion

1. Proxy support

- HTTP/1.1 CONNECT

Future expansion

1. Proxy support

- HTTP/1.1 CONNECT
- Other newer proxying methods include
 - RFC9298 - Proxying UDP in HTTP
 - RFC9484 - Proxying IP in HTTP
 - RFC draft - Template-Driven HTTP CONNECT Proxying for TCP

Future expansion

Future expansion

2. (m)DNS candidates

- STUN/TURN server dns resolution and ALTERNATE-SERVER support
- For WebRTC, . local candidates

Future expansion

2. (m)DNS candidates

- STUN/TURN server dns resolution and ALTERNATE-SERVER support
- For WebRTC, . local candidates

3. Completing support for SHA-256 STUN message integrity. Existing SHA-1 integrity is currently weak cryptographically.

Future expansion

2. (m)DNS candidates

- STUN/TURN server dns resolution and ALTERNATE-SERVER support
- For WebRTC, . local candidates

3. Completing support for SHA-256 STUN message integrity. Existing SHA-1 integrity is currently weak cryptographically.

4. Statistics (RTT, STUN message send/receive counts/bytes)

Future expansion

2. (m)DNS candidates

- STUN/TURN server dns resolution and ALTERNATE-SERVER support
- For WebRTC, . local candidates

3. Completing support for SHA-256 STUN message integrity.

Existing SHA-1 integrity is currently weak cryptographically.

4. Statistics (RTT, STUN message send/receive counts/bytes)

5. UPnP IGD and/or NAT-PMP

Possible Future: replace libnice in webrtcbin

1. Should we replace? Alternative is to ask users to use `webrtcsend/webrtcrecv` from `webrtcbin2`.
2. How? Configurable property or a hard cut?
3. When? At least when `librice` implements at least all of the features of `libnice`? Or is a subset acceptable?

I would welcome input as to whether this is something that people are interested in.

Thanks

If you have a network topology that is not supported by `librice`, I would like to know about it.

- <https://crates.io/crates/rice-proto>
- `nl.net`: <https://nlnet.nl/project/librice/>
- ystreet00 on #gstreamer on OFTC
- <https://gitlab.freedesktop.org/ystreet>
- ystreet00@floss.social on mastodon
- matthew at centricular dot com