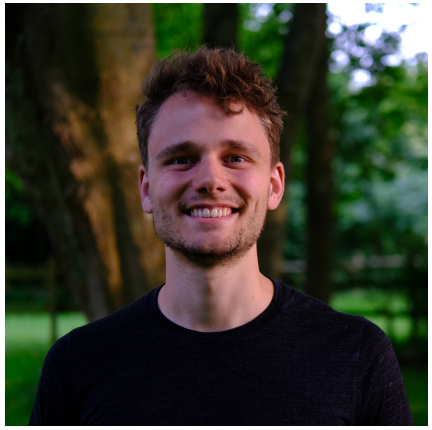
Peer-to-peer hole punching without centralized infrastructure

Max Inden (@mxinden)

April 30, 2022

Max Inden



- Software Developer at Protocol Labs.
- Stewarding the libp2p project.
- Maintainer of the libp2p Rust implementation.
- Prometheus maintainer.
- @mxinden on GitHub, Twitter, ...
- `https://max-inden.de`

Introduction to libp2p

- A modular peer-to-peer networking stack
- Composable building blocks based on a shared core
- Specified and implemented in 7+ languages
- Runs on many runtimes: browser, mobile, embedded
- Powers the IPFS, Ethereum 2, Filecoin and Polkadot network
- ~ 100 000 libp2p based nodes online at any given time

The Problem with Firewalls and NATs / Put simply

NATs

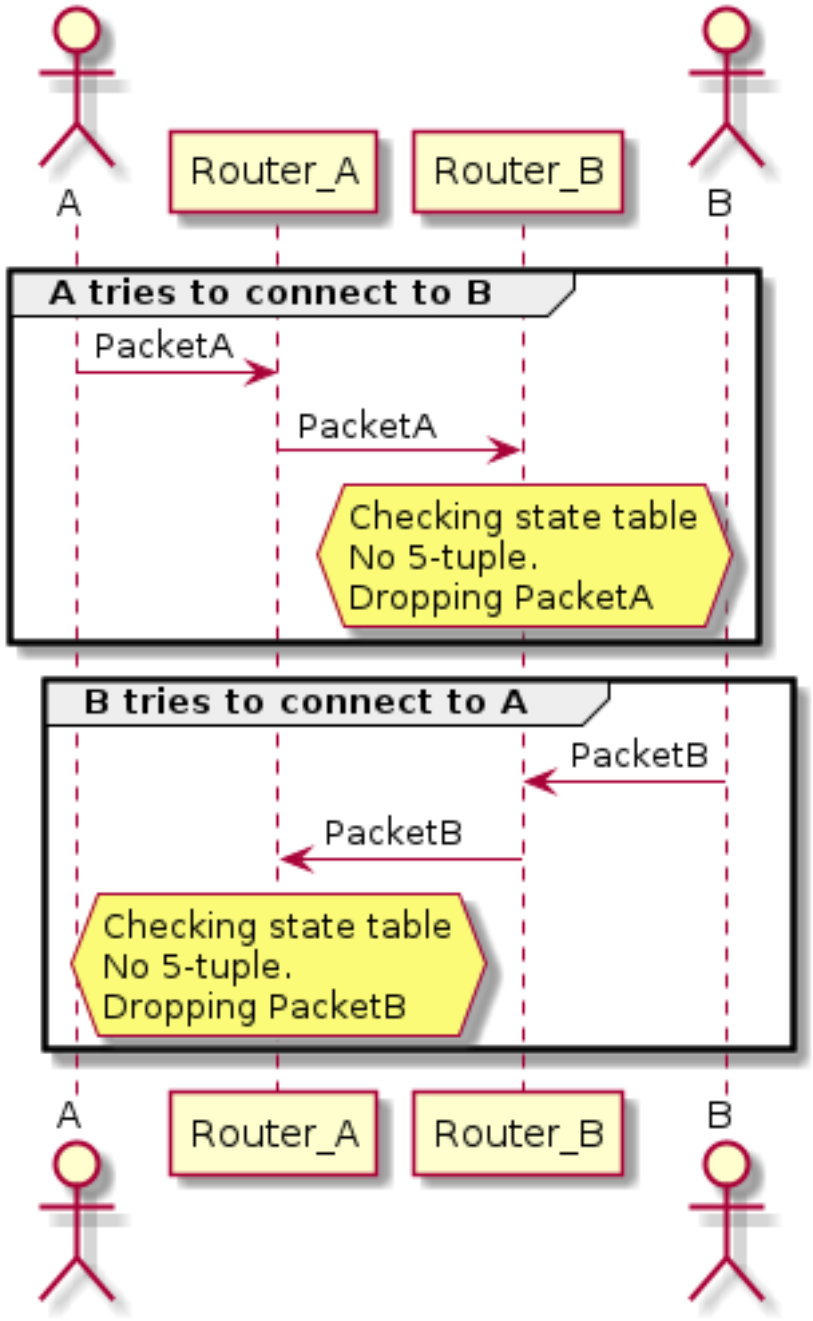
- Map your local network IP address to a public IP address

Firewalls

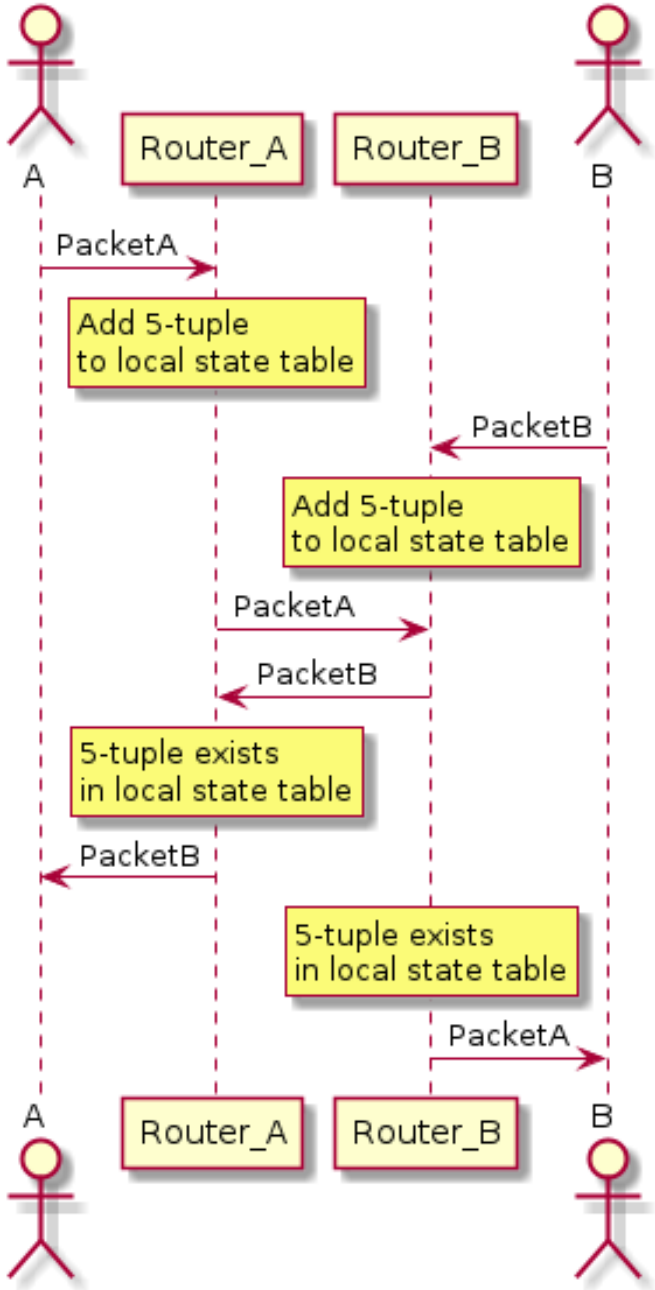
- Enforce *No packet coming in before a packet went out* via a state table

Src IP	Dst IP	Transport	Src Port	Dst Port
192.168.0.2	198.51.100.0	TCP	12345	443
192.168.0.2	198.51.100.1	UDP	12345	53
...	...	...	...	...

The Problem with Firewalls and NATs



Hole Punching



Hole Punching

Goal: Full connectivity among all nodes of a libp2p network despite NATs and Firewalls.

Requirements

- No central infrastructure
- QUIC / TCP (/ WebRTC)
- Integrate into libp2p stack

Hole Punching

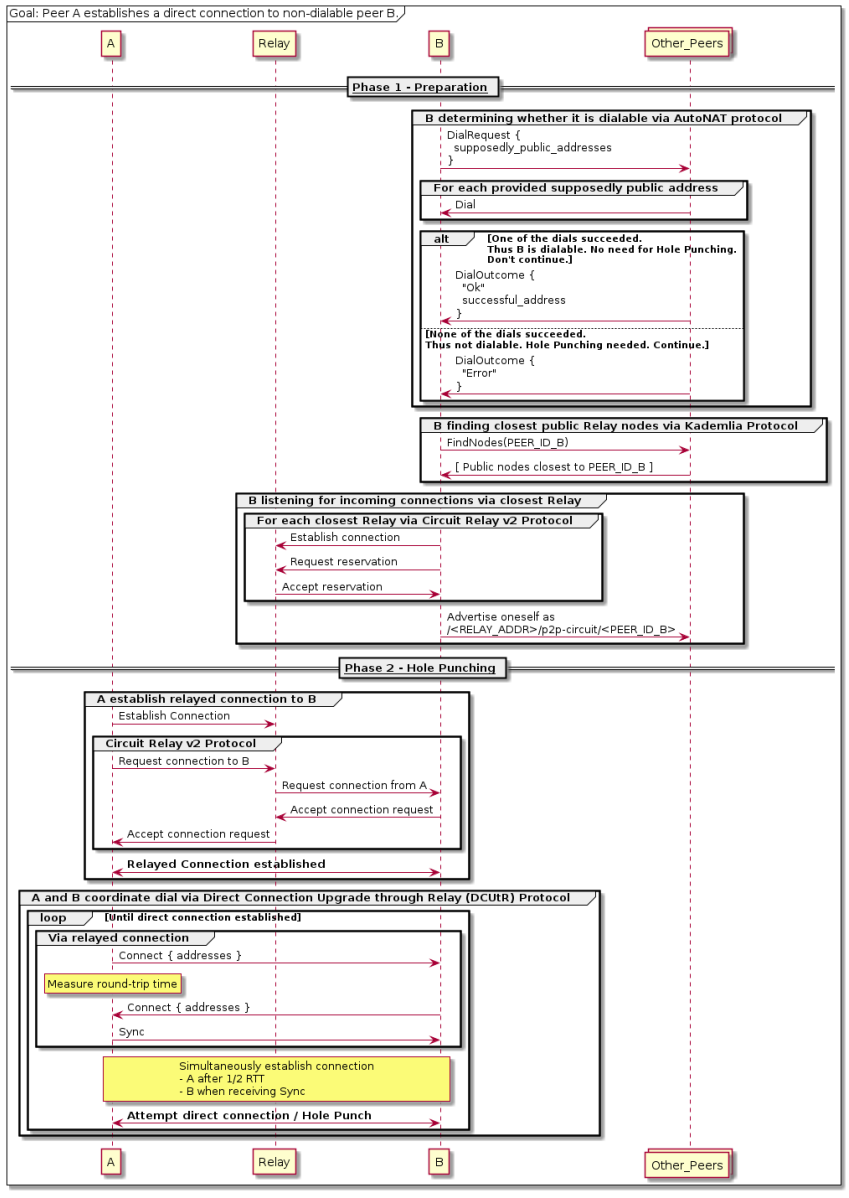
Phases

1. B: Preparing to be dialable
   - 1.1 Determine whether one is dialable (identify + AutoNAT)
   - 1.2 If not, find closest public Relay nodes (e.g. Kademlia)
   - 1.3 Listen for incoming connections via Relay (Circuit Relay)
2. A: Connecting to B
   - 2.1 A: Establish relayed connection to B (Circuit Relay)
   - 2.2 A & B: Coordinate simultaneous dial (DCUtR)

Hole Punching / Status

- Works with
  - TCP
  - UDP/QUIC (~90% success rate)
- Implemented and released in Go and Rust
- Included in IPFS
  - Public nodes run as limited relay server
  - Non-public nodes can punch holes (Not yet enabled by default)
  - Large measurement work planned

Overview


Goal: Peer A establishes a direct connection to non-dialable peer B.
A
Relay
B
Other_Peers
Phase 1 - Preparation
B determining whether it is dialable via AutoNAT protocol
DialRequest {
supposedly_public_addresses
}
For each provided supposedly public address
Dial
alt
[One of the dials succeeded.
Thus B is dialable. No need for Hole Punching.
Don't continue.]
DialOutcome {
"Ok"
successful_address
}
[None of the dials succeeded.
Thus not dialable. Hole Punching needed. Continue.]
DialOutcome {
"Error"
}
B finding closest public Relay nodes via Kademlia Protocol
FindNodes(PEER_ID_B)
[Public nodes closest to PEER_ID_B]
B listening for incoming connections via closest Relay
For each closest Relay via Circuit Relay v2 Protocol
Establish connection
Request reservation
Accept reservation
Advertise oneself as
/<RELAY_ADDR>/p2p-circuit/<PEER_ID_B>
Phase 2 - Hole Punching
A establish relayed connection to B
Establish Connection
Circuit Relay v2 Protocol
Request connection to B
Request connection from A
Accept connection request
Accept connection request
Relayed Connection established
A and B coordinate dial via Direct Connection Upgrade through Relay (DCUtR) Protocol
loop
[until direct connection established]
Via relayed connection
Connect { addresses }
Measure round-trip time
Connect { addresses }
Sync
Simultaneously establish connection
- A after 1/2 RTT
- B when receiving Sync
Attempt direct connection / Hole Punch
A
Relay
B
Other_Peers

1.1 Determine whether one is dialable

Identify

- Peers tell us what address they observe us at.

AutoNAT

- Ask public peers to dial supposedly public address.
  - On success -> public
  - On failure -> private

1.2 Find closest public Relay nodes

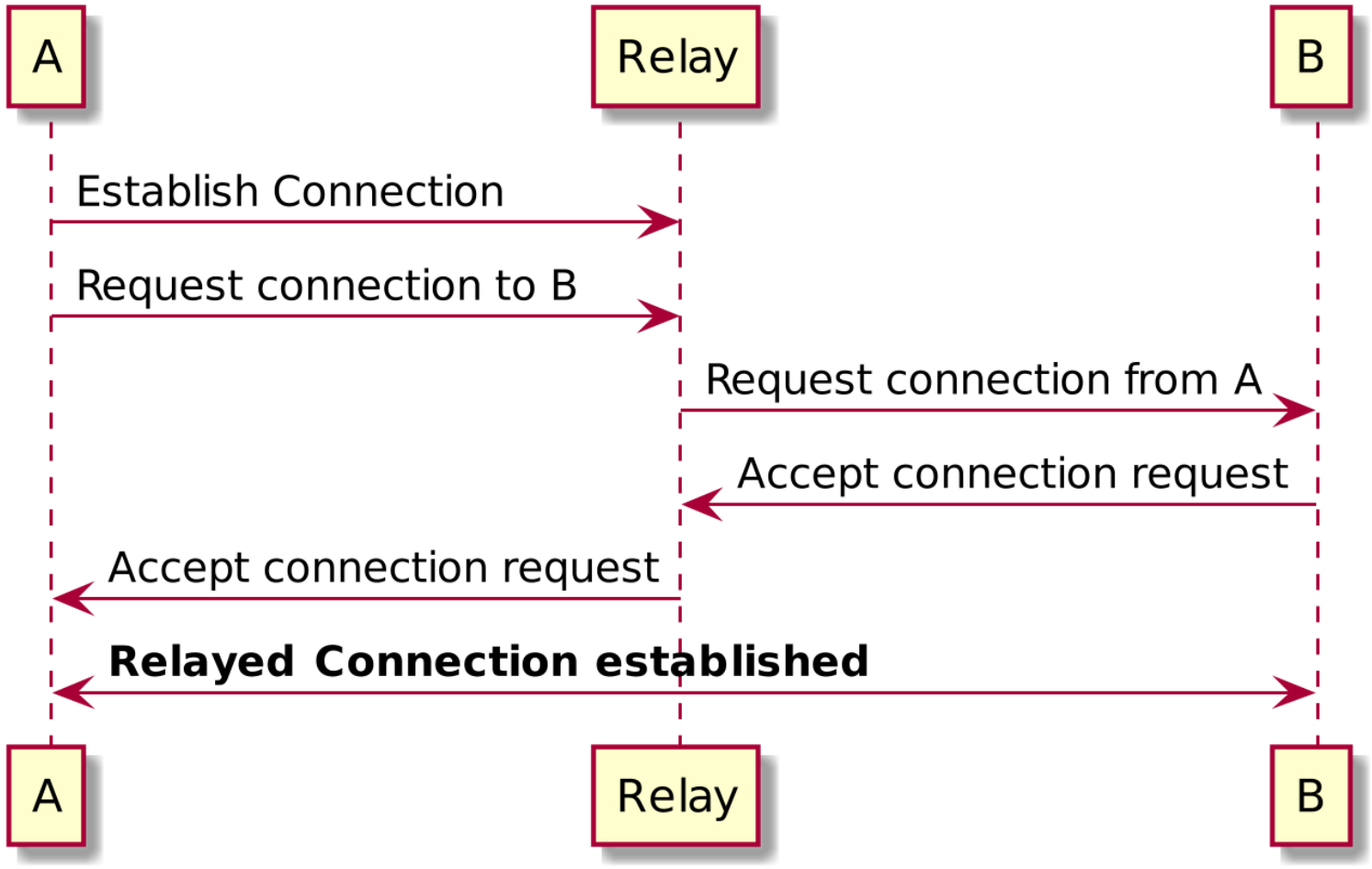
E.g. via Kademlia

- Take subset of 20 logically closest peers
- Optimize by latency

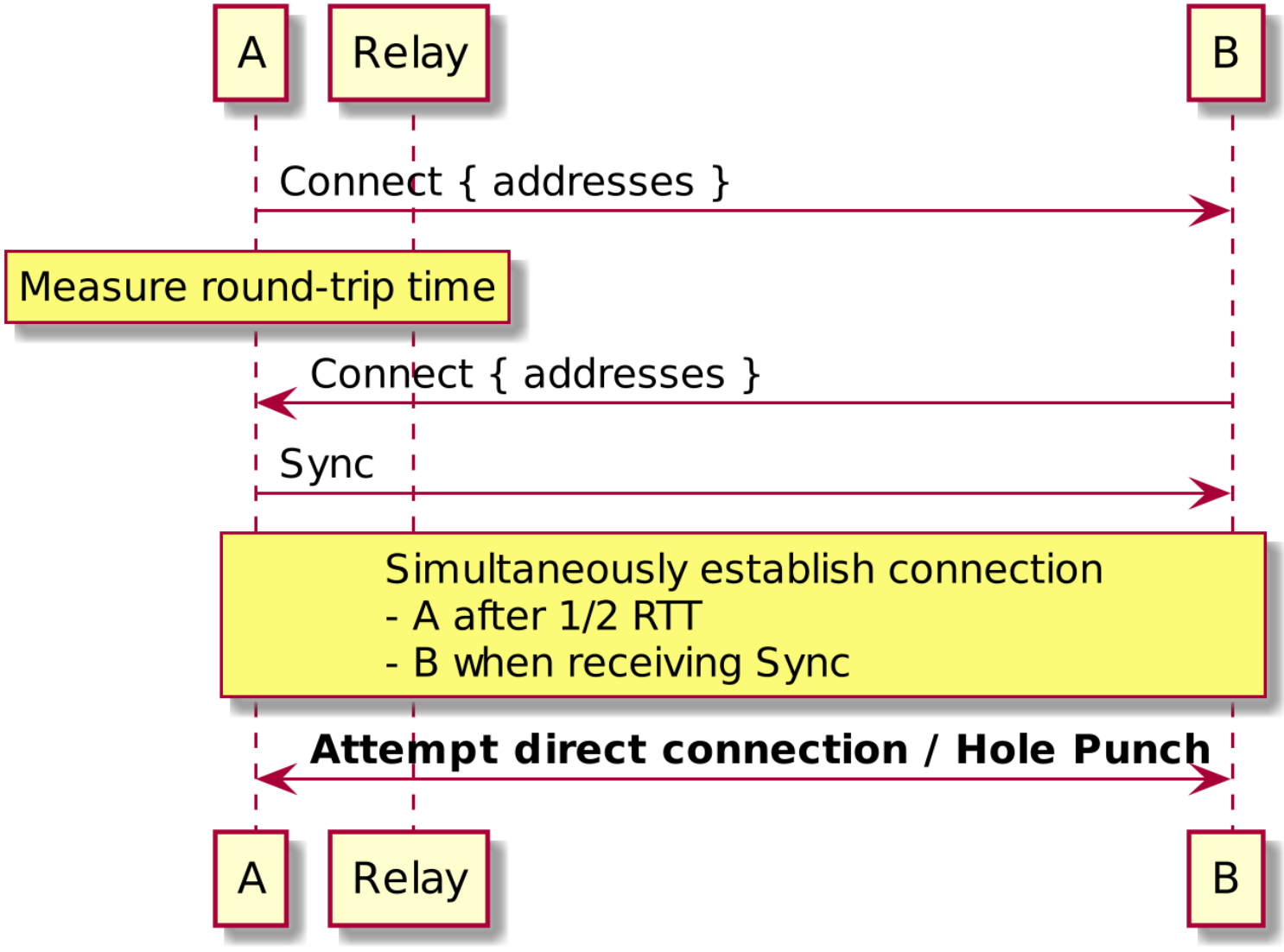
1.3 Listen for connections via Relay (Circuit Relay v2)

- Make a reservation with each relay
- Keep connection alive
- Advertise oneself as `/<RELAY_ADDR>/p2p-circuit/<PEER_ID_B>`

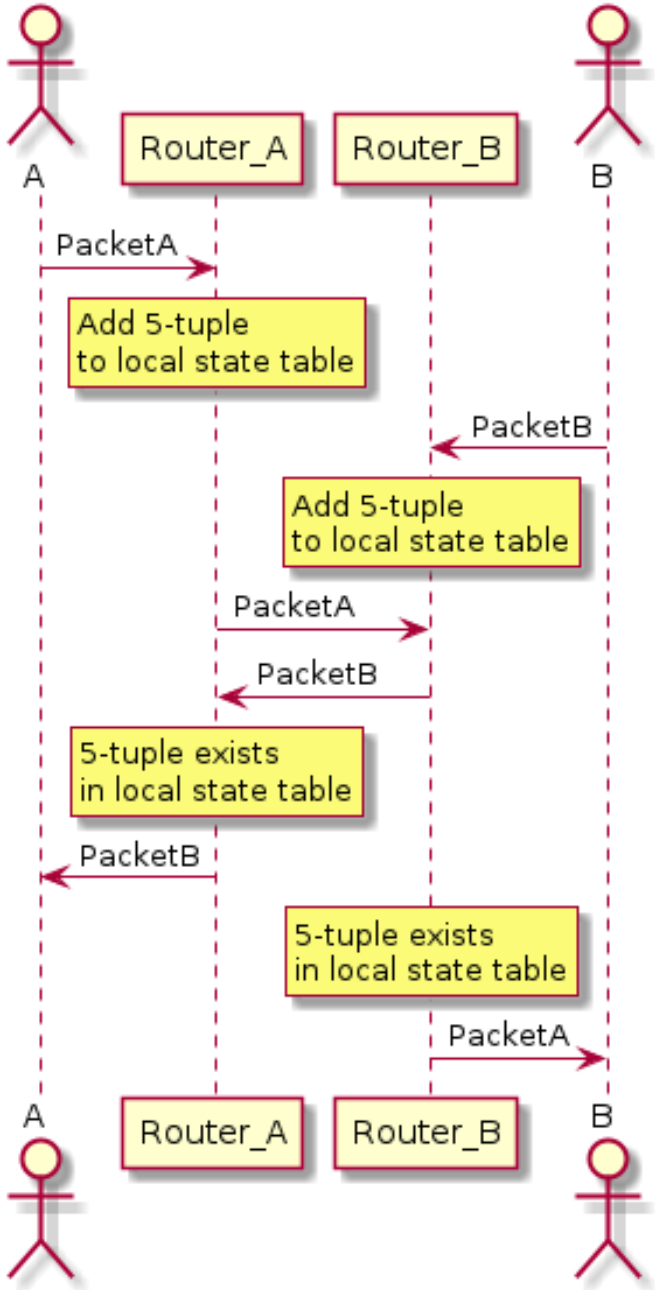
2.1 Establish relayed connection (Circuit Relay v2)



2.2. Coordinate simultaneous dial (DCUtR)



Hole Punching



How to get involved

- Read the ipfs blog post "Hole punching in libp2p"
- Follow the rust-libp2p "hole punching" tutorial

Questions?

- Talk to us here at the venue
- Documentation - docs.libp2p.io/
- Forum - discuss.libp2p.io/
- Specification & Roadmap - github.com/libp2p/specs/
- Implementations - github.com/libp2p/<LANGUAGE>-libp2p
- Join the community call

Contact

- @mxinden on GitHub, Twitter, ...
- `https://max-inden.de`