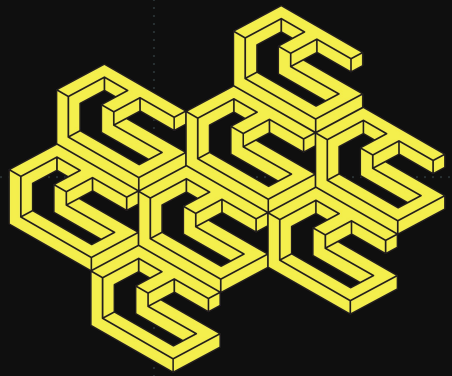
libp2p-nym Integration with Lighthouse



Timofey / Vinay
Chainsafe

Nodes

There only 2 types of nodes:

TCP-only nodes, *a la* Vanilla lighthouse.
Nym-only nodes, where the original transport module is replaced with *`rust-libp2p-nym`*.

You can also have a configuration where you have *`mixed nodes`*, where nodes support both transports through the *`libp2p circuit relay`*.

Which Node Type Should Validators Run?

- Validators interested in privacy should run nym-only nodes. Libp2p defaults to TCP connections unless otherwise specified.
- Nodes that don’t care for privacy but are happy to help those who do should run *mixed* nodes.
- Those that aren't concerned about privacy or about this enhancement will continue to run plain clients with the TCP-only transport.

Protocol

- The validator is chosen to be a block producer, it's running a nym-only node and has at least one more nym-only or *mixed* peer in its peer-store, and, therefore, in gossipsub topic meshes too.
- The block producer constructs the block and attempts to send it to peers in its block-sync topic mesh. It sends it through the nym mixnet to nym addresses that are encoded in libp2p multiaddr. All tcp-only nodes are skipped.
- A running nym-only or mixed peer receives the block from the mixnet without knowing who sent it. They can ACK to it, which will be handled via SURBs.
- After this first *shielded hop,* all other hops can be conducted through regular TCP transport to avoid additional latency. The block producer's identity is already anonymized.

Integration Components:

Transport Constructors

- Construct transport handlers wrapped in shared `BoxedTransport` type to be used in the libp2p swarm.
- `build_nym_transport()` initiates a `NymTransport` and wraps it in the `BoxedTransport` type.

```rust
/// Build a libp2p transport that supports NYM.
pub async fn build_nym_transport(
    local_private_key: Keypair,
    nym_client_uri: String,
    address: &mut Multiaddr,
) -> std::io::Result<BoxedTransport> {
    let nym = NymTransport::new(&nym_client_uri, local_private_key.clone())
        .await
        .map_err(|e| std::io::Error::new(std::io::ErrorKind::Other, e))?;

    *address = nym.listen_addr.clone();

    Ok(nym.map(|a, _| (a.0, StreamMuxerBox::new(a.1))).boxed())
}
```

`build_nym_transport` *function (source)*

Integration Components:

Transport Constructors

- `build_transport()` constructs a libp2p circuit relay transport consisting of both TCP and Nym transport modules and wraps it in a `BoxedTransport`.

```rust
/// Build a transport that supports both TCP and NYM connections.
pub async fn build_transport(
    local_private_key: Keypair,
    nym_client_uri: String,
    address: &mut Multiaddr,
) -> std::io::Result<BoxedTransport> {
    let (tcp, nym) = futures::join!(
        build_tcp_transport(local_private_key.clone()),
        build_nym_transport(local_private_key, nym_client_uri, address)
    );

    Ok(tcp?
        .or_transport(nym?)
        .map(|either_output, _| match either_output {
            Either::Left((peer_id, muxer)) => (peer_id, StreamMuxerBox::new(muxer)),
            Either::Right((peer_id, muxer)) => (peer_id, StreamMuxerBox::new(muxer)),
        })
        .boxed())
}
```

`build_transport` *function (source)*

Integration Components:

Select Transport in Network Service Constructor

- The *libp2p_transport* argument in the Network service constructor selects the Transport.

```rust
// Set up the transport - tcp/ws with noise and mplex
let (transport, bandwidth) = match config.libp2p_transport {
    Libp2pTransport::Tcp => build_tcp_transport(local_keypair.clone()).await,
    _ => {
        let nym_addr = config.nym_client_address.clone();
        let uri = format!("ws://{}:{}", nym_addr.addr, nym_addr.tcp_port);
        info!(log, "Connecting to nym client"; "address" => &uri);

        let mut self_addr = Multiaddr::empty();
        let t = match config.libp2p_transport {
            Libp2pTransport::Nym => {
                build_nym_transport(local_keypair.clone(), uri, &mut self_addr).await
            }
            Libp2pTransport::NymEitherTcp => {
                build_transport(local_keypair.clone(), uri, &mut self_addr).await
            }
            _ => unreachable!("checked before"),
        };

        address = Some(self_addr);
        t
    }
}
```

`Transport` *selection (source)*

Integration Components:

Nym Addresses Mapping

- A mapping between *peer_id* and nym addresses is added in places where Lighthouse expects TCP addresses.

```rust
// Handles the libp2p request to obtain multiaddrs for peer_id's in order to dial them.
fn addresses_of_peer(&mut self, peer_id: &PeerId) -> Vec<Multiaddr> {
    let trust_peers = self.network_globals.trust_peers();
    let Some(addr) = trust_peers.get(peer_id) else {
        return Vec::new();
    };

    vec![addr.clone()]

}
```

`peer_id` mapping to nym Multiaddrs (source)

Integration Components:

Caveats - Discovery and PeerManager

- Discovery and PeerManager components are disabled for the purposes of this PoC.
- DHT-Based peer discovery is disabled because integration with Kademlia is outside the scope of the SoW.
- Nym-enabled peers need to be added manually using `set_trust_peers`.
- peer manager generates large additional traffic that overwhelms `nym-client` causing throttling and dropped messages. Some of these messages are related to block gossiping which disturbs consensus. For the purposes of PoC we decided to disable PeerManager component.
- We're leaving Nym integration with PeerManager and Discovery for future work.
    - One potential direction is to use separate transports, i.e. use Nym for GossipSub and TCP for other components. This is currently not supported by Libp2p (uses one transport for everything, circuit relay doesn't allow developer to define rules for which transport to use, choice happens based on transport availability based on priority).
    - Dual transport might ruin all the benefits of privacy enabled by Nym, *e.g.* de-anonymize node based on peer manager activity and associate with associate it with validator identity that was hidden using Nym.
- Source Link

Integration Components:

Caveats - Peer Reporting and Banning; Timeout Tweaks

- Peer reporting and banning is disabled:
  - Due to increased latency and intermittently dropped connections, the scores of nodes using Nym transport are being reduced;
  - The low peer score prevents peers from being added in GossipSub meshes which prevents blocks from being disseminated in the network;
  - This is also out of the scope of a PoC.
  - Diff: Source
- Time-related configs are tweaked to account for mixnet latencies
  - Diff: Source.

Metrics

- The following New Metrics are added:
  - beacon_block_published
  - Beacon_block_received
  - Source
- Grafana Dashboard Config: Source.
- Run `docker compose up --build` from the root of the lighthouse-metrics directory.
- Prometheus: http://localhost:9090/targets
- Grafana: http://localhost:3000/ (*username: admin, password: changeme)*

Video for running Metrics Dashboards

Simulation

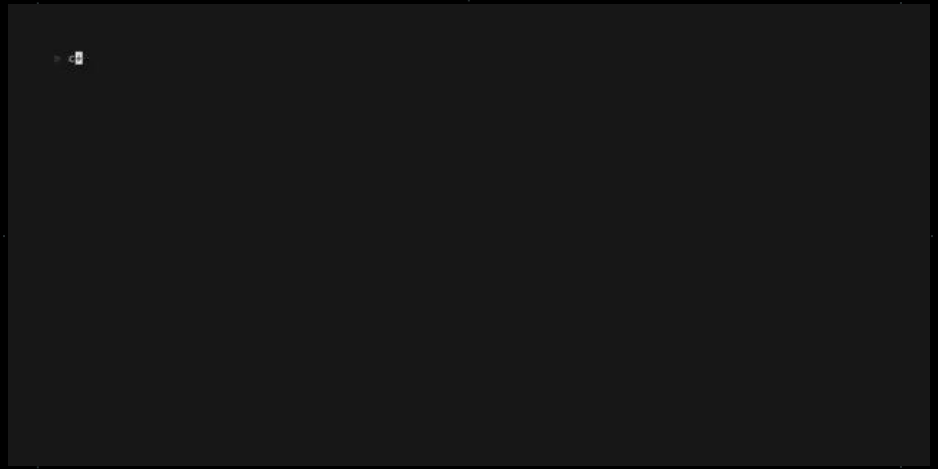
- Here, we primarily target small-scale local testnet simulations. The performance is measured by the number of successfully synced blocks, i.e. blocks that were successfully *gossipped* to all participants in the local network.

- The network here is local but the transport uses the mainnet Nym mixnet.

- Documentation to run simulation and to use the validator: Link.

Simulation

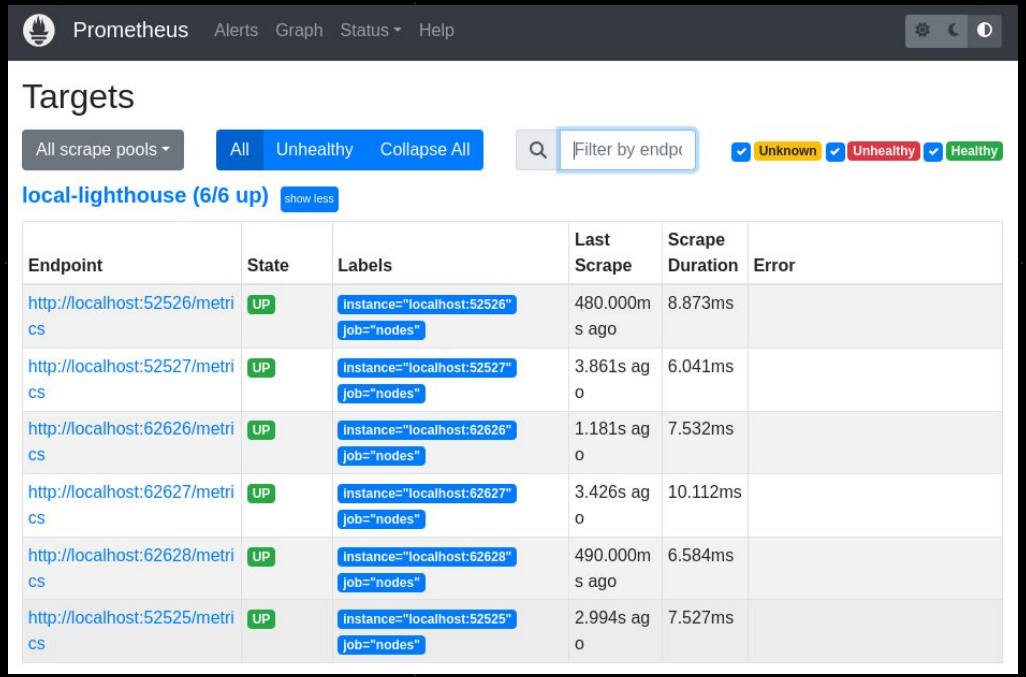
Running a Simulation with TCP Transport



```
cargo run -p simulator --release -- eth1-sim -t
tcp --nodes 3 --validators_per_node 1
```

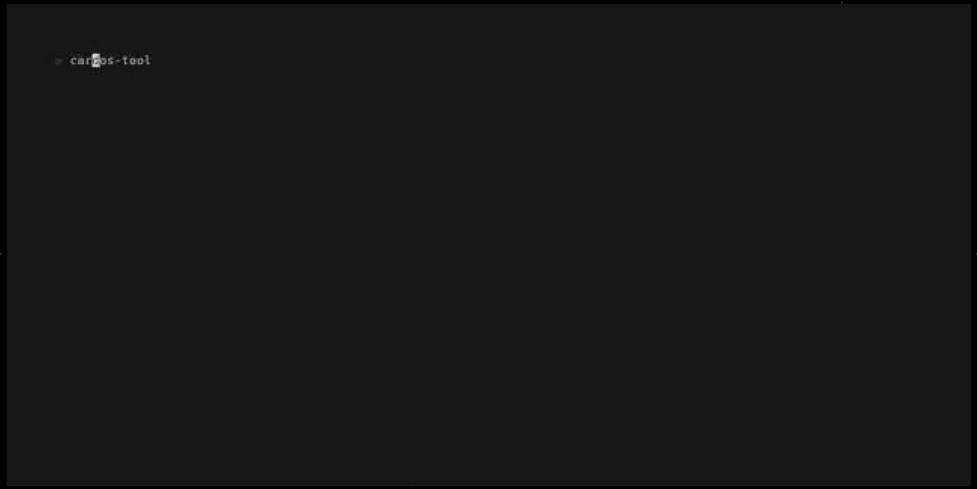
Simulation

Running a Simulation with TCP Transport - Prometheus Metrics



Simulation

Running a Simulation with Nym Transport

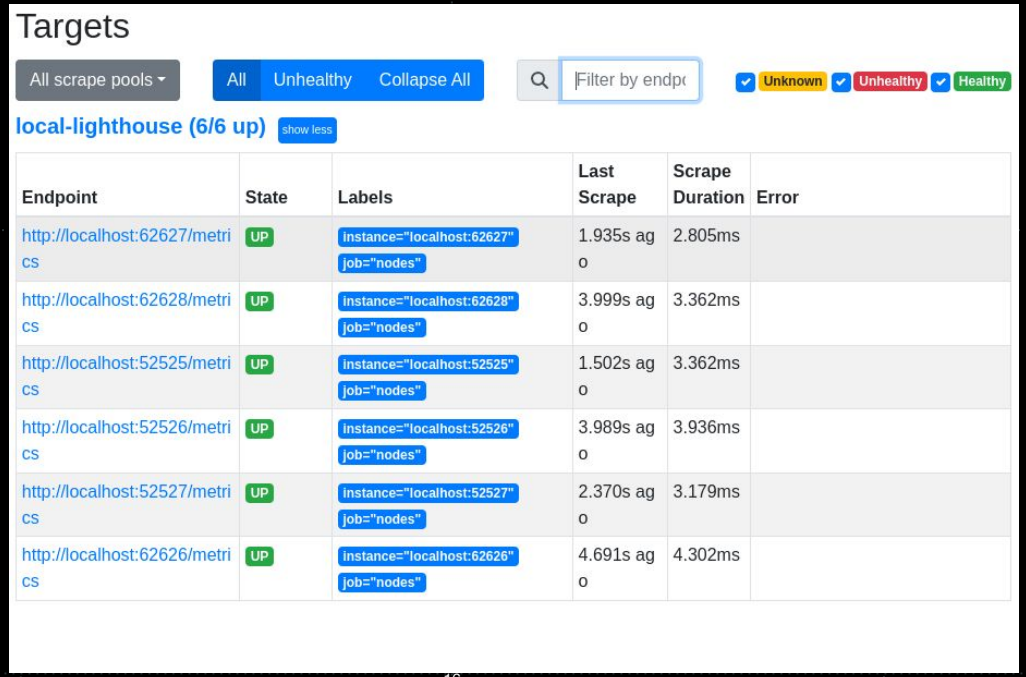



```
cargo run -p simulator --release -- eth1-sim -t
nym --nodes 3 --validators_per_node 1
```

Note: Ensure you have `nym-client` in your PATH when running this.

Simulation

Running a Simulation with Nym Transport - Prometheus Metrics



Resources

- Validator Privacy with Nym (Documentation)
- ChainSafe/lighthouse
- ChainSafe/rust-libp2p-nym

Thank you.