eRPC - An Efficient Relay Partition Checker

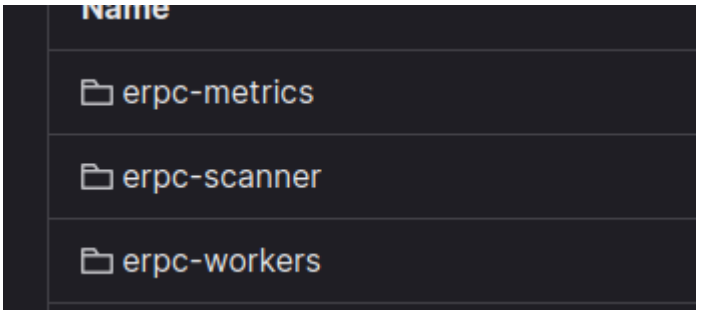
Introduction + 3 Week Report(May 29 - June 18)

Project Link : https://gitlab.torproject.org/rishadbaniya/erpc

eRPC is a flexible relay partition checking tool, flexible in a sense that the configuration that it aims to provide.

Let me explain the project structure briefly.

I've divided the project into two libraries **erpc-metrics** and **erpc-scanner** to be used within the application binaries within **erpc-workers**



erpc-metrics(To be worked on) : This module is all about using collecting the data of circuits that were created by OnionPerf as part of saying the **circuit was successful** or **failed**

erpc-scanner(Almost Complete) : This module handles the scanning part, it has this pool of **struct Client** that hold **CircMgr** of a **TorClient** with themself and they attempt to create Circuit through the **CircuitBuilder,** it only exposes two interfaces, a sender half of the channel to send

IncompleteWork and receiving half to receive **CompletedWork.**

Within the application there's this concept of **IncompleteWork**, which is basically a Type that holds metadata about the Relays we want to make two hop circuit with(which is just the fingerprint of the relay) and other is **CompletedWork**, which is the result of the attempt to create circuits, it stores the information about the circuit creation attempt, later on this is used.

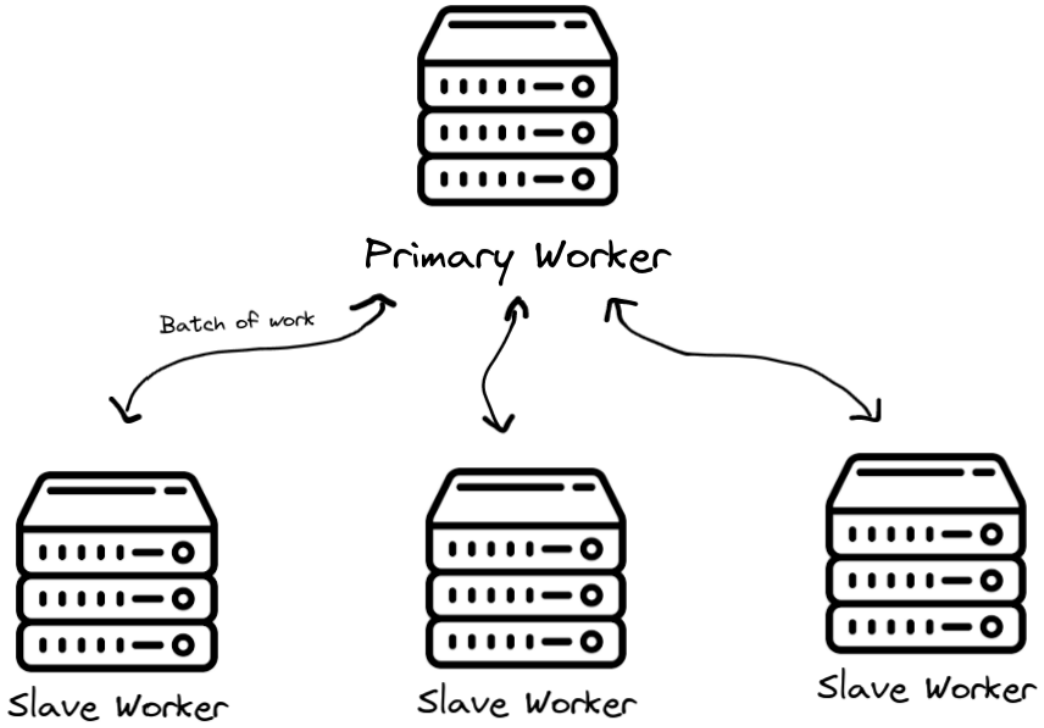
```rust
#[derive(Debug, Clone)]
pub struct IncompleteWork {
    /// Metadata to identify the source relay
    pub source_relay: String,

    /// Metadata to identify the destination relay
    pub destination_relay: String,
}
```

```rust
/// The definition of CompletedWork is the result of attempting to
/// and the Destination Relay
#[derive(Debug, Clone)]
pub struct CompletedWork {
    /// Fingerprint to identify the source relay
    pub source_relay: String,

    /// Fingerprint to identify the destination relay
    pub destination_relay: String,

    /// Shows how the work was completed, was it a success or failu
    pub status: CompletedWorkStatus,

    /// The Unix epoch at which the test was performed
    pub timestamp: u64,
}
```

This information is later on used to make rpc calls, also to store within the Graph Database **Neo4j**.

Btw, I plan to use a graph data structure and graph database within the application because my mentor mentioned that it seems to be perfect to express TorNetwork as a directed graph with these Relays as Nodes and the attempt to create circuits between these relays as the edge.

erpc-workers(Heavily working on right now): This module produces two binary, one is the *slave_worker* binary and other is *master_worker* binary. This is the very high level overview of how the application is going to work.

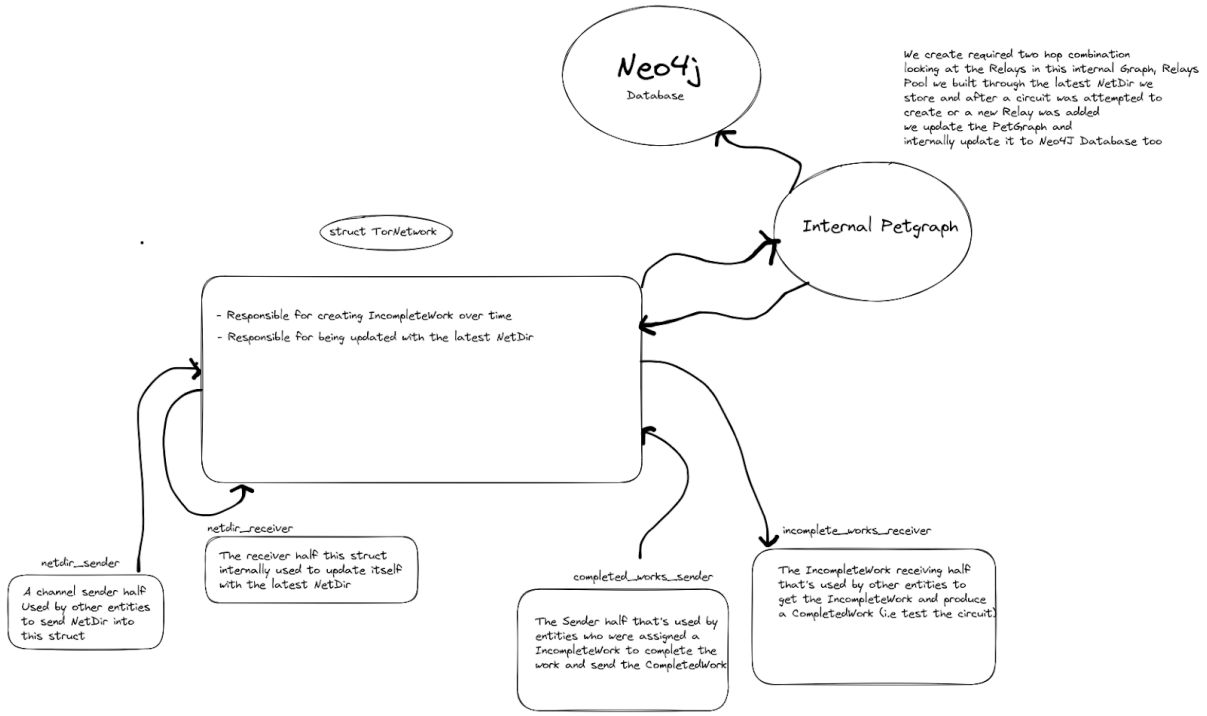


This application follows the Master - Slave architecture in a sense these slaves are pluggable, which means we can disconnect these slaves any time and the **pace at which the circuit testing** happens won't be hindered, because it

automatically adapts to the change and Master Worker **orders** these Slave workers to start testing circuits at a certain pace, also there won't be data loss in a sense that the assigned work is buffered in the MasterWorker until the work is completed.

Btw, i've used this concept of batch of work, it's like a Vec<IncompleteWork> that the Primary Worker sends to the Slave workers when they request for work. Because there would be million of circuits to be created and making a single RPC call to get them by the SlaveWorkers would be an overhead in terms of latency, so i've come up with the idea of assigning a certain batch of work of X no of these **IncompleteWork** and if the work is considered failed i.e by the SlaveWorker being disconnected something like that then we take that data that we had stored (the one assigned to that SlaveWorker) and put it back into the pool of **IncompleteWork**, this way i believe i can make the Slave Workers pluggable.

The Master Worker that i'm currently building, i'm planning to have two kinds of gRPC server, one is for distributing these **IncompleteWork** to the Slave Workers and getting back **CompletedWork** from them and other is to control the application behavior during the runtime, such as pausing, stopping, resuming changing batch size, no of allowed slave workers, custom relays to be excluded or included(this gRPC server to control the application is left for the last part)



Here's one of the core data structures(TorNetwork) within **erpc-workers** and its high level architecture that I had made to visualise what was going on.

It would be great to hear some feedbacks from you regarding **how you expect the partition checking tool to behave**, i mean how the scanning should **start** to take place, is it the relays with highest level of consensus to be checked first(because they have the highest chance to be selected in a **real** Tor Circuit) or what else should be my approach on **starting** a scan, **continuing** the scan and what level of configuration should this application provide so that the scanning can be tweaked **according to our goal** for the scan (i.e for checking censorship, failing relays etc)