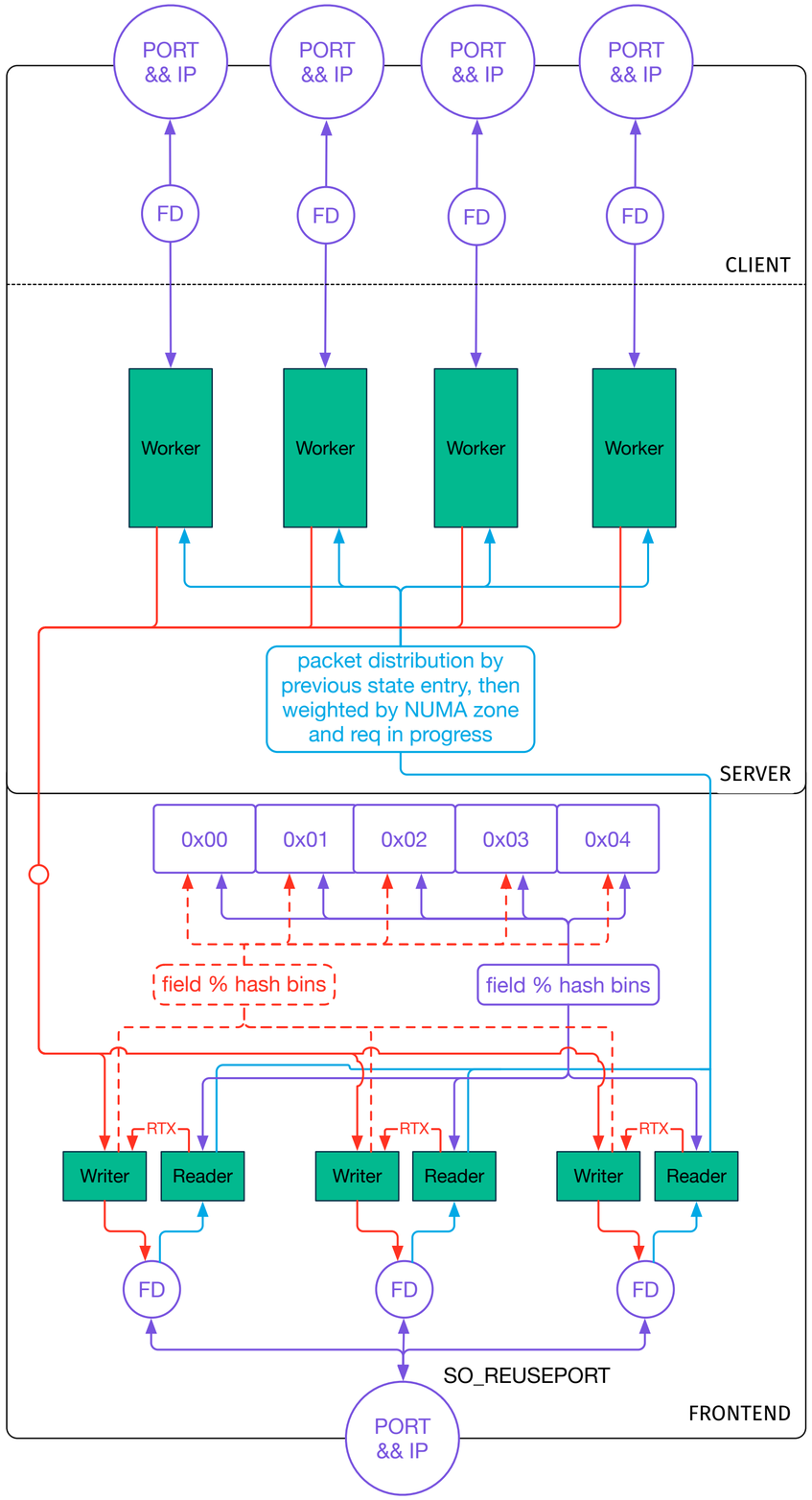

Workers share nothing, and have their own outbound connections for proxying, and handles for databases, APIs etc…

Number of workers is fixed. And defaults to number of cores.

Packet encode/decode handled by backend worker threads.

This hash function should ensure packets in same (EAP) conversation go to same worker thread.

Packet reading/writing handled by frontend threads.

State matching done by frontend threads (cut and forward style).

Use of separate reader threads allows migration path to split frontend and backend .

RTX handled by writer, as its not guaranteed unsynchronised writes will be atomic.

Use of separate reader threads allows migration path to split frontend and backend .

Dedup-trees should be part of frontend so they survive restarts.

*Workers *could* update dedup trees themselves, but having dedicated writer threads means we can coalesce updates to dedup trees and use sendmmsg().*

These operations would get more efficient as load increased. Unit of work would be outstanding requests on writer pipe at time of reading.

Would have interesting throttling effect due to increased contention with readers.

Protocols should export worker_send function, and an opaque context ptr the worker should pass to that function. Workers should have no knowledge of how packets are written.

In some instances writer threads and pipes may not be necessary, and worker_send function can write directly to appropriate FD.

Workers write back to socket/queue/pipe for the writer paired with the reader that originally read the packet. This should ensure even loading (if kernel distributes reads evenly).