Protocol versioning

We need protocol version negotiation when two nodes establish connection:

the connecting node advertises a range of supported protocol versions in the connection request,

while the responder either accepts the connection indicating the highest mutually supported protocol version, or rejects it.

Nodes need to support multiple protocol versions because we can't upgrade the whole network at once, and to be able to reason about correctness of the protocol, we should ensure that nodes speak the exact same protocol version when they communicate, and in the implementation make sure they cannot fallback to other versions after agreeing on one.

Protocol versioning fits into our type system neatly, we could do it as follows:

--- Ticker Engine v1 type TickerMsg1 := | Increment1 | Count1 ;

--- Ticker Engine v2 type TickerMsg2 := | Increment2 | Decrement2 ;

--- Protocol v1 type Msg1 := | MsgNetwork1 NetworkMsg1 | MsgTicker1 TickerMsg1 ;

--- Protocol v2 type Msg2 := | MsgNetwork1 NetworkMsg1 | MsgTicker2 TickerMsg2 ;

--- Supported protocols in the current specs release type Msg := | Msgv1 Msg1 | Msgv2 Msg2 ;

Msg

comprises all supported protocol versions by the current specs release, which gets incremented at every release.

Individual protocol versions (Msg1

, Msg2

) contain versioned engine messages in that specific protocol version.

Engines have versioned messages as well.

When changing/adding/removing any message to an engine between specs releases,

we need to increment both the message version and the engine version.

We should describe each engine version in separate files,

e.g. ticker_messages_v1.juvix.md

defines TickerMsg1

,

and ticker_messages_v2.juvix.md

defines TickerMsg2

.

When incrementing an engine version we can thus copy the latest version to a new file,

and increment all Message versions as well, even if they didn't change.

This will make it easier to remove old protocol versions just by deleting the corresponding file.

After a while we can remove old protocol versions that are not supported anymore

(e.g. when adding Msg3

we could remove Msg1

, in practice we should allow a reasonable time period for upgrades, a few weeks should be sufficient, similar to what Signal does)

Later we should also have some in-band software update protocol with signed releases, for this we could use a a more stable protocol that doesn't change that often, or something like HTTP, such that long-offline nodes can also upgrade when they return.

Protocol vs specs vs software vs product versioning

It’d make sense to keep the protocol and specs versions in sync, since these are closely coupled. Software and product versions should specify which specs version they use.