Notes on our discussion on networking and ids from yesterday. @cwgoes @isheff @tg-x @graphomath CC @nzarin

[

IMG_0321.HEIC

4032×3024 2.66 MB

](https://europe1.discourse-cdn.com/standard20/uploads/anoma1/original/1X/29689ae03f7105bd64b31c51cd2130c2cd260fae.jpeg)

My original proposal

Let's get rid of engine ids because:

- I think an external id is a trust domain, but different engines within the same node are not in different trust domains—a single node is a trust domain. Only a node should have an external id, and engines should be identified some other way
- The current way is operationally annoying, not just conceptually annoying, because we have to reason about the trust associated with the identities to which we send and from which we receive messages
- The current way is operationally annoying, not just conceptually annoying, because we have to reason about the trust associated with the identities to which we send and from which we receive messages
- Currently, we sometimes send to topics and sometimes to engines; let's make a uniform 'topic' abstraction and say that every message is sent to a topic, so that way we don't have to reason about multiple different types of destinations

Currently, a topic contains one identity, which identifies who is allowed to send to it. TG agreed with me at the last hacker house that a topic needs to additionally include a uuid, in order to enable having multiple different topics with the same set of allowed senders. ('UUID' means 'universally unique identifier'; generally, a 128-bit or 256-bit cryptographically random number.) My proposal: let's add one more field, which is an external id specifying allowed recipients, i.e.:

struct topic: allowed-senders: external-id allowed-receivers: external-id uuid: uuid

(Messages sent to the topic would, generally speaking, be encrypted so only the allowed-receivers can decrypt them.) Under this new system, an 'engine' would be identified by a topic where:

- allowed-senders is the 'all' identity
- allowed-receivers is the id of the node where the engine lives
- the uuid would be some uuid specific to that engine (to distinguish it from other engines on the same node)

A pubsub topic would be identified by a topic where:

- allowed-senders is some identity (we already have this for pubsub topics)
- allowed-receivers is the 'all' identity
- the uuid is some uuid specific to the pubsub topic

And we can of course imagine pubsub topics where allowed-receivers is somewhat restricted—this seems like a useful thing to be able to do.

Another somewhat convenient thing this enables (but does not enforce) is standardised uuids. Suppose we agree that f51f2ae7ff1816e4 is a standard uuid for mempools—i.e., somebody will decide on a specification for the behaviour of mempools, and generate the uuid f51f2ae7ff1816e4, with the convention that mempools will use the uuid f51f2ae7ff1816e4. Then, if I want to talk to a given node's mempool, it suffices for me to know that it has a mempool, and then I can just send a message to topic with uuid f51f2ae7ff1816e4 at that node. Rather than having to learn, not only that it has

a mempool, but also the specific id of its mempool.

Subsequent discussion

Went on for a long time—I will write everything I remember but please weigh in if you said something and I forgot. Or if you have new or different opinions or arguments, of course. Also not entirely chronological, but a mix of chronological, topological, and increasing complexity. Also prioritising saying things over making them consistent.

TG: engine message/topic message was supposed to be unicast/multicast and is operationally different.

Elijah: but unicast is a special case of multicast where there happens to just be one recipient.

Elijah: multicast involves complicated routing, but unicast also involves complicated routing, and we shouldn't necessarily expect that the latter will always be uniformly simpler than the former (even if it tends to be in general). Further, we can have a special case in the routing logic for the case where a topic's allowed recipients are just a single node (and implicitly assume that such a topic is always subscribed

TG: shows routing table (top right whiteboard) where a given id can map to a topic or an engine

Chris: in that case these are already being somewhat conflated so I lean somewhat towards agreeing that a unified abstraction makes sense

Elijah: (top right whiteboard topic|engine

) that is—we currently have routing table entries which are a topic or an engine, so just let them always be a topic

TG: if a message dest. includes a node id then we do not need engine adverts which is another operational advantage

TG: but it's kind of annoying that now that I cannot just send a message to an id (I don't understand this complaint but I remember there was something like it)

More interesting further discussion

We tentatively agree that some degree of change and likely unification is requisite. But we're discovering ontological problems that need to be resolved? What are 'node' and 'id'? What concepts do we have (do we need more concepts than just those?), and what words should we use for them?

Node

We might want to have multiple different computers with a single logical thing spread over them (for ex. single logical mempool, mempool workers on different computers). Should each computer have a distinct 'node id' or no? Probably not. We can have an id node1 OR node2 OR node3 for contexts where we want to treat the group as a single trust domain. We want to have some notion of a 'routing domain', that is, something to which messages can be sent over the network (a single computer). We tentatively agreed that a node is a routing domain. (Next-day discussion—perhaps Isaac disagrees.)

Routing

Elijah: I think 'id' means 'trust domain'.

Isaac: I think 'id' means 'something to which you can send messages'. (Writes 'an X is …' on the bottom of the whiteboard, then says 'I posit that an X is an id')

Elijah: but a topic is not an id and we can send messages to it too

Isaac: a topic is an id (i.e. message target) plus

some access controls

(I don't understand why we can't use a more uniform general mechanism where all message targets have associated access controls, and we sometimes have the degenerate case of no restrictions on access.)

(I didn't say this at the time, but: if an id just denotes something to which I can send messages, then I want something that's just a trust domain? Further discussion with Isaac today: maybe there shouldn't be something that's just a trust domain. Routing to compositional identities is confusing. Maybe some

identities are message targets and some are something else that we don't know what they are yet. Also maybe 'node'≠'routing domain'—I forget why.)

Isaac: we can do your thing where engine has the same cryptographic key as the node it lives on while also saying you always send to an id. Add a special compositional id id-with-uuid(id, uuid). (Or go for: pubkey-with-uuid(ed25519-pubkey, uuid), or whatever.)

Elijah: I guess it works but it's aesthetically displeasing and implications for signs-same-as/reads-same-as is weird. (I guess if 'id' means 'message target', then signs-same-as/reads-same-as is not equivalent to is-same-message-target.)

Elijah: a message should have a destination id and a delivery hint? (For example, send to the 'all' id and have a hint 'only to people who've said they want to subscribe to uuid blablabla.)

Isaac: meh. maybe

Chris: node is a routing domain

Elijah: what is that—how identified, concretely?

Chris: idk

Elijah: uuid? cryptographic key?

Chris: routing and trust domains shouldn't be conflated—should be orthogonal. So could be uuid, sure

Elijah: is a routing domain uniquely associated with at least one trust domain (and possibly non-uniquely associated with more trust domains)?

Chris: shouldn't necessarily because orthogonal

Elijah: but a routing domain may have trust associated with it—connection is to a routing domain and has associated authentication (tg agrees—isaac still wants to only send to ids)

Chris: maybe somewhat convinced

We want some notion of a 'user'. Somebody has 5 devices. Something differentiates them but there should be something they have in common. (Is a 'device' or 'node' always associated with exactly one user?)

Metacomments: i think it's somewhat

clear what the ontological problems are but not entirely clear the solution. This forum post is not very clear.