@mariari and @ray brought up the question of how applications running in the resource machine should interface with other engines running locally on the node, such as engines within the identity machine, the dynamic configuration engine, and engines within the networking machine. I think, in general, that we want to expose as much interface surface as possible to applications running within the resource machine, within the constraints of:

- maintaining consensus (so no calls outside the RM can return different results on different nodes running the execution engine for the same controller)
- safe execution (since transactions are arbitrary user input), and appropriate resource pricing of any computational/storage resources expended in the course of transaction processing
- transaction functions themselves being pure functions returning transactions and any I/O operations necessary (as opposed to I/O operations happening during transaction function execution)

Identity machine interface

The identity machine provides several functions which may be useful to transactions:

- verifying commitments
- creating commitments (in particular, commitments which are implemented through a executor quorum signature over a particular piece of state having a particular value)
- encrypting & decrypting data (really only useful if the executors otherwise keep execution data private by default)
- checking signs-for, reads-for, and naming relationships (with these ones, we need to be careful to segregate state known to the local node and state known to all nodes in a particular controller)

I think some ways in which these interfaces can be connected are:

- Stateless functions - e.g. commitment verification - can be included as precompiles / standard library jets. We need to either prohibit the usage of local signs-for state in this verification, or (better) segregate signs-for relationships on the basis of which are known to which controller, and only allow for the usage of relationships which are known by the controller which is executing the transaction.
- Commitments can be requested via specially designated resources, which will have permissions governing what kinds of commitments can be requested and what must be paid.
- Encrypting & decrypting data can also be included as precompiles, although here we will need to think about the information flow control reasoning.

Networking machine interface

In order to maintain deterministic execution, the resource machine cannot read any inputs from the networking machine. It can, however, perform writes, which will primarily take the form of sending a message to another node or topic. It could also - in principal - instruct the operating node to join or leave a domain, change routing or peer state, or perform other operations. This topic is also discussed here.

I think that all of these operations would take the form of instructions returned by the transaction function - technically as part of the transaction

data structure, although we may want to split out these kinds of I/O operations for clarity. Different I/O operations will need to be governed with different permissions; for example:

- Payment for message sends (this is just bandwidth)
- Signature checks for changes to routing or peer state (e.g. a peer would need to sign over a change to that peer's stored routing preferences - such preferences can also be relayed over the networking machine without consensus, so we should figure out whether this is actually useful).
- Payment for joining a domain (and perhaps some restrictions on how much bandwidth will be consumed in the future) - these sorts of (implicit) commitments to future networking behaviour are probably useful, since we can incorporate e.g. explicit payment in the resource machine, but require some further thought and design to implement safely and reliably.

Advanced topics

OTA software updates

The system could read from a specially designated resource an actual code hash

(e.g. for the project Git repository), and when that resource is changed, automatically shut down the system, check out the

code hash, recompile, and reboot. Obviously, this resource would need to be governed by very restrictive permissioning logic, such as a signature from a quorum of trusted parties, and OTA updates will likely be more palatable if the node itself is relatively sandboxed. Alternative versions of this concept could include hot-reloading specific actors (@mariari is this easily possible?), type-checking the new code before relaunching, or other kinds of partial updating & verification.