A “chain-cluster

” is a set of L2 or L3 chains that allows communication between chains in the cluster, trustlessly, faster than the settlement times of the individual chains. This fast communication would include the possibility of cross-chain transfers of ETH and other assets.

We will require a synchronous guarantee for cross-chain communication, that is, that any cross-chain message within the cluster that is sent by one chain at time T will be visible on its destination chain by time T+Delta

at the latest, where Delta

is less than the settlement time of the sending chain.

There are two use cases for this:

- a single chain can take a sharding approach, expanding into a set of clustered chains that have identical code and configuration, and are governed as the original single chain was; or
- a set of separately governed chains can agree to connect their chains into a cluster, accepting a level of interdependence in their security in exchange for much faster cross-chain interaction.

Although these use cases are different, the technical considerations are essentially the same.

Node-clusters

If Chain A and Chain B are in the same cluster, and A can send faster-than-settlement messages to B, then any B-node must have a trusted source of information about what happened on A, and this source cannot rely on A’s settlement mechanism because it needs to happen faster than A’s settlement.

The obvious way for a B-node to know what happened on Chain A is for the party running the B-node to also run an A-node. Then the A-node can provide information about events on A, and the B-node can trust that information, because the two nodes are operated by the same party.

Applying this argument to all pairs of chains in the cluster, we can see that a party that wants to run a node for any chain in the cluster must run a node for all chains in the cluster. Let’s call such a set of nodes, one for each chain in the cluster, a node-cluster

.

Joint settlement

The chains in the cluster will need to have some kind of joint settlement. If settlement is by ZK-proof, we would want a single proving framework to cover all of the chains in the cluster, because events on one chain can affect events on another chain over time-frames shorter than settlement time. If settlement uses optimistic proving, we would want a single assertion and dispute protocol.

In principle it is possible for each chain to use separate proofs, with the proofs being contingent on events having occurred on other chains, that is, an assertion and its proof are deemed to be null and void if the events they assumed on other chains did not occur. For this to be workable, each prover would need to have reliable information about what happened on the other chains, in order to avoid wasting time (and potentially endangering liveness) by doing irrelevant proofs. Additionally, the on-chain proof verification mechanism (on the parent L1 chain) would need to be able to verify whether the assumed preconditions for a proof (events on other chains) actually did occur. So in any case, proving would need to be organized as a joint endeavor across the chains in the cluster.

An obvious way to do this is to give the prover access to a node-cluster so it knows what it needs to prove, and then have the proof establish the correct results across the entire chain-cluster.

Security dependencies

A failure of liveness on one chain in the cluster will cause failure of liveness on all chains in the cluster. If Chain A does not make progress after time T

, then other chains in the cluster cannot make progress beyond time T+Delta

because they cannot know whether chain A would have sent them a message at time T+1

, so they cannot know the set of messages they need to see at time T+1+Delta

.

Similarly, a failure of safety on one chain at time T

will invalidate assumptions of safety on other chains after time T+Delta

. If Chain A violates safety, then any cross-chain messages from Chain A—including asset transfers—cannot be assumed safe. The result is that other chains in the cluster cannot guarantee safety.

Security in a chain-cluster is a weakest-link phenomenon: a failure of any one chain in the cluster will propagate to all of the others.

Joint governance

It follows that any chain-cluster will need to have some kind of joint governance structure or policy. This governance would have at least two purposes.

- Governance would decide which chains can join the cluster. When a new chain joins the cluster, this imposes an obligation on every node that serves the cluster: to run a node for the newly joining chain. Because of this obligation, membership in a cluster cannot be permissionless, and many clusters will want to limit carefully which chains can join.
- Governance would approve any upgrades or security-critical configuration changes to any of the member chains. Because a failure of liveness or safety on any member chain will affect every chain in the cluster, the cluster must defend itself against upgrades or other changes that might endanger security of any member chain. This does not require that every chain in the cluster use the same code or configuration; but it does require that every chain use code and configuration that the others are willing to trust.

Some clusters might handle these issues through a set of fixed rules, such as having a fixed set of members, identical code and configuration, and always upgrading when some public reference chain does so. But many clusters will want to have more flexibility than this, or more diversity, which would require governance.

If the chains in the cluster affiliate with each other voluntarily, they can design whatever joint governance structure they want. There is no need to impose any particular structure on them, nor is there any need to limit the numbers or types of clusters that exist.

We’ll be writing more about design approaches for chain-clusters, soon.