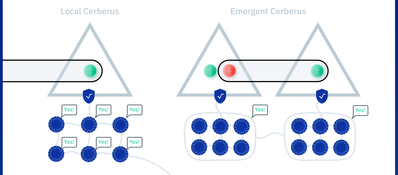
RADIX

Cerberus - How Radix achieves infinite linear scalability while preserving atomic composability

Consensus - Local Cerberus

So now on to Cerberus. As previously mentioned, Cerberus' method of reaching consensus is in two tightly integrated parts:



Local Cerberus, which is how nodes come to consensus with each other, for a single shard, within a validator set.

Emergent Cerberus, which is how nodes come to consensus across shards, between validator sets.

We start with local Cerberus.

This is the same transaction as in episode 9.

Transaction (Tx)

Shard 1:	*Shut down Alice's Token A substate*
Shard 2:	*Bring up Bob's Token A substate [Index: 0]*
Shard 3:	*Shut down Bob's Token B substate*
Shard 4:	*Bring up Carol's Token B substate [Index: 1]*
Shard 5:	*Shut down Carol's Token C substate*
Shard 6:	*Bring up Alice's Token C substate [Index: 2]*
Signed:	*Alice, Bob, Carol*

Tx ID: ...97f7

So as before, a client submits a transaction to the network.

Once a node picks up the request, it forwards the transaction to the "leader" of each relevant validator set.

We can see that as this transaction covers six shards, the node that picks up the transaction forwards the transaction to six different leaders - one for each validator set serving each shard.

As we are covering local Cerberus here only, let's see what happens just for:

"Shard 1: Shut down Alice's Token A substate".

Proposal Vertex

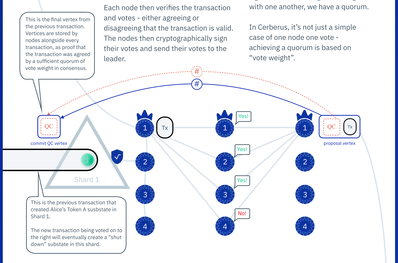
When the leader of the validator set serving Shard 1 receives the transaction, it first verifies that the transaction is valid. Once verified, it broadcasts the transaction to all the other nodes in the validator set. As this is the first message, this is the "proposal".

Each node then verifies the transaction and votes - either agreeing or disagreeing that the transaction is valid. The nodes then cryptographically sign their votes and send their votes to the leader.

As the nodes in this example only have visibility of Shard 1, when they sign, they're only signing that they're happy with the transaction as it relates to Shard 1.

The leader then collects all the signed votes. If enough nodes agree with one another, we have a quorum.

In Cerberus, it's not just a simple case of one node one vote - achieving a quorum is based on "vote weights".



Cerberus weights votes made by nodes by how many tokens are "staked" to them. We cover how this works in episode 15 - Sybil Resistance Through Proof of Stake. For now we'll just assume one node one vote.

In Cerberus, you need "2f+1" of vote weight, or just over 2/3rds, to achieve a quorum. "f" means the maximum tolerated number of "faulty" nodes, so a threshold of 2f+1 means that just over 2/3rds of vote weight has to be non-faulty, or trustworthy.

If a quorum is achieved, the leader takes all the signed votes, and issues what's called a Quorum Certificate (QC). The QC provides cryptographic proof that a 2f+1 majority of vote weight in the validator set approved the transaction.

But the leader doesn't just create a QC on its own - it creates a data structure called a vertex.

The vertex contains:

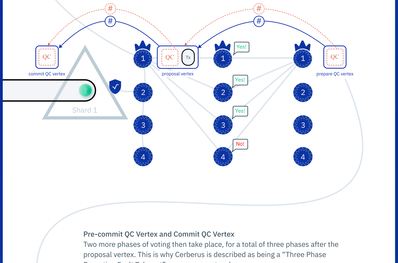
- the transaction
- the QC proving the transaction was agreed by 2f+1 of vote weight
- hashes tying both the QC and the proposal vertex back to the final vertex from the transaction that created the input substate.

As this vertex contains proof that the nodes agreed on the proposal, it's called the "proposal vertex".

Prepare QC Vertex

We then move on to the next phase, where the process repeats. The leader now broadcasts the proposal vertex to relevant nodes, and the nodes then vote on whether the proposal vertex is valid.

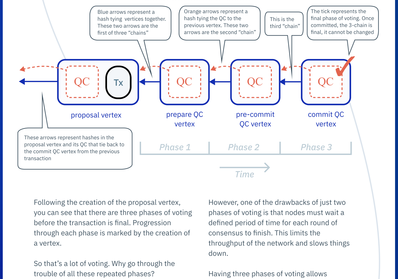
If 2f+1 of vote weight agrees with the proposal vertex, then the leader packages up proof of this in another QC, in another vertex, with hashes back to the proposal vertex. This next vertex is called the "Prepare QC Vertex".



Pre-commit QC Vertex and Commit QC Vertex

Two more phases of voting then take place, for a total of three phases after the proposal vertex. This is why Cerberus is described as being a "Three Phase Byzantine Fault Tolerant" consensus protocol.

If we zoom out on all these rounds of voting, the process looks like this:



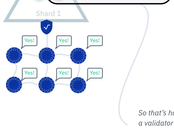
Following the creation of the proposal vertex, you can see that there are three phases of voting before the transaction is final. Progression through each phase is marked by the creation of a vertex.

So that's a lot of voting. Why go through the trouble of all these repeated phases?

The answer is that two phases of voting is the minimum that's required to allow nodes sufficient opportunity to be sure that they are all in agreement, but still allow them to safely fail the transaction at any phase if a quorum of vote weight cannot be achieved. Two phases of voting is common in other BFT-style consensus protocols such as "Tendermint".

However, one of the drawbacks of just two phases of voting is that nodes must wait a defined period of time for each round of consensus to finish. This limits the throughput of the network and slows things down.

Having three phases of voting allows consensus to proceed "optimistically". This means that consensus rounds can be completed as fast as the nodes can exchange enough votes – there is no longer a need for a fixed waiting period. This is the key innovation of a BFT-style consensus protocol called "HotStuff" that local Cerberus draws heavily on.



To close the loop, we can see that as the nodes in the validator set serving Shard 1 have come to consensus - evidence of which is contained in the commit QC vertex - they are now able to create the shut down substate in Shard 1 as per the transaction.

So that's how local Cerberus allows nodes within a validator set to come to consensus.

But local Cerberus on its own wouldn't allow a single transaction to complete, as all transactions are cross-shard.

To understand how cross-shard transactions work, we now cover emergent Cerberus.

The version of Cerberus described in this infographic series is scheduled to launch as part of the fully sharded Radix Xi'an release. Please visit www.radixdlt.com for details on the Radix roadmap.

© Copyright Radix Tokens (Jersey) Limited