Each client maintains a “suspicion score” for each chain C

, which works as follows:

score = \underset{v \in votes}{max} \left(TimeSeenIncluded(v, C) - TimeSeen(v) - \frac{now - TimeSeenIncluded(v, C)}{16} \right)

Where votes

is the set of votes that were valid to be included in C

at the time they were submitted, now

is the current timestamp, TimeSeen(v)

is the time that the node first saw that vote, and TimeSeenIncluded(v, C)

has a three-part conditional definition:

- if v \in C

(ie. v

has been included in C

), then it’s the time that the block in C

that included v

was received

- if v \not\in C

but including v

would no longer be valid (eg. because an epoch timed out), then it’s the time that the last block in C

in which including v

is still valid was received

- otherwise, it’s now

Basically, it’s the longest amount of time that the client has seen that a vote could have been included in the chain but was not included, with a “forgiveness factor” where old events are discounted by how old they are divided by 16.

It’s fairly easy to show that if there is a latency bound \delta

between two clients, then the two clients’ suspicion scores will differ by at most 2 * \delta

. It’s also easy to show that a chain cannot censor votes without racking up a suspicion score among all clients.

We can also add two modifications to the Casper FFG protocol:

1. If an epoch is not finalized, the next epoch is 25% longer. The lengths reset only if an epoch does get finalized.
2. A vote is allowed to include one transaction (in a stateless client model, it would need to have witnesses based on the state root of the block the vote is voting for). If a block includes the vote, it must process that transaction.

(1) does the double-duty of both (i) making Casper FFG a properly partially synchronous consensus algo, able to provably eventually get consensus with any

latency bound \delta

, and (ii) ensuring that if a chain continues censoring for a long time, it’s suspicion score will increase unboundedly. (2) ensures that full transaction censorship cannot happen without vote censorship.

The purpose of this scheme is to serve as a social coordination tool for users to reject censoring chains. Users are guaranteed to have suspicion scores that are reasonably similar, and this will make it easier to coordinate on whether or not to perform a minority fork to escape a censorship attack at any given time.

Addendum (2017.12.30):

In the current FFG spec, only ~20 minutes of latency is sufficient to make Casper never include certain blocks, transactions

or votes, so this suspicion score would not work very well in that context. To solve this problem, we can add a rule that if an epoch does not finalize, the next epoch's length doubles, and epoch lengths continue doubling until an epoch does finalize. If EpochLength = 50 * 2^{EpochsSinceFinality}

, or any other formula, then we'd maintain the quadratic leak by simply setting LeakThisEpoch = k * EpochLength * BlocksSinceFinality

; in this context, BlocksSinceFinality = EpochLength - 1

, but in other contexts it could be different. This way, you would actually need to accrue a high suspicion score to fully censor votes.

This also makes Casper FFG secure in the "latency < \delta

but we don't know what \delta

is" formulation of the partially synchronous network model.