Casper research:
A specification of Casper the Friendly Ghost

October 30, 2017

Abstract

This work gives a specification and limited experimental observations of a blockchain-based consensus protocol, "Casper the Friendly Ghost."

The protocol uses an adaptation of Y. Jompolovsky and A. Zohar's Greedy Heaviest Observed Sub-tree (Ghost) as "a fork-choice rule". It it able to finalize/decide on blocks with asynchronous and Byzantine fault tolerant consensus safety. It allows blocks to be finalized while the network operates with the network overhead of the Bitcoin blockchain, wich each node receiving $\mathcal{O}(1)$ messages/block. This is contrast to the $\mathcal{O}(N)$ traditionally required for Byzantine fault tolerant state machine replication.

For pedogological reasons, the document first specifies a binary consensus protocol (which decides on a bit, 0 or 1), a protocol that satisfies the same consensus safety proof as the blockchain consensus protocol (and is therefore very similar to the blockchain consensus protocol).

1 Introduction

Consensus protocols are used by nodes in distributed systems to decide on the same values, or on the same list of inputs to a replicated state machine. There are, roughly speaking, two classes of consensus protocols known today. One we refer to as "traditional consensus". This class has its "genetic roots" in Paxos and multi-Paxos[CITE], and in the "traditional" consensus protocol research from the 80s and 90s[CITE]. The other we refer to as "blockchain consensus". These are protocols that have their roots in the Bitcoin blockchain and Satoshi Nakamoto's whitepaper[CITE]. We now discuss the differences between these classes of protocols, before giving an overview of the safety proof, and then finally specifying the consensus protocols at hand.

1.1 Comparing Traditional Consensus to Blockchain Consensus

Traditional consensus protocols (such as multi-Paxos and pbft) are notoriously difficult to understand[CITE (http://paxos.systems/)]. Blockchain consensus protocols, on the other hand, are much more accessible. This difference comes at least in part from the relative simplicity of Bitcoin's specification[CITE].

In the context of state machine replication, traditional protocols decide (with irrevocable finality) in sequence on one "block" of state transitions/transactions to add to the shared operation log at a time. To decide on a block, each node requires $\mathcal{O}(N)$ messages, where N is the number of consensus-forming nodes.

Blockchain consensus protocols like Bitcoin do not finalize/decide on one block one at a time. In fact, the Bitcoin blockchain in particular does not make "finalized decisions" at all; blocks are "orphaned" if/when they are not in the highest total difficulty chain. However, if the miners are able to mine on the same blockchain, then the blocks that get deep enough into the blockchain won't be reverted ("orphaned"). Thus, block-depth serves as a proxy for finalization. In the average case for blockchain consensus protocols, each node only requires approximately one message, $\mathcal{O}(1)$, for every block.

Traditional consensus protocol research has focused on producing protocols that are asynchronously safe (i.e. blocks won't be reverted due to arbitrary timing of future events) and live in asynchrony (or partial synchrony) (i.e. nodes eventually decide on new blocks). On the other hand, the Bitcoin blockchain are safe and live (for unknown block-depth or "confirmation count") in a partially synchronous network.

Traditional Byzantine fault tolerant consensus protocols have precisely stated Byzantine fault tolerance numbers (often $n = 3f + 1$)[CITE]. On the other hand, it is less clear exactly how many faults (measured as a proportion of hashrate) the Bitcoin blockchain protocol can tolerate[CITE].

1.2 Overview of the Work Presented here

The goal of this paper is to give the specification of a consensus protocol, "Casper the Friendly GHOST," which has both the low overhead of blockchain consensus protocols and the asynchronous Byzantine fault tolerant safety normally associated with traditional consensus protocols. However, before we share the blockchain consensus protocol, we will give the specification of a binary consensus protocol (which chooses between 0 and 1 with asynchronous, Byzantine fault tolerant safety).

Understanding the binary consensus protocol makes it much easier to understand the blockchain consensus protocol; the protocols are remarkably similar. They are so similar because they are both "generated" in order to satisfy the same consensus safety proof.

Since this process for choosing the protocol specification is not specified or justified here, but in another paper[CITE], this paper will lack information about *why exactly* certain choices were made, or *how exactly* certain claims are proven. Our aim is nonetheless to leave a reader who does not look at the "process paper" with clear intuitions about why/how the blockchain protocol works: so, we will cover a high level overview of the safety proof which these protocols satisfy. Then, we will begin the presentation of the promised protocols.

1.3 Consensus Safety Proof

Each of the protocols presented here will satisfy the same consensus safety proof. (And indeed, any consensus protocol generated by the correct-by-construction process satisfies the same proof.)

The proof refers to an "estimator", which maps protocol states to propositions about the consensus. In the binary consensus, we will instead think of the estimator as a map from protocol states to 0 or 1. And, in the blockchain consensus, we will consider the estimator to be a map from protocol states to a blockchain (our "fork choice").

An estimate in the binary consensus (0 or 1) is said to be "safe" (have "estimate safety") for a particular protocol state if it is returned by the estimator on all future protocol states (meaning all states accessible from that state through any protocol execution). For the blockchain consensus, a block is said to be "safe" for a paritcular protocol state if it is also in the fork choice for all future protocol states.

The consensus safety proof shows: decisions on safe estimates have consensus safety (as long as there are not more than t Byzantine faults).

The proof relies on the following key result: If node 1 with state σ_1 has safe estimate e_1 and another node 2 with state σ_2 has safe estimate e_2, *and if they have a future state in common* σ_3, then node 1 and node 2's decisions on e_1 and e_2 are consistent. The proof of the result is quite simple and follows without much work from the definition of estimate safety. So first the proof shows that decisions on safe estimates are consensus safe *for any pair of nodes who have a future protocol state in common.*

Next, we have to construct protocols ("protocol states" with "state transitions") which guarantee that nodes will have common future protocol states unless there are more than t Byzantine faults. We will complish this in a few steps.

First, we assume that protocol states are sets of protocol messages and then insist that the union $\sigma_1 \cup \sigma_2$ of any two protocol states σ_1 and σ_2 is itself a protocol state. Further, we insist that there is a state transition from each protocol state σ to $\sigma' \supset \sigma$ (any superset of σ). This means that $\sigma_1 \cup \sigma_2$ is a protocol future of σ_1 and σ_2, both.

This assumption by itself lets any two states always have a common future state, which by itself guarantees consensus safety of decisions on safe estimates. But there's a problem: such a protocol fails to satisfy the non-triviality property of consensus. Non-triviality means that the protocol is able to choose between mutually exclusive values. In our context, non-triviality means that there are two protocol states σ_1 and σ_2 that are each safe on two mutually exclusive estimates. But two protocol states with mutually exclusive safe estimates cannot have a common future, but we just insisted that $\sigma_1 \cup \sigma_2$ would be such a common future. This contradiction means that we haven't yet satisfied non-triviality.

We will instead be sure that σ_1 and σ_2 have common a future *only as long as there are less than t Byzantine faults in* $\sigma_1 \cup \sigma_2$. This will let us have states without shared protocol futures (allowing non-triviality), but still allow us to have consensus safety from our previous result (although now only as long as there are less than t Byzantine faults).

So the next step is to give a process that counts "the number of Byzantine faults" that are evidenced in any given protocol state. In the final step we define a "new version" of our initial protocol by excluding states with more than t Byzantine faults, using the process from the previous step.

And indeed, both of the protocols specified here have the property that for any pair of protocol states σ_1 and σ_2, their union $\sigma_1 \cup \sigma_2$ is also a protocol state if it does not have more than t faults.

So the nodes who decide on safe estimates have consensus safety if there is less than t Byzantine faults. And this required saying almost nothing about the estimator! :)

2 Casper the Friendly Binary Consensus

We will specify the Binary consensus protocol by first defining protocol messages, then by defining the estimator (which maps sets of protocol messages to 0 or 1).

The definition of "protocol messages" is parametric in a set of "validator names" $\mathcal{V}$, which are identified as the names of the consensus forming nodes.

Protocol messages have three parts. An "estimate" (a 0 or a 1), the "sender" (a validator name), and a "justification". The justification is itself a set protocol message. The idea with the protocol message definition is that the message's "estimate" will be the result of applying the estimator on the message's "justification".

So, to be more formal and explicit, protocol messages for the binary consensus have the following form:

$$\begin{aligned}
\mathcal{M}_0 &= \{0,1\} \times V \times \{\emptyset\} \\
\mathcal{M}_n &= \{0,1\} \times V \times \mathcal{P}(\bigcup_{i=0}^{n-1} \mathcal{M}_i) \\
\mathcal{M} &= \lim_{n\to\infty} \bigcup_{i=0}^{n} \mathcal{M}_i
\end{aligned} \tag{1}$$

$\mathcal{M}_0$ is the "base case", the set of messages with "null justifications". $\mathcal{M}_n$ is the set of messages at "height" n, which have messages of height $n-1$ (and/or lower) in their justification. Note that messages $\mathcal{M}_0$ have height 0. $\mathcal{P}$ denotes the "power set" function, which maps sets to the set of all of their subsets, so $\mathcal{P}(\bigcup_{i=0}^{n-1} \mathcal{M}_i)$ denotes all sets of protocol messages at height n or lower.

The estimator is a function with the following signature:

$$\mathcal{E} : \mathcal{P}(\mathcal{M}) \to \{0,1\} \cup \{\emptyset\} \tag{2}$$

But before we can define the estimator, we need a few more basic definitions.

We will define E, a "helper function" that picks out the "estimate" given in a protocol message:

$$E(m) = e \iff m = (e, \_, \_)$$

Similarly, S is a function that picks out the "sender", and finally J is a function that picks out the "justification".

We say that message m_1 is "a dependency" of message m_2, and we write $m_1 \prec m_2$ if:

$$m_1 \prec m_2 \iff m_1 = m_2 \text{ or } m_1 \in J(m_2) \text{ or } \exists m' \in J(m_2) \; . \; m' \prec m_2 \tag{3}$$

So we can define "the dependencies" of a message m as the following

$$D(m) = \{m\} \cup \bigcup_{m' \in J(m)} D(m') \text{ or } \exists m' \in J(m_2) \; . \; m' \prec m_2 \tag{4}$$

Note that D(m) contains every message m' such that $m' \prec m$. Further, it can be expanded in a natural way to define the dependencies of a set of messages (by taking the union of the dependencies of the individual messages).

We will also say that m_2 is "later" than m_1 and write $m_2 \succ m_1$, if we have $m_1 \prec m_2$.

We now have the language to talk about the latest messages from a sender v out of a set of messages M, which we denote as $L(v, M)$:

$$m \in L(v, M) \iff \nexists m' \in D(M) \text{ such that } S(m') = v \text{ and } m' \succ m \tag{5}$$

Latest messages will end up being critical to defining the estimator, which returns 0 if "more" of the nodes have latest messages with estimate 0 than with estimate 1. We will use "weights" for nodes to measure which estimate has "more" consensus forming nodes, implemented by a map from validator names to positive real numbers.

$$W : V \to \mathbb{R}_+$$

From this we can define the "score" of an estimate e in a set of messages M as the total weight of validators with latest messages with estimate e.

$$S(e, M) = \sum_{\substack{v \in V \\ \text{such that } m \in L(v,M) \\ \text{with } E(m)=e}} W(v) \tag{6}$$

Finally, we define the estimator for the Binary consensus:

$$\mathcal{E}(M) = 0 \qquad \text{if } S(0, M) > S(1, M), \tag{7}$$
$$\mathcal{E}(M) = 1 \qquad \text{if } S(1, M) > S(0, M), \tag{8}$$
$$\mathcal{E}(M) = \emptyset \qquad \text{if } S(1, M) = S(0, M) \tag{9}$$

So, at this stage we have protocol messages and an estimator. If we additionally had a way to count Byzantine faults from a set of protocol messages and a way to detect estimate safety, then we would have a consensus protocol.

Byzantine fault detection, in short [not sure where this really belongs in]:

A protocol message m is said to be "valid" if either $\mathcal{E}(J(m)) = \emptyset$ or $E(m) = \mathcal{E}(J(m))$. A pair of protocol messages m_1 and m_2 is an "equivocation" if $S(m_1) = S(m_2)$ and $m_1 \not\succ m_2$ and $m_1 \not\prec m_2$. The weight of Byzantine faults evidences in a set of messages M is

$$F(M) = \sum_{\substack{v \in V \\ \exists m \in M \text{such that } S(m)=v \text{ and } invalid(m), \\ \text{or } \exists m_1, m_2 \in M \text{with } v\,=\,S(m_1) \text{ such that } Equivocation(m_1, m_2)}} W(v) \tag{10}$$

The binary consensus protocol, for some amount of fault tolerance t, will therefore have protocol states $\Sigma \subset \mathcal{M}$ such that $M \in \Sigma \implies F(M) <= t$, and protocol state transitions from any $\sigma_1 \in \Sigma$ to $\sigma_2 \in \Sigma$ if $\sigma_1 \subset \sigma_2$. We have seen from our consensus safety proof that decisions on safe estimates in this protocol are consensus safe if there are less than t Byzantine faults (by weight). It remains to be seen that there are protocol states with estimate safety, and to be shown that estimate safety can sometimes be reasonably efficiently detected.

Now that we have covered the binary consensus protocol, it will be a lot easier to understand the blockchain consensus protocol.

3 Casper the Friendly Ghost

The specification of Casper the Friendly Ghost is going to proceed along *very* similar lines to the binary consensus protocol. We will again define protocol messages and then the estimator, which is going to be the Greedy Heaviest-observed Sub-tree (GHOST) "fork choice rule".

The definition of protocol messages is again going to be parametric in a set of validator names, $\mathcal{V}$.

Protocol messages are called "blocks" and have the same three components as the messages in the binary consensus protocol. The "estimate" is a block, called "the prevblock" or "the parent block". For valid messages, the estimate will be the block on the head of the blockchain chosen by fork choice rule in the justification. The "sender" (a validator name) is defined and treated as before. Finally, the justification is again simply a set protocol message.

So, to be more formal, for the blockchain consensus we have protocol messages (blocks) which have following form:

$$\begin{aligned}
&\text{Genesis Block} = \{\emptyset\} \times \{\emptyset\} \times \{\emptyset\} \\
&\mathcal{M}_0 = \{\text{Genesis Block}\} \times V \times \{\text{Genesis Block}\} \\
&\mathcal{M}_n = \bigcup_{i=0}^{n-1} \mathcal{M}_i \times V \times \mathcal{P}(\bigcup_{i=0}^{n-1} \mathcal{M}_i) \\
&\mathcal{M} = \{\text{Genesis Block}\} \cup \lim_{n \to \infty} \bigcup_{i=0}^{n} \mathcal{M}_i
\end{aligned} \tag{11}$$

The definitions of the "helper functions" E, S, and J, and the definitions of "dependency", "later", "latest messages", and "validator weights" given in the previous section binary consensus also apply unchanged to the blockchain consensus. We will therefore use these definitions here without giving the same definitions again.

Before we are ready define the esimator $\mathcal{E} : \mathcal{P}(\mathcal{M}) \to \mathcal{M}$, though, we will need a couple of more notions.

We will write $m_1 | m_2$ and say that block m_1 is "in the blockchain" of block m_2, if

$$m_1 | m_2 \iff m_1 = m_2 \text{ or } m_1 = E(m_2) \text{ or } m_1 | E(m_2) \tag{12}$$

We can then define the "score" of a block e as

$$S(e, M) = \sum_{\substack{v \in V \\ m \in L(v,M) \\ e | E(m)}} W(v) \tag{13}$$

And the "children" of a block e in a set of protocol messages M are the blocks with e as their prevblock.

$$C(e, M) = \{b \in M : E(b) = e\}$$

We now have the language required to define the estimator for the blockchain consensus, the Greedy Heaviest-Observed Sub-Tree rule!

```
Data: A set of blocks $M$
Result: The block at the head of the fork choice
$b$ = Genesis Block
while $b$ has children ($C(b, M)$ is nonempty) do
    scores = dict()
    for each child of block $b$, $b' \in C(e, M)$ do
        scores[$b'$] = $S(b', M)$
    end
    if scores has a unique maximum then
        $b$ = argmax(scores)
    else
        $b$ = the max score block with the lowest hash
    end
end
return $b$
```

Algorithm 1: The Greedy Heaviest-Observed Sub-tree Fork-choice rule, $\mathcal{E}$

Byzantine fault detection is defined here in precisely the same way as in the binary consensus. We will therefore not give the definitions again. Finally, the protocol states $\Sigma \subset \mathcal{M}$ similarly will have $M \in \Sigma \implies F(M) < t$, and the protocol will (as with the binary consensus) thereby have consensus safety that tolerates up to t Byzantine faults.

4 Safety Oracles

5 Validator Rotation

6 Liveness Considerations