1 Introduction

Renewed interest originating by the Blockchain arena in a long standing problem of Byzantine Fault Tolerant Agreement (BFT) focuses on two challenges: One, the classical foundations do not accommodate external mechanisms for proposer-replacement, e.g. originating by Proof-of-Work or other means. Two, scaling the protocols for proposer-replacement is challenging. This paper addresses both challenges with a BFT protocol called HotSauce that embodies the first fully asynchronous linear proposer-replacement protocol. It is cast in a novel framework called HotStuff that contains a Liveness Gadget that separates the mechanism for guaranteeing progress from the core safety protocol.

The need for a BFT framework in the age of blockchains

It is hard to overstate the stark abyss between the complexity of classical BFT foundations and the simplicity of Nakamoto Consensus (NC).

Classical BFT solutions are constructed using rounds of message exchanges; participants have different roles and need to maintain state from round to round; the safety of the protocols involve intricate voting rules; and liveness hinges on timeouts and suspicion mechanisms. Not only are these classical foundations penetrable only to experts in the field, the inherent complexities and subtleties they encompass led to safety and liveness flaws of some of the best known solutions, as recently exposed in [**?**, **?**].

Here, we focus on BFT solutions that work for an admission-controlled model, where $n = 3f + 1$ members are known, and safety is maintained against f bad members and against asynchrony.

BFT via Nakamoto Consensus (NC) is built around a single broadcast proposal; a proposal is initiated whenever a participant solves a Proof-of-Work puzzle; there is but a single safety rule, favoring the longest known chain of proposals. NC has stood robust in forming agreement on a history of cryptocurrency payments whose current volume is in the order of a trillion dollars.

NC works for a *permissionless* model, where unknown "miner" entities are participating, and safety is maintained against a threshold of the mining power being controlled by bad parties.

The boundaries between the models are blurred in recent hybrid solutions, where a revolving sub-committee is elected among permissionless parties. For example, Bitcoin-NG [**?**] and Byzcoin [**?**] admit leaders/participants among successful miner entities in a blockchain; Solida [**?**] is a chain-less protocol that admit members directly based on compute power; Casper [**?**] admits members by deposit of *stake* previously mined on a blockchain; ALGORAND [**?**] admits members by cryptographic sampling weighed by balance previously mined on a blockchain.

The need to address rotating sub-committee leaders/participants in these hybrid solutions and others rekindle interest in scalability and robustness of BFT solutions in the age of blockchains.

The barrier to scaling proposer-replacement

The crux of the difficulty of BFT solutions is the protocol for transitioning between proposers.

Indeed, the prevailing approach in practical BFT solutions is to optimize for the case of stable leaders (see e.g., PBFT, Zyzzyva, SBFT, Byzcoin, BFT-SMaRt, Tendermint, Casper). These protocols are "practical" in the sense that a stable leader can drive a decision in two exchanges, each of which can be optimized [**?**, **?**, **?**, **?**] to transmit only $O(n)$ (authenticated) values (which is asymptotically optimal).

Are we decentralized yet? Unfortunately, even on"a good day"when the system behaves synchronously (without which, solving BFT is impossible), a cascade of "proposer" (leader) replacements incurs prohibitive costs in all existing solutions.

In section [**?**], we categorize existing PBFT-based into three flavors, vanilla, strawberry, and mango. The first two incur$O(n^2)$communication complexity per proposer replacement, the last one requires a new proposer to wait the maximal network delay. That is, the best practical BFT solutions might require$O(n^3)$transmissions of authenticated values for a single consensus decision, or repeatedly wait the maximal network delay.

Should we care about asymptotic bounds? When Byzantine consensus protocols were originally conceived, a typical target system size was$n = 4$or$n = 7$, tolerating one or two faults. But scaling BFT consensus to(say) $n = 2,000$means that even on a good day,when communication is timely and a handful of failures occurs, quadratic steps require to transmit $4,000,000$ authenticated values. Acascade of failures might bring the communication complexity to whopping$8,000,000,000$ (!) transmissions of authenticated values for a single consensus decision. No matter how good the engineering and how we tweak and batch the system, these theoretical measures are a roadblock for scalability.

Moreover, as indicated above, hybrid blockchains solutions need frequent and uncontrollable proposer regimes.

Contribution and organization

To recap, HotSauce is a BFT solution that embodies the first fully asynchronous, linear proposer-replacement protocol. It is cast in a novel framework called HotStuff that illuminates the BFT world in the lens of blockchains.

More concretely, we present the following contributions..

The rest of the paper is organized as follows.

2 HotStuff and HotSauce

In this section, we describe HotSauce using the HotStuff framework.

The HotStuff framework casts BFT into a block-by-block framework with only a single 'vote' message type. It provides an algorithmic framework for a family of protocols that solve multi-decree, quorum-based BFT consensus, providing the same guarantees as PBFT: Agreement is maintained on a growing sequence (chain) of blocks against a threshold f Byzantine members out of $n = 3f + 1$, and against asynchrony; progress is guaranteed during periods of synchrony.

Blocks. The logical unit of consensus in HotStuff is a *block*. A block proposal can either be *committed* or rejected according to a consensus *decision*, and the goal of the multi-decree BFT consensus is to form agreement on a growing sequence (chain) of blocks. Blocks are opaque to the consensus protocol. In practice, they can contain multiple state-machine 'commands'. For ease of exposition, in our description we ignore batching and treat each block as a single command.

Heights. A block contains a reference to a *parent* block, and a *height* value. Thus blocks are linked into chains of growing heights, forming a DAG rooted at an initial "genesis" block. Note that a height is a unifying counter, replacing commonly used dual counters like a sequence number and a view number.

Two blocks are on the same *branch* if one block is reachable by following parent links from the other. A block *conflicts* with another if they are not on the same branch. Conflicting proposals may occur when a proposer for a height *equivocates*, or when multiple proposers contend for the decision at the same height. By a slight abuse of terminoloy, we will refer to a block-DAG as a *blockchain system*.

a picture would be nice here; it should depict conflicting proposals

Proposers. In HotStuff, there is no distinctive "view-change" phase or specialized messages thereof. Instead, an implicit view change happens whenever a proposer changes. Every replica can potentially be a proposer.

The HotSauce framework contains a separate abstraction called a Liveness Gadget for progress. A Liveness Gadget captures two ingredients. One is to generate blocks that will be *preferred* by all correct parties, in order to be able to gain a QC for the block. The other is to remove contention and elect a unique leader, in order to an uncontended proposal and arrive at a commit decision.

The HotStuff framework decouples leader election mechanisms for liveness from the core used for safety.

A leader is suggested by a separate Liveness Gadget that serves as an oracle. With proper assumptions, as we show later, there are possible designs of the gadget that can provably guarantee liveness. The choice of gadgets will never affect safety. A gadget that implements rotational leadership can cause view change for