What are the Differences between Tendermint and HotShot?

HotShot is a consensus protocol designed and developed by Espresso Systems, which will be used for our sequencing solution. Tendermint[^1] is a well-known, established consensus protocol used by projects such as Cosmos, Polygon, and others.

Does one of these protocols subsume the other? Do we need both of these protocols to exist?

The HotShot protocol is purpose-built for sequencing among a large number of nodes, and it is optimized for this task. We argue that such a solution has benefits that Tendermint does not provide.

At the same time, Tendermint is a battle-tested solution. It is a protocol that works among tens or hundreds of nodes and is designed to work as a full-fledged SMR solution, tackling issues like state changes, which HotShot intentionally chooses not to support.

Therefore, our answer to the two questions above is no and yes, respectively — one system does not subsume the other, and yes, there is a need for both solutions.

Detailed Features

HotShot's design intentionally aims to tackle the requirements of a sequencing solution, which requires consensus. However, the participating nodes do not execute transactions, hence, individual nodes only need assurance of data availability to vote in consensus, not to have full access to the data. On the other hand, the Tendermint protocol satisfies the requirements of SMR, where nodes need to have access to the data, perform consensus, and execution.

Differences in the Key Properties

1. **Separating data availability (DA) and execution from sequencing**. The HotShot implementation is purpose-built for sequencing alone. In particular, it does not perform execution, and the data availability requirement (i.e., ensuring that the system has access to data) is implemented by a separate Tiramisu data availability layer. Such modularity allows the use of various appropriate sub-protocols as needed.

2. **Scalability**. Compared to Tendermint, which requires all-to-all communication, HotShot relies on all-to-leader and leader-to-all communication, thus reducing the consensus communication complexity to linear in the number of nodes. Since sequencing does not require every node to get a full copy of transaction data, low consensus communication is especially important. HotShot combines this with a *content delivery network (CDN)* to efficiently route data and perform computation. This reduces the leader bottleneck and supports a system with a heterogeneous set of nodes. These improvements will help HotShot to scale to thousands of nodes, such that it can be run by a large number of Ethereum validators.

3. **Responsiveness**. Both Tendermint[^2] and HotShot are optimistically responsive and thus under favorable conditions, commit at the network's speed without waiting for any pessimistic network delays. This ensures that the protocols' performance is directly related to the state of the network — under optimistic conditions, the protocol can have low latency and consequently high throughput, too. In HotShot, using a CDN at the network layer synergizes with the optimistic responsiveness property to provide an even better performance.

4. **Maturity**. One of the main advantages Tendermint, and likewise its current implementation inCometBFT, offers is that it is a battle-tested system that has been adopted in multiple blockchains, predominantly the Cosmos ecosystem, but also incorporated in other networks for its core technology, e.g., part of the Polygon network. It has been operated and evaluated in diverse settings. On the other hand, HotShot is in a more nascent state, currently being developed by Espresso Systems.

Technical Differences

- **Authenticator complexity**. The Tendermint protocol embodies two all-to-all voting steps per leader proposal over a peer-2-peer gossip layer.

  For the all-to-all broadcast, each node forwards messages only to its gossip neighbors. The overall communication complexity is quadratic and the latency increases to log(n).

  More importantly, much of the communication load incurred in protocols stems from the need to validate messages, not only from transmitting bits over wires. For this reason, it is sometimes useful to measure complexity differently: instead of counting messages (or bits) transferred over wires, we can measure the total number of message validations processed. The two all-all voting steps in Tendermint each incur a quadratic number of validations.

  HotShot incurs a linear number of messages and a linear number of validations per voting step.

- **Pipelining**. Tendermint is built around a 2-phase core consensus protocol (e.g., depictedhere). The two phases implement two steps of voting by two-thirds of the validators, which are required to successfully commit a block: a pre-vote and a pre-commit. Validators iterate through these two rounds within the same block “height” repeatedly until they reach a commit. They only move to the next height if they reach a commit decision or receive a quorum of votes for a higher block.

  HotShot is pipelined: The second round of voting (“pre-commit”) is simultaneously the first voting round (“pre-vote”) of the next block. Pipelining allows the next proposer to start early, rather than wait for the current proposer to go through two phases. Therefore, despite adding two network hops in HotShot due to linearizing communication, compared with Tendermint, end-to-end latency to transaction settlement may not suffer as much.

- **View change**. One of the hallmarks of the originalTendermint consensus is a simplification of view-change that uses the same mechanism to commit a block as it does to change to the next view. The principle underlying this simplification is embraced in HotShot.

  However, in Tendermint, a faulty view goes through two (nil) voting phases to skip to the next round (as depicted, e.g., here). Due to pipelining, HotShot can skip a faulty view after only one voting step.

- **View synchronization**. Tendermint has an implicit, embedded view synchronization due to its all-all voting phases. In linear protocols like HotShot, the bottleneck shifts to the view synchronization. On balance, HotShot already implements an optimistically linear view-synchronization method designed by Naor-Keidar.

We are grateful to the CometBFT team for contributing useful comments on an earlier draft and helped improve the clarity of this post.

References

1. CometBFT and CometBFT GitHub repository

2. Tendermint: Byzantine Fault Tolerance in the Age of Blockchains

3. The Latest Gossip in BFT Consensus (whitepaper, 2018)

4. Expected Linear Round Synchronization

[^1]:When investigating Tendermint, we examine open source code onGitHub, the original masters thesis and their whitepaper.

[^2]:Their open source code and the whitepaper provides an option to be responsive but theoriginal thesis does not.