**Topic Proposal: Research on HoneyBadgerBFT Asynchronous Consensus Protocol.**

Problem Addressed:

The research proposed will aim to address specific challenges in enhancing scalability of the HoneyBadgerBFT protocol, while maintaining its byzantine fault tolerant properties and decentralization. Despite touting itself as “the first practical asynchronous BFT protocol, which guarantees liveness without making any timing assumptions” (Miller et al., 2016), other researchers like (Duan et al., 2018) have argued“there are still significant pain points and challenges that prevent it from being used in practice.” The challenge is to explore strategies for improving performance and scalability without sacrificing resiliency or decentralization.

Prior Research:

The body of literature below highlights the motivation of this research while also emphasizing the shortcomings.

*The Honey Badger BFT:*

An asynchronous, byzantine fault tolerant consensus protocol designed to ensure secure and decentralized agreement. HoneyBadgerBFT uses a combination of threshold cryptography and other cryptographic techniques to enable consensus even in the presence of malicious nodes. As mentioned in the Rust Crate documentation for HoneyBadgerBFT “Asynchronous protocols do not make assumptions about timing: Even if an adversary controls network scheduling and can delay message delivery, consensus will still be reached as long as all messages are eventually delivered.” Although HoneyBadgerBFT stands out for its robustness against adversarial attacks its shortcomings become apparent in scenarios with a small number of faulty nodes with added overhead from underlying architecture, this can impact performance. It is additionally known that the protocols scalability may be challenged with larger batch sizes, affecting throughput. These limitations highlight the need to optimize asynchronous consensus for better performance and scalability. (Miller et al., 2016)

*The BEAT Protocol:*

Following the release of HoneyBadgerBFT, a collection of asynchronous byzantine fault-tolerant protocols called the BEAT protocols were released. BEAT comprises of five asynchronous BFT protocols that can be combined to achieve meaningful trade-offs between functionality and performance. In the research, BEAT is deployed on Amazon EC2’s across multiple continents and demonstrates superior efficiency in terms of latency and throughput when compared to the HoneyBadgerBFT. The shortcoming of the BEAT protocol suite is they are specialized and not robust enough to be a well-defined solution. The shortcomings of the BEAT protocols will be discussed below.

* *Network Bandwidth Consumption:* BEAT1 and BEAT2 achieve higher throughput than HoneyBadgerBFT when the batch size is small, but they decrease in throughput when the batch size exceeds 5000. Leading to performance degradation. This suggests that BEAT might struggle to maintain high throughput in scenarios with larger batch sizes.
* *Batch Size Scalability:* BEAT0, BEAT1, and BEAT2 become more visible as the batch size increases. This implies that BEAT might face scalability limitations when processing larger batch sizes of transactions.
* *Network Scalability:* BEAT3 consistently outperforms HoneyBadgerBFT, but when the number of nodes increases these protocols become more latent. This diminishes BEAT3’s efficiency in large-scale networks.
* *Small Number of Faulty Nodes*: BEAT4 experiences added overhead when the number of faulty nodes is small. This suggests that BEAT4 is inefficient in scenarios with a small number of faulty participants.

(Duan et al., 2018)

*Asynchronous Byzantine Fault Tolerant Protocol:*

Following the works of BEAT came the Asynchronous Byzantine Fault Tolerance (ABFT) protocol. This protocol was designed to address challenges posed by blockchain systems in conjunction with the Internet of Things (IoT). ABFT argues that traditional consensus protocols like Raft and PBFT that assume a fast and stable network are moot points in real-world scenarios, and that low-quality and asynchronous networks should be assumed. ABFT aims to provide robustness against these challenges. ABFT leverages the concept of Asynchrouns Common Subset from the HoneyBadgerBFT and builds upon it to exhibit significantly lower computational overhead, higher performance, and greater scalability compared to the HoneyBadgerBFT. ABFT shortcomings are as follows.

* *I/O Bound Protocol:* During the Reliable Broadcast (RBC), the larger the batch size the greater the chance of idle time waiting for messages in the I/O operations. This indicates that ABFT's performance can be constrained by the input-output operations, particularly when dealing with larger batch sizes.
* *Network Resource Utilization:* Since ABFT depends on the (N) parties and (B) transaction batch size. As both numbers increase so does the data being transmitted, leading to higher network usage limiting scalability.
* *Threshold ECDSA Scheme:* ABFT does not implement the full threshold ECDSA schema and uses precomputed signing material for optimization. This raises questions to the resources needed to maintain precomputed material and suggests that this implementation doesn't truly harness the advantages of threshold ECDSA
* *Performance Impact of Network Degradation:* ABFT performance is robust when the number of nodes affected by network degradation is fewer than the fault tolerance threshold, but when the number of affected nodes surpasses that tolerance threshold ABFT’s performance drops and leads to latency. This indicates that ABFT’s performance can suffer in scenarios where a large portions of nodes experience network issues.
* *Wait Times for Affected Parties:* When subset of nodes are affected by network issues the remaining parties may need to wait for the affected to catch up. This can introduce delays and impact the overall progress of ABFT.

(Knudsen et al., 2021)

**Significance:**

Researching asynchronous consensus holds significance in the field of distributed computing and blockchain technology. Having nodes achieve consensus is paramount for the integrity and reliability of transactions. Having confidence in these transactions are especially important in scenarios where nodes can be malicious or faulty. Asynchronous consensus protocols address the challenges of unpredictable message delays, network partitions, and node failures, making them particularly relevant in real-world scenarios with a focus in where communication is typically unpredictable. In contrast the HoneyBadgerBFT argues that “protocols based on timing assumptions are unsuitable for decentralized, cryptocurrency settings “(Miller et al., 2016) signifying the focus that asynchronous consensus should be the standard chosen for decentralized distributed systems. Because decentralization and fault tolerance are paramount in the blockchain domain, asynchronous consensus should be chosen to enable systems to operate reliably even in the face of adverse or malicious nodes. The BEAT protocol argues for the research of asynchronous consensus because ““While robustness is natively achieved in asynchronous BFT, we still require different designs and trade-offs for different performance metrics.” (Duan et al., 2018) signifying there is no one size fits all solution tailored towards asynchronous consensus. The need to research asynchronous consensus is also supported in the works of ABFT as stated “in some blockchain-based systems, e.g., supply chain management (SCM) systems, some IoT nodes can only rely on the low-quality network sometimes to achieve consensus. As an answer to such challenges, an asynchronous BFT consensus protocol is warranted” (Knudsen et al., 2021) signifying that importance of asynchronous consensus when relying on low-quality networks such as in decentralized distributed systems. From the previous research cited above, we can see there is a collective agreement in decentralized distributed systems for the research of asynchronous consensus.

**Methodology:**

The methodology of this study will involve researching and enhancing the asynchronous consensus protocols HoneyBadgerBFT using the implementation provided by Poanetworks in the Rust programming language. The source code is available at the specified GitHub repository (<https://github.com/poanetwork/hbbft>). The following steps outline the methodology and justify its appropriateness:

*Understanding HoneyBadgerBFT*: Will begin by thoroughly understanding the HoneyBadgerBFT protocol's implementation in the provided RUST codebase. This will include grasping its underlying algorithm, message propagation, cryptographic mechanisms, and fault tolerance mechanisms.

*Performance Analysis*: Perform a performance analysis of the HoneyBadgerBFT protocol via AWS, Azure, or Tor. Measure the performance metrics such as throughput, latency, and resource utilization. The analysis provides a baseline understanding of the protocol's behavior under varying network conditions.

*Benchmarking*: Design and execute bench marking experiments on various network configurations such as varying network delays, and packet losses, to simulate real-world scenarios. Taking note to measure the protocol's responsiveness and effectiveness in achieving consensus across different conditions. This step helps identify performance bottlenecks and areas for improvement.

*Optimization Strategies*: Investigate potential optimization strategies for the HoneyBadgerBFT protocol. This may involve optimizing cryptographic operations, message propagation mechanisms, or parallelizing computation. Implement these optimizations in the RUST codebase and assess their impact on performance. A List of issues to get started on these strategies can be found here (https://github.com/poanetwork/hbbft/issues).

*Comparative Analysis*: Compare the performance of the optimized HoneyBadgerBFT protocol against its original version. Measure how the optimizations affect throughput, latency, and resource consumption. The comparison validates the effectiveness of the proposed enhancements.

*Fault Tolerance Testing*: Simulate Byzantine faults in the network to evaluate the protocol's fault tolerance capabilities. Measure the protocol's ability to detect and mitigate Byzantine behavior while maintaining consensus.

*Real-world Deployment Simulation*: Simulate a real-world deployment scenario with a geographically distributed network using the RUST implementation on AWS, Azure, or Tor. Evaluate the protocol's performance and resilience across different network topologies and distances.

*Reporting*: Document the entire study process, including the steps taken, optimizations implemented, and results obtained. Present the findings in a comprehensive report that includes graphs, charts, and statistics. Provide insights into the impact of optimizations on protocol behavior and performance.

The selected methodology is justified because it provides empirical analysis of the protocol's behavior over realistic simulations. The metrics will be measurable such as throughput, latency, and resource consumption, that will allow for a comparative evaluation. The study will be reproducible such that it can be easily replicated from documentation in the comprehensive reporting. Lastly the methodology is justified because the Rust implementation is relevant to real-world applications, making the studies outcomes directly applicable to the field of distributed systems and blockchain technology. Overall, the methodology will balance theoretical understanding and practical implementation, making it suitable for evaluating and enhancing asynchronous consensus protocols in a realistic, replicable, and impactful manner.

## **References**

Baird, L., & Luykx, A. (2020). The Hashgraph Protocol: Efficient Asynchronous BFT for High-Throughput Distributed Ledgers. *International Conference on Omni-layer Intelligent Systems*, pp. 1-7. <https://doi.org/10.1109/COINS49042.2020.9191430>

Duan, S., Reiter, M., & Zhang, H. (2018). BEAT: Asynchronous BFT Made Practical. *In ACM SIGSAC Conference on Computer and Communications Security,* pp. 2028–2041. <https://doi.org/10.1145/3243734.3243812>

Hood, K., Oglio, J., Nesterenko, M., & Sharma, G. (2021). Partitionable Asynchronous Cryptocurrency Blockchain. *IEEE International Conference on Blockchain and Cryptocurrency,* pp. 1-9. https://doi.org/10.1109/ICBC51069.2021.9461080

Knudsen, H., Li, J., Notland, J., Haro, P., & Ræder, T. (2021). High-Performance Asynchronous Byzantine Fault Tolerance Consensus Protocol. *IEEE International Conference on Blockchain*, pp. 476-483. <https://doi.org/10.1109/Blockchain53845.2021.00073>

Miller, A., Xia, Y., Croman, K., Shi, E., & Song, D. (2016). The Honey Badger of BFT Protocols. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 31–42. <https://doi.org/10.1145/2976749.2978399>