High-Performance Asynchronous Byzantine Fault Tolerance Consensus Protocol

1st Henrik Knudsen
Bouvet
Bergen, Norway
henrik.knudsen@bouvet.no

2nd Jingyue Li
Department of Computer Science
Norwegian University of Science and Technology
Trondheim, Norway
0000-0002-7958-391X

3rd Jakob Svennevik Notland
Department of Computer Science
Norwegian University of Science and Technology
Trondheim, Norway
jakob.notland@ntnu.no

4th Peter Halland Haro
Sintef Nord
Tromsø, Norway
peter.haro@sintef.no

5th Truls Bakkejord Ræder
Sintef Nord
Tromsø, Norway
truls.rader@sintef.no

2021 IEEE International Conference on Blockchain (Blockchain) | 978-1-6654-1760-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/BLOCKCHAIN53845.2021.00073

***Abstract*—In response to new and innovating blockchain-based systems with Internet of Things (IoT), there is a need for consensus mechanisms that can provide high transaction throughput and security, despite varying network quality. Honeybadger was the first practical, asynchronous Byzantine Fault Tolerance (BFT) consensus protocol, achieving high scalability and robustness without making any timing assumptions regarding the network. To improve the current asynchronous consensus protocols, we designed Asynchronous Byzantine Fault Tolerance (ABFT) consensus protocol through integrating threshold Elliptic Curve Digital Signature Algorithm (ECDSA) signatures and optimization of erasure coding parameters, as well as additional implementation-level optimizations. We implement a prototype of ABFT, and evaluate its performance at scale in a global WAN network and a network affected by asymmetric network degradation. Our results show that ABFT provides considerably higher performance, significantly lower computational overhead, and greater scalability than its predecessors. ABFT can reach up to 38.700 transactions per second in throughput. Furthermore, we empirically show that ABFT is unaffected by asymmetric network degradation within the fault threshold.**

***Index Terms*—Blockchain, consensus protocol, high-performance, asynchronous, Byzantine fault tolerance**

I. Introduction

The consensus algorithm of blockchain systems enables participants to reach an agreement in a decentralized fashion. Most blockchain technologies assume an environment with a fast and stable network to achieve consensus. For example, Practical Byzantine Fault Tolerance (PBFT) [1] and Raft [2] require eventual synchrony in order to provide forward progress. If these timing assumptions cannot be satisfied, the blockchain systems' transaction throughput halts. Additionally, the protocols are vulnerable towards Denial of Service (DoS) and timing attacks [3], [4].

However, 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, providing robustness against such attacks. [5] showed that such protocols might be constructed using Asynchronous Common Subset (ACS) [6], i.e., a set of peers may asynchronously agree upon a set of transactions. A practical problem of this particular construction, however, was its O(n^3) communication complexity, caused by the expensive usage of the Multi-valued Validated Byzantine Agreement (MVBA) [5] primitive, making the protocol unable to scale.

Honeybadger BFT [3] was a major breakthrough in the endeavor to create a practical asynchronous BFT algorithm. By using a novel construction of ACS, the communication complexity of the algorithm could be brought down to O(n^2), similar to that of PBFT. [3] further showed that Honeybadger was competitive with PBFT in a realistic deployment, in which it outperformed PBFT in terms of transaction throughput for networks scaling past 16 nodes. However, a challenge of the original Honeybadger BFT protocol was its runtime complexity. The specific construction of ACS Honeybadger BFT used, while it is more efficient in communication complexity, incurred a $O(\log n)$ runtime complexity. [7] presented an alternative construction of ACS, which carefully utilizes MVBA, allowing a reduction of runtime complexity down to $O(1)$. A more efficient construction of MVBA [8] reduced the protocol's communication complexity from O(n^3) to O(n^2). Other contributions, e.g. [4], have been dedicated to optimizing the initialization of Honeybadger BFT.

In an endeavor towards improving the state-of-the-art asynchronous consensus protocols, we intended to consolidate these individual improvements into a single consensus protocol, denoted ABFT. Furthermore, we seek to lower the computational overhead and increase its scalability. We additionally want to investigate the consensus protocol's performance in a truly asynchronous network affected by asymmetric network degradation. Our study answered the following research questions:

RQ1: How to integrate state-of-the-art technologies to achieve high-performance ABFT consensus protocol?

978-1-6654-1760-0/21/$31.00 ©2021 IEEE
DOI 10.1109/Blockchain53845.2021.00073

RQ2: To what degree is ABFT's performance impacted by an asynchronous network, with asymmetric, high network delay and packet loss?

We consolidated the aforementioned improvements of [4] and [7]. We provided additional innovation of introducing threshold Elliptic Curve Digital Signature Algorithm (ECDSA) signatures for use within the context of ABFT. Furthermore, we show the importance of optimizing erasure coding parameters has for the protocol's performance and provide a framework for empirically precomputing these parameters, enabling dynamic, optimal choice during runtime. Finally, we provide additional implementation-level optimizations in the form of optimistic verification of signatures, and precomputing cryptographic material, providing additional performance benefits. We implement a prototype of ABFT using the Rust programming language and evaluate it in a global WAN network. In addition, we evaluate its performance in an asynchronous network, with asymmetric, high network delay and packet loss, emulating network degradation using Network Emulator (NetEm) [9], [10]. Our evaluation shows:

- ABFT has a smaller computational overhead than its predecessors but provides several times lower transaction latency than previous implementations [3], [4], [7].
- ABFT scales well, with transaction throughput reaching up to 38.700 transactions per second for larger batch sizes while keeping transaction latency within a minute.
- ABFT is unaffected by asymmetric, high network delay and packet loss, given that the number of affected nodes stays within the fault threshold. If it exceeds the fault threshold, the consensus protocol still terminates, without the need for countermeasures or additional configurations.

The rest of this paper is organized as follows. Section 2 presents the design and implementation of ABFT. Section 3 shows the evaluation results. Section 4 presents a brief overview of related work. Section 5 discusses the results, and Section 6 concludes and gives our future work.

II. ABFT DESIGN AND IMPLEMENTATION

Starting with the Honeybadger construction [3], we utilize the asymptotically more efficient ACS construction of [7]. We instantiate ACS using the communication optimized MVBA construction of [8], and the bandwidth optimized Reliable Broadcast (RBC) construction of [11]. For the protocol's cryptosystems, we adopt the TDH2 threshold encryption scheme of [12], and threshold coin flipping scheme of [13], motivated by the experimental results of [4]. We use the threshold ECDSA signature scheme of [14] to improve the pairing-based signature scheme originally used by Honeybadger. We use the ristretto255 group [15] for each of the cryptosystems to yield 128 bits of security. Additionally, we use precomputation to increase the performance of encryption and signature operations. For erasure coding, we adopt the Cauchy Read-Soloman (CRS) code implementation of Jerasure [16], inspired by [4] and the performance evaluation of [17]. We go on to optimize its parameters for use within ABFT.

In the following sections, improvements over the related work [3], [4], [7] are presented. In particular, we describe how performant threshold ECDSA signatures can be used within ABFT, the optimization of word and packet sizes for CRS codes, and how precomputing of cryptographic material may yield additional performance benefits during runtime.

A. Threshold ECDSA signatures

The original Honeybadger construction [3] utilizes the threshold Boneh–Lynn–Shacham (BLS) [18] signature scheme provided by [19]. Threshold BLS signature schemes have the advantage of non-interactive, short signatures, and interesting properties like signature aggregation [20], but suffer from several efficiency and security issues. In particular, BLS signature schemes require the use of pairing-friendly curves, which require larger key sizes than conventional curves [21]. Furthermore, BLS signature schemes rely on relatively expensive pairing operations [22], yielding poor performance compared to alternatives like ECDSA [23], [24]. Concerning ECDSA and Curve25519, we specifically refer to the Ristretto255 group [15], built on top of Curve25519.

In the context of ABFT, we deem that the following qualities are desired from its threshold signature scheme:

- **Efficiency**: Parties should be able to produce and verify signatures with minimal cost.
- **Non-interactive**: Sub-protocols within ABFT expect a one-round signing procedure. Additional rounds of communication would severely degrade performance, especially in a global network. Thus, the threshold signature scheme should be non-interactive in terms of producing a threshold signature share.
- **Fault attributability**: Honest parties must be able to detect and identify corrupt parties which produces invalid threshold signature shares to take appropriate actions.

While previous threshold ECDSA schemes have required a significant amount of communication rounds (i.e., interactivity) [25]–[27], multiple recent works have yielded non-interactive threshold signature schemes [14], [28], [29], utilizing precomputation of signing material to reduce communication during the actual signing process. Additionally, some of the recent threshold signature schemes also support *identifiable abort* [14]. If the threshold signature scheme produces an invalid signature, parties can unanimously identify the corrupted party that offers fault attributability. We provide a deterministic mapping of signature operations within ABFT. Additionally, we show how we can use optimistic signature verification to increase ABFT performance.

1) Deterministic mapping of signature operations within ABFT: A caveat related to the use of precomputed signing material is their restriction to one-time use. In particular, reusing the same signing material for multiple signatures is deemed insecure, as it can enable an adversary to extract information about the party's secret key share [14]. In a more traditional setting, the latter is seldom a problem. If a group of parties wants to generate signatures for multiple messages, selecting corresponding signing material is trivial as long as

there is a mutual agreement on the ordering of the messages. However, in the context of ABFT, this poses a challenge, given that we cannot guarantee the ordering in which messages arrive at different parties. Our proposal for a deterministic mapping of signature operations within ABFT enables parties to agree upon what signing material to use for a particular signature without assuming any particular order of messages. The variables related to the proposal are explained in table I.

TABLE I
DETERMINISTIC MAPPING VARIABLES

Variable	Note
N	The number of parties participating in the ABFT consensus protocol.
f	The number of faulty parties participating in the consensus protocol, assuming $N \geq 3f + 1$.
i	The unique identifier of a particular party, having $N > i \geq 0$.
r	The current round number of ABFT.
k	The current round number of MVBA.
K	The total number of rounds required to terminate an instance of MVBA.
s	The current step of a proposal promotion sub-protocol within MVBA, having $s \in \{1, 2, 3, 4\}$, in which 4 steps of proposal promotion guarantees that at least $f + 1$ honest parties commit a party's proposal [8].

The following signing operations may take place during a single round of ABFT:

- **Provable Reliable Broadcast (PRBC)**: Each party generates a signature to certify that it has broadcasted its value using the Reliable Broadcast (RBC) sub-protocol.
- **MVBA (Proposal promotion)**: During each step of proposal promotion, each party generates a signature for its proposal to certify that it has promoted the signature to sufficiently many parties.
- **MVBA (Promotion completion)**: After completing proposal promotion, parties generate a signature for the current round k and certify that enough proposals have been promoted, in order to progress to the election process.

For each round of ABFT, we thus have up to:

- N signing operations, one for each instance of PRBC.
- $K \times 4N$ signing operations, 4 per call to proposal promotion and per round of running MVBA.
- K signing operations, one per round of running MVBA.

This equates to a maximum of $N + K \times (4N + 1)$ signing operations in total.

Given N parties participating in the sub-protocol MVBA, the minimum probability that the protocol terminates for a given round is at least 1/2 [8]. For a given K, the cumulative probability that MVBA terminates within the K'th round is therefore at least $1 - 2^{-K}$. Choosing an appropriate value for K, parties can precompute $S = N + K \times (4N + 1)$ signing material to be used in a round of ABFT, with a negligible probability that the number of signing operations will exceed S. Given the non-determinism of MVBA, should the sub-protocol terminate in some smaller round number K', the unspent signing materials can be reused in a future round of ABFT. We deem that the aforementioned deterministic mapping enables usage of threshold signature schemes using precomputing of signature material within the context of ABFT, despite its asynchronous and non-deterministic nature.

2) Optimistic verification of signatures: Within MVBA and PRBC, a common use case of the threshold signature API is:

- Party P_i waits to receive t signature shares for a particular message M.
- For each unique signature share σ_j received from Party P_j, Party P_i verifies that σ_j is a valid share from P_j for the message M.
- When Party P_i has successfully received t signature shares, they combine them into a proper signature σ.

If parties act honestly (at least for the current round), verifying each share adds unnecessary overhead to the consensus protocol at N scales. Given the context of ABFT, we assume that appropriate actions are taken towards identified parties, and corrupted parties and behavior are rare. Thus, we can optimistically neglect verifying shares as they are received. Rather, after constructing the proper signature σ, we rely on the verification of the signature instead, performing identifiable abort if it fails. This yields the following altered use case:

- Party P_i waits to receive t signature shares for a particular message M.
- When Party P_i has successfully received t signature shares, they combine them into a proper signature σ.
- Party P_i verifies the validity of the signature σ:
  - If it succeeds, we continue the protocol as normal.
  - If it fails, we know that at least one of the shares included in the signature combination operation was corrupted. Then, we perform an identifiable abort, verify the validity of each signature share, and decide which parties are corrupted.

Should the verification operation fail, we can quickly resume running ABFT with little to no additional delay compared to the original API of MVBA and PRBC. By discarding the identified, corrupted shares, we can continue receiving shares from non-corrupted parties until we have sufficiently many shares to re-combine them into a signature. Given a signature scheme with threshold t, we verify at most $t + f$ shares, irrespective of the number of identifiable aborts, in which f is the fault tolerance. We also perform at most $a + 1$ signature combination and verification operations, where a is the number of aborts performed. Thus, in the worst case, where an honest party receives $t - 1$ honest shares, followed by f corrupted shares, we have $a = f$, and thus incur a total of $t + f$ share verifications and $f + 1$ signature combination and verifications. Given the assumption that aborts are rare, the potential cost of additional operations is considered negligible.

3) Illustration of performance increase: To illustrate the potential gains of incorporating the threshold ECDSA scheme, we conduct a performance benchmark of the signing operations used within ABFT. We compared our implementation of the threshold ECDSA scheme of [14] with the implementation of the [19] threshold BLS scheme using threshold_crypto [30].

We set the number of participants N as 100 and threshold T as $N/4$. The results of the comparison are shown in Table II. The comparison was implemented using criterion.rs [31] and ran on an Intel i7-3770K CPU @ 3.50GHz, with 16 GB of RAM. We utilize the curve25519_dalek cryptographic library [32] for implementing the threshold ECDSA signature scheme. The results indicate improvements in performance , especially for the **Share-Sign** and **Verify-Signature** operations.

TABLE II
SIGNING PERFORMANCE OF ECDSA AND BLS SIGNATURE SCHEMES.

Operation	ECDSA	BLS	Relative Improvement
Share-Sign	7.0043 us	3.8290 ms	54366.4 %
Combine-Shares	10.902 ms	14.189 ms	30.2 %
Verify-Signature	58.528 us	7.8209 ms	13262.7 %

We furthermore conduct a performance benchmark of the ABFT consensus protocol itself. We set N as 8, f as 2, and B as 10.000, and run 10 iterations of each of the protocols locally using the different threshold signature schemes. We calculated the average required runtime for the protocol to terminate. The results are in Table III and show significant improvements.

TABLE III
RESULTS OF COMPARING ABFT RUNTIME PERFORMANCE, UTILIZING THRESHOLD ECDSA AND THRESHOLD BLS SIGNATURE SCHEMES, RESPECTIVELY.

	ECDSA	BLS	Relative Improvement
ABFT Runtime	238.28 ms	3.41 s	1331 %

B. Optimal choice of word size and packet size for erasure coding

An important step of ABFT is the dispersion of the parties' input value into the protocol (e.g., the transaction set). This is done using the RBC sub-protocol, originally proposed in [33], which once terminated ensures that sufficiently amount of parties have received the input value. Since the input value may grow very large as the batch size B grows, the consensus protocol utilizes the bandwidth optimized construction presented in [11], which incorporates erasure coding to reduce the amount of data broadcasted. Regarding the performance of erasure coding operations, the following parameters are relevant:

- **Number of blocks**: The number of data and coding blocks, denoted k and m in the context of ABFT. These are constrained to be $N - 2f$ and $2f$, respectively.
- **Input size**: The amount of data to be coded. In the context of ABFT, this scales linearly with the batch size B.
- **Block size**: The size of each data and coding block. For CRS erasure codes, this is often described as the product of word size w and packet size p [16]. Furthermore, there is a constraint that $k + m = N > 2^w$.

While N and B fix the number of blocks and the input size, the block size may be altered as one sees fit. In particular, the choice of w and p may have a significant impact on the performance of erasure coding operations [17]. Furthermore, the applicability of different values for w and p may vary for different configurations of N and B. There seems to be no universal, optimal choice for w and p, irrespective of N and B. The choice of w and p may additionally have complex effects on the performance of the erasure coding scheme. What data structures are used internally and how these relate to platform-specific instructions (e.g., Single instruction, Multiple Data (SIMD)) and cache behavior [17] make an analytical choice of w and p challenging. The Honeybadger implementation utilizes Read-Soloman (RS) erasure codes and uses the zfec Python library [34]. The use of zfec brings with it several efficiency and usability issues [4]: zfec does not allow fine-tuning of erasure coding parameters because it uses a fixed word size $w = 8$. This may be an inefficient choice for certain configurations of N and B and putting an unnecessary limit on N to be less than $2^8 = 256$. Alternatively, [4] proposes using Jerasure's [16] CRS codes, allowing user-specified values for w and p, and provides a highly optimized implementation [17].

We adopt the choice of using the CRS codes of [16] and endeavor to find optimal choices for w and p within the context of ABFT. For a given hardware environment and a configuration of N and B, we can calculate optimal choices for w and p empirically in the following manner:

- Define an evaluation function (i.e., a benchmark) to accurately evaluate the performance of an erasure coding scheme with the configuration N, B, w and p.
- Define a range of appropriate word sizes, W.
- Define a range of suitable packet sizes, P.
- For each w in W, execute a search algorithm (i.e., simulated annealing or hill climbing) over the packet sizes in P, evaluate the corresponding erasure coding scheme at each point, and record its performance.
- Choose the pair (w, p) with the highest performance for each configuration of N and B.

The method builds upon the assumption that there is a single (few) good choices for the word size w, and that for each word size w, there exists a single, optimal choice for its packet size (i.e., there exists some global maximum). Using CRS codes, the performance peaks with the smallest possible word size [17]. Given N, we can thus choose the smallest w with $N < 2^w$, as well as r of its successors, giving $W = \{w, w+1, ..., w+r\}$, for some small constant r. Larger packet size allows for more efficient matrix operations, with an inherent trade-off that larger block sizes have worse cache utilization. We achieve optimal performance with maximum utilization of L1 cache [17]. P should therefore at least be extended to a value p_{cache}, in which $w \times p_{cache}$ is some multiple of the host machine's L1 cache storage capacity.

To demonstrate, we conducted a simplified version of the aforementioned method for ABFT, constraining the packet sizes P to powers of 2, up to and including 2^{16}. We evaluated the erasure coding scheme's proficiency for the operations needed within ABFT using the following benchmark.

- Encoding a payload T of 250 bytes per transaction.
- Decoding an encoding of T, with up to f erasures.

- Decoding an encoding of T, with up to $2f$ erasures.

Knowing what hardware environment ABFT is going to be deployed onto, we stored the aforementioned evaluation results in a $N_{len} \times B_{len}$ matrix, where N_{len} and B_{len} are the number of different party and batch sizes one would want to use during runtime, respectively. This can then be used as a lookup table within the consensus protocol, allowing dynamic, the optimal choice of w and p based on the input values N and B. This might be especially useful if one wants to dynamically change the batch size B between rounds of running ABFT, allowing one to dynamically update w and p during runtime, ensuring optimal erasure coding parameters are used at all times, avoiding performance degradation.

To illustrate the performance difference between using optimal and sub-optimal parameters for w and p, we perform a mini evaluation of the erasure coding operations used within ABFT. We set $N = 100$, $f = 25$, and evaluate the performance of erasure encoding 250 bytes per transaction, for batch sizes B = 100, 100.000 and 2.000.000, respectively. Given $k + m = 100$, we choose the smallest possible word size $w = 7$, and compare performance using packet sizes of $p = 1$ and $p = 8192$. The results of the benchmarks are shown in Table IV and indicate that the performance depends heavily on the choice of the packet size p. Data in Table IV also show that the optimal choices are dependent on the batch size B.

TABLE IV
BENCHMARK OF ERASURE ENCODING, WITH K = 50, M = 50, W = 7, USING DIFFERENT PACKET SIZES.

Operation	w = 7, p = 1	w = 7, p = 8192	Relative difference
Encode, B = 100	206.00 us	30.877 ms	14988.8 %
Encode, B = 100.000	144.86 ms	27.805 ms	521 %
Encode, B = 2.000.000	2.9298 s	56.702 ms	5167 %

C. Precomputing of cryptographic material

An important aspect within most public-key cryptosystems is to provide performant exponentiation within a given group, usually of prime order. In the context of Elliptic Curve Cryptography (ECC) systems, this involves multiplying some point g on the curve by a scalar r. There exist efficient ways of doing this for arbitrary g and r, namely variants of the square-and-multiply method [35].

However, for a fixed base g, performance improvements can be gained by precomputing tables of intermediate values and storing them in-memory. By looking up these values during runtime, the number of multiplications needed for the exponentiation can be effectively reduced [36]–[38]. This comes at the expense of storing the precomputed tables. In order to achieve optimal efficiency, the lookup tables may need to fit into the host's cache. This is for instance true in the case of Curve25519 [39]. For the concrete curve Curve25519, in which scalars are represented as 256-bit integers, a reasonable choice for most modern architectures is to write scalars in radix 16 to yield a total lookup table size of 30 KB [39]. Within the context of ABFT, we can improve performance by utilizing precomputing for any fixed point g used in the protocol's threshold cryptosystems. This applies to the generator of the curve, G, but also any other long-lived points. In particular, we can precompute lookup tables for parties' public keys during the startup of the protocol.

To illustrate the potential gains of utilizing precomputing, we implement the TDH2 threshold encryption scheme of [12], with and without precomputing of public keys. We perform a benchmark of the scheme's operations with number of parties $N = 100$ and threshold $T = 25$, and compare the performance of the two implementations. The results of the benchmark can be found in table V.

TABLE V
BENCHMARK OF THD2 THRESHOLD ENCRYPTION SCHEME, WITH AND WITHOUT PRECOMPUTING OF PUBLIC KEYS.

Operation	TDH2	TDH2 precomputed	Relative improvement
Encrypt	415.35 us	128.28 us	223.8 %
Decrypt-Share	239.50 us	175.22 us	36.7 %
Verify-Share	301.17 us	199.89 us	50.7 %
Combine-Shares	18.500 ms	18.286 ms	-

D. Prototype Implementation

The differences in the construction of ABFT compared to the Honeybadger Python implementation [3] reduced the benefit of code reuse. Thus, we opt to implement a prototype of ABFT from scratch using the Rust programming language Rust [40] allows for high performance and concurrency while guarantees memory safety, which makes it a good fit for an inherently asynchronous system because multi-threading would be beneficial. We implement asynchronous network communication among parties using Tokio, Hyper, and Tonic, i.e., a performant network stack within the Rust ecosystem [41]. All threshold cryptographic primitives are implemented from scratch using the curve25519_dalek cryptographic library [32], which provides group operations for ristretto255 [15] and is built on top of Curve25519 [42]. The overall implementation consists of 10600 lines of Rust code [43]

As a caveat for the threshold signature scheme, we do not implement the full protocol. In particular, we do not implement the zero-knowledge proofs required for precomputing signing material *without* a trusted dealer. This has no impact on the performance of the consensus protocol, and robust precomputation of signing material could be added later. Furthermore, we evaluate the protocol under the honest setting (e.g., no parties send faulty signature shares), i.e., aborting the algorithm if an invalid signature is produced. Finally, to allow the usage of Jerasure's [16] CRS codes, we additionally write a wrapper library to enable allocation of core data structures as well as encoding and decoding operations from Jerasure.

III. EVALUATION RESULTS

To answer RQ1, we evaluated the performance of ABFT in a stable network and compared it with the implementations of [3], [4], [7], according to their experimental results as

seen in table VI. We answered RQ2 by evaluating how asymmetric, high packet delay, and packet loss impact ABFT's performance. We did not compare ABFT's RQ2 results with related work because [3], [4], [7] did not present RQ2-related results.

In the evaluation, parties executing the ABFT protocol were deployed onto AWS EC2 t2.medium instances, each with 2 virtual CPUs and 4GB memory. Throughout our evaluation, we utilized a fixed transaction size of $m_T = 250$ bytes.

TABLE VI
HONEYBADGER IMPLEMENTATIONS EVALUATED

Name	Runtime Complexity	Crypto Scheme
Honeybadger [3]	$O(\log n)$	BLS (80 bits)
BEAT0 [4]	$O(\log n)$	Non-BLS (128 bits)
Dumbo1 [7]	$O(\log k)$	BLS (80 bits)
Dumbo2 [7]	$O(1)$	BLS (80 bits)
ABFT	$O(1)$	ECDSA (128 bits)

A. Results of RQ1

We first evaluated ABFT in a LAN network, comparing its bare latency to the implementation of [4]. Then, we evaluated ABFT in a global WAN network, comparing its bare latency to the implementation of [7]. Finally, we evaluated ABFT in a global WAN network with batch sizes B ranging from 100 to 2×10^6. We evaluate ABFT in several separate experiments in order to give consistent results because there are slight differences in experimental setup between [3], [4], [7].

1) Bare latency - LAN: Evaluating results show that ABFT can provide fast termination, in particular for configurations up to and including 16 nodes. ABFT can have a transaction latency at most 120 ms, when each node proposes a single transaction. Comparing our results with the experimental results of [4], this is deemed a significant improvement. When N is 7, the Honeybadger implementation terminates closer to 1.5 seconds, and BEAT0 terminates at around 0.5 seconds. Additionally, in ABFT, the transaction latency seems to grow significantly slower as the number of nodes increases, compared to Honeybadger and BEAT0.

2) Bare latency - WAN: Evaluating ABFT's bare latency in a global WAN network configuration show that ABFT manages to terminate in less than 7 seconds, up to and including 100 nodes. Figure 1 shows the comparison of our results with the experimental results of [7]. This is deemed a noteworthy improvement. When evaluating for $N = 100$, ABFT provide a 70% reduction in transaction latency over Dumbo2 [7], and a 99% reduction over Honeybadger [3].

3) Latency, Throughput, Scalability, Resource Utilization: The results of evaluating ABFT in a WAN network configuration, using batch sizes B ranging from 100 to 2×10^6, show that ABFT is able to terminate within a minute for all configurations of N and B, up to and including $N = 100$ and $B = 2 \times 10^6$. Translated to transaction throughput, this equates to a throughput of around 38.700 transactions per second at a maximum.

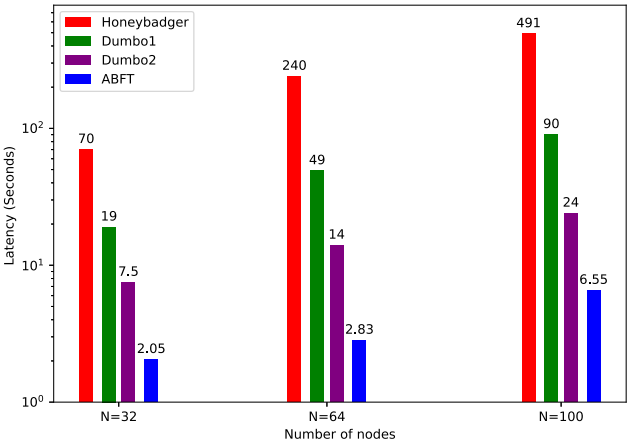

Fig. 1. Bare latency of Honeybadger, Dumbo1, Dumbo2 and ABFT in a WAN network configuration. The evaluation uses $N = 4f$, and includes results for $N = 32, 64, 100$. Nodes are uniformly distributed across geographical regions around the globe. Data for Honeybadger, Dumbo1, and Dumbo2 are based on experimental results of [7].

In an endeavor to compare our results with the experimental results of [7], Figure 2 presents a comparison of transaction throughput for Honeybadger, Dumbo1, Dumbo2, and ABFT, for various values of N, with a fixed, maximum batch size of $B = 2 \times 10^6$. In particular, for $N = 32$, ABFT provides a 2.5 times increase in transaction throughput over Dumbo2.

In addition, ABFT is evaluated at a security level of 128 bits, while the implementations evaluated by [7] were evaluated at a lower security level of 80 bits. A higher security level usually leads to more expensive cryptographic operations, which reduces performance.

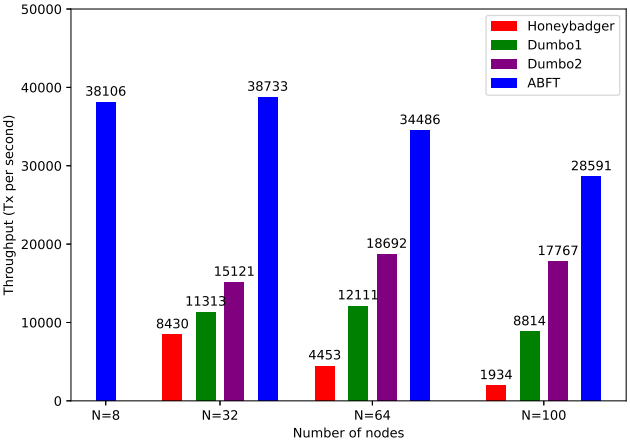

Fig. 2. Transaction throughput of Honeybadger, Dumbo1, Dumbo2 and ABFT in a WAN network configuration, with a batch size $B = 2 \times 10^6$. Data for Honeybadger, Dumbo1, and Dumbo2 are based on experimental results of [7].

ABFT's network utilization grows sharply as the batch size B increases, which reaches a maximum of around 1

GB outbound traffic for $B = 2 \times 10^6$ irrespective of N. Furthermore, network utilization also depends on the number of parties N, in which the minimum required network usage grows with an increase in N. This is as expected. As we include more parties in our system and increase the number of the transaction being agreed upon, the amount of data being transmitted correspondingly increases. It is worth noting that when evaluating the ABFT consensus protocol for larger batch sizes, the protocol is clearly I/O bound. In particular, at the start of the protocol, in which the input value (i.e., the transaction set) is dispersed using the RBC sub-protocol, system logs indicate that there is often a several seconds long idle time in which the protocol has not received any messages yet, and simply has to wait. In the context of asynchronous consensus protocols like ABFT, it is, therefore, deemed prudent to utilize infrastructure with sufficiently large network capacity to strike a better balance in terms of resource utilization. This could allow for additional performance gains without significantly increasing costs.

B. Results of RQ2

Our evaluation results show that ABFT is not affected by asymmetric network degradation if the number of nodes affected, M, is less than the fault tolerance f. In particular, during our evaluation, using $N = 8$, $f = 2$, $M = 2$, $B = 10.000$, have no significant impact on the performance as the network degrades. Considering the construction of ABFT, this is to be expected. Since ABFT does not require the collaboration of more than $N - f$ parties at any point.

ABFT's performance degrades as M grows larger than the fault tolerance f. In particular, during our evaluation, using $N = 8$, $f = 2$, $M = 4$, the latency grow steadily, with performance decreasing harshly as the network degrades. At the maximum, the protocol's transaction latency is over 17 times higher, when imposed by additional packet delay of 5000 ms. However, ABFT still guarantees both termination and security, without the need for ad-hoc changes to the protocol nor additional configuration. We expect there to be a performance degradation, given several message thresholds requiring $N - f$ to achieve progress. In all of these instances, the remaining parties must wait upon at least some of the affected parties.

Due to space limitation, more detailed data and charts related to RQ1 and RQ2 are in [44].

IV. Related work

Although many consensus protocols have been proposed, few of them provide high-performance and good security in the context of asynchronous and low-quality network environment [45]. The work of [3] proposed a practical, asynchronous BFT protocol. Since then, there has been renewed interest within the space of asynchronous BFT, such as [7] [8].

Our work differentiates from the related work by introducing threshold ECDSA signatures, providing a deterministic mapping of signature operations, and allowing transition from the previously used BLS signatures. Furthermore, we provide a framework for optimizing erasure coding parameters based on the number of parties N and the transaction batch size B. As additional optimizations, we introduce optimistic signature verification and use precomputing for cryptographic material to enable higher performance for cryptographic operations. We have conducted extensive performance comparisons of ABFT and existing consensus protocols. In addition, we evaluated ABFT in an asynchronous network affected by asymmetric, high network delay, and packet loss. Based on this, we deem the implementation of ABFT as a significant step forward within the context of practical, asynchronous consensus protocols and a natural point of extension for future improvements.

V. Discussions

A. Limitations of ABFT

One potential limitation of ABFT is that we do not implement the full threshold ECDSA scheme. The precomputing of the signing material is done using a trusted dealer. Related to this, we do not evaluate the cost of running the precomputing protocol for the threshold ECDSA scheme in the context of ABFT. There seems to be a significant performance benefit of using threshold ECDSA signatures within ABFT, but this might be somewhat diminished by the cost of running the precomputing protocol. It is, however, important to note that the precomputing protocol can be delegated to other machines. Thus, the concern is related to the amount of additional computational resources needed to maintain enough precomputed signing material for each round of ABFT and how the cost of this compares to the performance benefit of threshold ECDSA signatures over the previously used threshold BLS signatures.

B. Threats to Validity

We adopt the experimental setup of our related work [3], [4], [7]. In particular, we adopt their choice of independent variables, their experiments, and their choice of the testing infrastructure. While we believe that these are sensible choices and that it is necessary for us to adopt these choices to have comparable results to their work, it limits the scope of the experimental evaluation. In particular, most of our experimental evaluation bases itself on a global WAN network, spanning multiple geographic regions, in which parties have a reasonable amount of computational, memory, and network resources. As the related work was conducted some years back, there might be a concern about the instrumental change.

The ABFT prototype is built using a different programming language than the other implementations. This leads to the use of different networking and cryptography libraries. Furthermore, we use different serialization schemes. This might be a minor external validity concern to what extent our improvements can be generalized to other systems.

VI. Conclusions and future work

In this study, we improved the Asynchronous Byzantine Fault Tolerance consensus protocol and designed a high-performance one. The evaluation results show that our consensus protocol is faster and more reliable than state-of-the-art

protocols. Our planned work is to pursue further improvement of ABFT's performance for unstable network conditions.

ACKNOWLEDGMENT

This work is jointly supported by the National Key Research and Development Program of China (No. 2019YFE0105500) and the Research Council of Norway (No. 309494 and 274816).

REFERENCES

[1] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, pp. 173–186.

[2] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference (USENIX-ATC 14)*, pp. 305–319.

[3] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, p. 31–42. [Online]. Available: https://doi.org/10.1145/2976749.2978399

[4] S. Duan, M. K. Reiter, and H. Zhang, "Beat: Asynchronous bft made practical," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, p. 2028–2041. [Online]. Available: https://doi.org/10.1145/3243734.3243812

[5] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *Advances in Cryptology — CRYPTO 2001*, J. Kilian, Ed. Springer Berlin Heidelberg, pp. 524–541.

[6] M. Ben-Or, B. Kelmer, and T. Rabin, "Asynchronous secure computations with optimal resilience (extended abstract)," in *Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing*. ACM, p. 183–192. [Online]. Available: https://doi.org/10.1145/197917.198088

[7] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo: Faster asynchronous bft protocols," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, p. 803–818. [Online]. Available: https://doi.org/10.1145/3372297.3417262

[8] I. Abraham, D. Malkhi, and A. Spiegelman, "Asymptotically optimal validated asynchronous byzantine agreement," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. ACM, p. 337–346. [Online]. Available: https://doi.org/10.1145/3293611.3331612

[9] F. Ludovici and H. P. Pfeifer, "Netem - network emulator at linux.org," 2011. [Online]. Available: https://www.linux.org/docs/man8/tc-netem.html

[10] S. Hemminger, "Network emulation with netem," *Linux Conf.*, 2005.

[11] C. Cachin and S. Tessaro, "Asynchronous verifiable information dispersal," in *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pp. 191–201.

[12] V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," ser. Advances in Cryptology — EUROCRYPT'98. Springer Berlin Heidelberg, pp. 1–16.

[13] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography," *Journal of Cryptology*, vol. 18, no. 3, pp. 219–246, 2005. [Online]. Available: https://doi.org/10.1007/s00145-005-0318-0

[14] R. Gennaro and S. Goldfeder, "One round threshold ecdsa with identifiable abort," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 540, 2020.

[15] H. d. Valence, "The ristretto group," 2021. [Online]. Available: https://ristretto.group/

[16] J. S. Plank, S. Simmerman, and C. D. Schuman, "Jerasure: A library in c/c++ facilitating erasure coding for storage applications-version 1.2," *University of Tennessee, Tech. Rep. CS-08-627*, vol. 23, 2008.

[17] J. Plank, J. Luo, C. Schuman, L. Xu, and Z. Wilcox-O'Hearn, *A Performance Evaluation and Examination of Open-Source Erasure Coding Libraries For Storage*, 2009.

[18] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," ser. Advances in Cryptology — ASIACRYPT 2001. Springer Berlin Heidelberg, pp. 514–532.

[19] A. Boldyreva, "Efficient threshold signature, multisignature and blind signature schemes based on the gap-diffie-hellman-group signature scheme," 2002.

[20] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," ser. Advances in Cryptology — EUROCRYPT 2003. Springer Berlin Heidelberg, pp. 416–432.

[21] A. Menezes, P. Sarkar, and S. Singh, "Challenges with assessing the impact of nfs advances on the security of pairing-based cryptography," ser. Paradigms in Cryptology – Mycrypt 2016. Malicious and Exploratory Cryptology. Springer International Publishing, pp. 83–108.

[22] M. Scott, "Implementing cryptographic pairings," *Lecture Notes in Computer Science*, vol. 4575, p. 177, 2007.

[23] B. Lynn, "Pbc library - the pairing-based cryptography library," 2013. [Online]. Available: https://crypto.stanford.edu/pbc/

[24] D. Moody, R. Peralta, R. Perlner, A. Regenscheid, A. Roginsky, and L. Chen, "Report on pairing-based cryptography," *Journal of research of the National Institute of Standards and Technology*, vol. 120, pp. 11–27, 2015. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/26958435 https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4730686/

[25] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ecdsa with fast trustless setup," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, p. 1179–1194. [Online]. Available: https://doi.org/10.1145/3243734.3243859

[26] Y. Lindell and A. Nof, "Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, p. 1837–1854. [Online]. Available: https://doi.org/10.1145/3243734.3243788

[27] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, "Threshold ecdsa from ecdsa assumptions: the multiparty case," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 1051–1066.

[28] R. Canetti, N. Makriyannis, and U. Peled, "Uc non-interactive, proactive, threshold ecdsa," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 492, 2020.

[29] A. Gagol and D. Straszak, "Threshold ecdsa for decentralized asset custody," Cryptology ePrint Archive, Report 2020/498, 2020. https://eprint.iacr.org/2020/498, Tech. Rep., 2020.

[30] Poanetwork, "threshold_crypto," 2021. [Online]. Available: https://github.com/poanetwork/threshold_crypto

[31] bheisler, "Criterion.rs," 2021. [Online]. Available: https://github.com/bheisler/criterion.rs

[32] dalek cryptography, "curve25519-dalek," 2021. [Online]. Available: https://github.com/dalek-cryptography/curve25519-dalek

[33] G. Bracha, "Asynchronous byzantine agreement protocols," *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987.

[34] Z. Wilcox-O'Hearn, "zfec 1.5.5," 2013. [Online]. Available: https://pypi.org/project/zfec/

[35] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 2018.

[36] N. Pippenger, "On the evaluation of powers and related problems," in *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, pp. 258–263.

[37] E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson, "Fast exponentiation with precomputation," ser. Advances in Cryptology — EUROCRYPT' 92. Springer Berlin Heidelberg, pp. 200–207.

[38] C. H. Lim and P. J. Lee, "More flexible exponentiation with precomputation," ser. Advances in Cryptology — CRYPTO '94. Springer Berlin Heidelberg, pp. 95–107.

[39] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of cryptographic engineering*, vol. 2, no. 2, pp. 77–89, 2012.

[40] Rust-lang, "Rust," 2021. [Online]. Available: https://www.rust-lang.org/

[41] Tokio.rs, "Tokio," 2021. [Online]. Available: https://tokio.rs/

[42] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *International Workshop on Public Key Cryptography*. Springer, pp. 207–228.

[43] H. Knudsen, "Asynchronous byzantine fault tolerance," 2021. [Online]. Available: https://github.com/Henriknu/consensus-unstable-throughput

[44] H. Knudsen, "High-performance asynchronous byzantine fault tolerance consensus protocol," 2021. [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2787249

[45] H. Knudsen, J. S. Notland, P. H. Haro, T. B. Raeder, and J. Li, "Consensus in blockchain systems with low network throughput: A systematic mapping study," in *2021 3rd Blockchain and Internet of Things Conference*, ser. BIOTC 2021. ACM, 2021, p. 15–23. [Online]. Available: https://doi.org/10.1145/3475992.3475995