Efficient Asynchronous Atomic Broadcast

Andrew Miller
University of Maryland, College Park

Yu Xia
Tsinghua University

Kyle Croman
Cornell University

Elaine Shi
Cornell University

Dawn Song
University of California, Berkeley

2016/07/25

Abstract

The surprising success of cryptocurrencies has led to a surge of interest in deploying large scale, highly robust, Byzantine fault tolerant (BFT) protocols for mission-critical applications, such as financial transactions. Although the conventional wisdom is to build atop a (weakly) synchronous protocol such as PBFT (or a variation thereof), such protocols rely critically on network timing assumptions, and only guarantee liveness when the network behaves as expected. We argue these protocols are ill-suited for this deployment scenario.

We present an alternative, HoneyBadgerBFT, the first practical *asynchronous* BFT protocol, which guarantees liveness without making any timing assumptions. We base our solution on a novel atomic broadcast protocol that achieves optimal asymptotic efficiency for large batches, improving by $O(N^2)$ compared to the prior known best due Cachin et al [4].

1 Introduction

Distributed fault tolerant protocols are promising solutions for mission-critical infrastructure, such as financial transaction databases. Traditionally, they have been deployed at relatively small scale, and typically in a single administrative domain where adversarial attacks might not be a primary concern. As a representative example, a deployment of Google's fault tolerant lock service, Chubby, consists of five nodes, and tolerates up to two crash faults.

In recent years, a new embodiment of distributed systems called "cryptocurrencies" or "blockchains" have emerged, beginning with Bitcoin's phenomenal success [2, 8]. Cryptocurrency systems challenge our traditional belief about the deployment environment for fault tolerance protocols. Unlike the classic "5 Chubby nodes within Google" environment [3], cryptocurrencies have revealed and stimulated a new demand for consensus protocols over a wide area network, among a large number of nodes that are mutually distrustful, and moreover, network connections can be much more unpredictable than the classical LAN setting, or even adversarial. This new setting poses interesting new challenges, and calls upon us to rethink the design of fault tolerant protocols.

Most existing Byzantine fault tolerant (BFT) systems, even those called "robust," assume some variation of *weak synchrony*, where, roughly speaking, messages are guaranteed to be delivered after a certain bound Δ, but Δ may be time-varying or unknown to the protocol designer. We argue that protocols based on timing assumptions are unsuitable for decentralized, cryptocurrency settings, where network links can be unreliable, network speeds change rapidly, and network delays may even be adversarially induced.

2 Practical asynchronous BFT

We propose HoneyBadgerBFT, the first BFT *atomic broadcast* protocol to provide *optimal asymptotic efficiency* in the asynchronous setting. We therefore directly refute the prevailing wisdom that such protocols a re necessarily impractical.

We make significant efficiency improvements on the best prior-known asynchronous atomic broadcast protocol, due to Cachin et al. [4], which requires each node to transmit $O(N^2)$ bits for each committed transaction, substantially limiting its throughput for all but the smallest networks.

2.1 Problem Definition: Atomic Broadcast

We first define our network model and the atomic broadcast problem. Our setting involves a network of N designated nodes, with distinct well-known identities ($\mathcal{P}_0$ through $\mathcal{P}_{N-1}$). The nodes receive transactions as input, and their goal is to reach common agreement on an ordering of these transactions. Our model particularly matches the deployment scenario of a "permissioned blockchain" where transactions are submitted by arbitrary clients.

Our network model assumes asynchronous, reliable, authenticated, point-to-point channels between each pair of parties. The delivery schedule is entirely determined by the adversary, but every message sent between correct nodes must eventually be delivered. Our failure model is *static Byzantine faults*, where the adversary is given control of up to f faulty nodes, where f is a protocol parameter. Note that $3f+1 \leq N$ (which our protocol achieves) is the lower bound for broadcast protocols in this setting. For ease of presentation, we assume that nodes may interact with a trusted dealer during an initial protocol-specific setup phase, which we will use to establish public keys and secret shares.

Definition 1. *An atomic broadcast protocol must satisfy the following properties, all of which should hold with high probability (as a function of a security parameter, λ) in an asynchronous network and in spite of an arbitrary adversary:*

- *(Agreement)* If any correct node outputs a transaction tx, then every correct node outputs tx.
- *(Total Order)* If one correct node has output the sequence of transactions $\{\mathsf{tx}_0, \mathsf{tx}_1, ...\mathsf{tx}_j\}$ and another has output $\{\mathsf{tx}'_0, \mathsf{tx}'_1, ...\mathsf{tx}'_j\}$, then $\mathsf{tx}_i = \mathsf{tx}'_i$ for $i \leq j$.
- *(Validity)* If a transaction tx is input to $N-f$ correct nodes, then it is eventually output by every correct node.

The validity property is called *Censorship Resilience*, since it precludes an adversary from preventing even a single transaction from being committed.[1]

2.2 Informal Protocol Description

In HoneyBadgerBFT, nodes receive transactions as input and store them in their (unbounded) buffers. The protocol proceeds in epochs, where after each epoch, a new batch of transactions is appended to the committed log. At the beginning of each epoch, nodes choose a subset of the transactions in their buffer (by a policy we'll define shortly), and provide them as input to an instance of a randomized agreement protocol. At the end of the agreement protocol, the final set of transactions for this epoch is chosen.

Our protocol is centered around an instance of the Asynchronous Common Subset (ACS) primitive. Roughly speaking, the ACS primitive allows each node to propose a value, and guarantees that every node node outputs a common vector containing the inputs values of at least $N-2f$ correct nodes. It is trivial to build atomic broadcast from this primitive — each node simply proposes a subset of transactions from the front its queue, and outputs the union of the elements in the agreed-upon vector. However, there are two important challenges.

Challenge 1: Achieving Censorship-resilience. The cost of ACS depends directly on size of the transaction sets proposed by each node. Since the output vector contains at least $N-f$ such sets, we can therefore improve the overall efficiency by ensuring that nodes propose *mostly disjoint* sets of transactions, thus committing more distinct transactions in one batch for the same cost. Therefore instead of simply choosing the first element(s) from its buffer (as in CKPS01 [4]), each node in our protocol proposes a randomly-chosen sample, such that each transaction is, on average, proposed by only one node.

[1] We remark that the validity aka censorship resilience property has also been called "fairness" in other literature on atomic broadcast (e.g., [4]. We avoid this term, since it also conflicts with a different definition of "fairness" from the literature on multi-party computation and fair contract signing.

Algorithm HoneyBadgerBFT (for process $\mathcal{P}_i$)

Let $B = O(\lambda N^2 \log N)$ be the batch size parameter.
Let PK be the public key from $\mathsf{TPKE.Setup}$ (executed by a dealer), and let SK_i be the secret key for $\mathcal{P}_i$.
Let $\mathsf{buf} := [\,]$ be a FIFO queue of input transactions.
Proceed in consecutive epochs numbered r:

// Step 1: Random selection and encryption

- let proposed be a random selection of $\lfloor B/N \rfloor$ transactions from the first B elements of buf
- encrypt $x := \mathsf{TPKE.Enc}(\mathsf{PK}, \mathsf{proposed})$

// Step 2: Agreement on ciphertexts

- pass x as input to $\mathsf{ACS}[r]$
- receive $\{v_j\}_{j \in S}$, where $S \subset [1..N]$, from $\mathsf{ACS}[r]$

// Step 3: Decryption

- for each $j \in S$:
  - let $e_j := \mathsf{TPKE.DecShare}(\mathsf{SK}_i, v_j)$, and multicast $\mathtt{DEC}(r, j, i, e_j)$
  - wait to receive at least $f+1$ messages of the form $\mathtt{DEC}(r, j, k, e_{j,k})$
  - decode $y_j := \mathsf{TPKE.Dec}(\mathsf{PK}, \{(k, e_{j,k})\})$
- let $\mathsf{block}_r := \mathsf{sorted}(\cup_{j \in S}\{y_j\})$, such that block_r is sorted in a canonical order (e.g., lexicographically)
- set $\mathsf{buf} := \mathsf{buf} - \mathsf{block}_r$

Figure 1: **HoneyBadgerBFT.**

However, implemented naïvely, this optimization would compromise censorship resilience, since the ACS primitive allows the adversary to choose *which* nodes' proposals are ultimately included. We avoid this pitfall by using a threshold encryption, TPKE, which prevents the adversary from learning which transactions are proposed by which nodes, until after agreement is already reached. A *robust threshold encryption* scheme TPKE is a cryptographic primitive that allows any party to encrypt a message to a master public key, such that the network nodes must work together to decrypt it. Once $f+1$ correct nodes compute and reveal *decryption shares* for a ciphertext, the plaintext can be recovered; until at least one correct node reveals its decryption share, the attacker learns nothing about the plaintext. The robustness property furthermore guarantees that any two nodes decrypt a (potentially malicious) ciphertext to the same values.

Challenge 2: Efficient Instantiation of ACS. Prior known implementations of asynchronous atomic broadcast [4, 5], use an inefficient instantiation of ACS. However, by stitching together a carefully chosen array of sub-components, we can efficiently instantiate ACS and attain much greater throughput both asymptotically and in practice. Specifically, we observe that a reduction from ACS to reliable broadcast[2]) due to Ben-Or et al. [1], composed with an efficient reliable broadcast using erasure codes [6] leads to an $O(N)$ efficiency improvement. A self-contained presentation of this construction can be found in our full online version.

Analysis. First we observe that the agreement and total order properties follow immediately from the definition of ACS and robustness of the TPKE scheme.

Theorem 1. (Agreement and total order). *The HoneyBadgerBFT protocol satisfies the agreement and total order properties, except for negligible probability.*

Proof. These two properties follow immediately from properties of the high-level protoocls, ACS and TPKE. Each ACS instance guarantees that processes agree on a vector of ciphertexts in each epoch (Step 2). The robustness of TPKE guarantees that each correct node decrypts these ciphertexts to consistent values (Step 3). This suffices to ensure agreement and total order. □

We next analyze the *efficiency* of our atomic broadcast protocol. Assume that the input buffers of each honest node are sufficiently full $\Omega(\mathsf{poly}(N, \lambda))$. Then *efficiency* is the expected communication cost for

[2] The reliable broadcast primitive is also referred to as "A-cast" when discussed in the context of an asynchronous network.

each node amortized over all committed transactions. We start by showing that the expected communication complexity for each epoch is bounded by $O(B)$.

Theorem 2. (Complexity). *Assuming a batch size $B = O(\lambda N^2 \log N)$, the running time for each HoneyBadgerBFT epoch is $O(\log N)$ in expectation, and the total expected communication complexity is $O(B)$.*

Proof. The cost and running time of ACS [1], using reliable broadcast [6] are $O(\log N)$ rounds and is $O(N|v| + \lambda N^2 \log N)$ assuming each party proposes a value of size $|v|$. Since each party proposes $O(B/N)$ messages, the total cost is absorbed as $O(B)$. The N instances of threshold decryption incur one additional round, so the overall running time is $O(\log N)$ rounds. □

The HoneyBadgerBFT protocol may commit up to B transactions in a single epoch. However, the actual number may be less than this, since some correct nodes may propose overlapping transaction sets, others may respond too late, and corrupted nodes may propose an empty set. Fortunately, we prove in the full online version [7] that assuming each correct node's queue is full, then $B/4$ serves as an lower bound for the expected number of transactions committed in an epoch.

Theorem 3. (Efficiency). *Assuming each correct node's queue contains at least B distinct transactions, then the expected number of transactions committed in an epoch is at least $\frac{B}{4}$, resulting in constant efficiency.*

The above definition of efficiency assumes the network is *under load*, reflecting our primary goal: to sustain high throughput while fully utilizing the network's available bandwidth.

In practice, network links have limited capacity, and if more transactions are submitted than the network can handle, a guarantee on confirmation time cannot hold in general. Therefore we prove (in the full version of our paper [7]) that the adversary cannot significantly delay the commit of any transaction.

Theorem 4. (Censorship Resilience). *Suppose an adversary passes a transaction* `tx` *as input to $N - f$ correct nodes. Let T be the size of the "backlog", i.e. the difference between the total number of transactions previously input to any correct node and the number of transactions that have been committed. Then* `tx` *is commited within $O(T/B + \lambda)$ epochs except for negligible probability.*

References

[1] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience. In *Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing*, pages 183–192. ACM, 1994.

[2] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. Kroll, and E. W. Felten. Research perspectives on bitcoin and second-generation digital currencies. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015.

[3] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 335–350. USENIX Association, 2006.

[4] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *Advances in Cryptology – Crypto 2001*, pages 524–541. Springer, 2001.

[5] C. Cachin, J. Poritz, et al. Secure intrusion-tolerant replication on the internet. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 167–176. IEEE, 2002.

[6] C. Cachin and S. Tessaro. Asynchronous verifiable information dispersal. In *Reliable Distributed Systems, 2005. SRDS 2005. 24th IEEE Symposium on*, pages 191–201. IEEE, 2005.

[7] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of bft protocols. `http://eprint.iacr.org/2016/199`, 2016.

[8] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `http://bitcon.org/bitcoin.pdf`, 2008.