Balanced Quadratic Reliable Broadcast and Improved Asynchronous Verifiable Information Dispersal

Sourav Das, Ling Ren, Zhuolun Xiang

University of Illinois at Urbana-Champaign
{souravd2, renling, xiangzl}@illinois.edu

Abstract

In this paper, we present an asynchronous Byzantine reliable broadcast (RBC) protocol with balanced costs and an improved asynchronous verifiable information dispersal (AVID) protocol. Our RBC protocol broadcasts a message M among n nodes with total communication cost $O(n|M|+\kappa n^2)$ and per-node communication cost $O(|M|+\kappa n)$. In contrast, the state-of-the-art reliable broadcast protocol has imbalanced costs where the broadcaster incurs $O(n|M|)$ while other nodes incur a communication cost of $O(|M|+\kappa n)$. We then use our new RBC protocol and additional techniques to design an asynchronous verifiable information dispersal (AVID) protocol with total dispersal cost $O(|M|+\kappa n^2)$ and retrieval cost $O(|M|+\kappa n)$. In our AVID protocol, the clients incur a communication cost of $O(|M|+\kappa n)$ in comparison to $O(|M|+\kappa n \log n)$ cost of prior best AVID protocol that does not require any trusted setup. Moreover, each node in our AVID protocol incurs a storage cost of $O(|M|/n + \kappa)$ bits, whereas in prior best AVID protocol each node incurs a storage cost of $O(|M|/n + \kappa \log n)$ bits. Finally, we present lower bound results on per-node communication cost of any deterministic RBC protocol and the total communication cost of dispersal and retrieval phase in any deterministic AVID protocol. Both our balanced RBC and AVID protocols have near-optimal communication costs.

1 Introduction

Reliable broadcast (RBC) and Verifiable information dispersal (VID) are fundamental primitives in distributed computing [9], and have many applications such as fault-tolerant consensus and replication [23, 14, 16, 21], secure multiparty computation [20, 28], verifiable secret sharing [11], and distributed key generation [1, 19, 12]. The goal of RBC is to have a designated broadcaster send its input message and to have all nodes output the same message. VID lets a *client*, here on referred to as the *dispersing* client, disperse a message among a set of nodes such that the message can be later retrieved by any node or any other client, which we refer to as the *retrieving* client. In this paper, we consider these two problems in asynchronous networks and we assume Byzantine faults that may deviate arbitrarily from the protocols. The problem of asynchronous VID (AVID) was introduced by Cachin and Tessaro [10]. A protocol for AVID immediately implies a protocol for asynchronous RBC, where the broadcaster acts as the dispersing client and each node retrieves the data by acting as a retrieving client.

Existing works. The first RBC protocol due to Bracha [9] has a total communication cost of $O(n^2|M|)$, where n is the number of protocol nodes and $|M|$ is the size of the broadcaster's message in bits. Two decades later, Cachin and Tessaro [10], assuming a collision resistant hash function, proposed a RBC protocol that has a total communication cost of $O(n|M|+\kappa n^2 \log n)$. Here κ is the size of output of the hash function. In both of these RBC protocols, every node, including the broadcaster, incurs the same asymptotic communication cost. Here on, we will refer to such a RBC protocol as a *balanced* RBC protocol. The state-of-the-art asynchronous RBC protocol due to Das et al. [11], which has a total communication cost of $O(n|M|+\kappa n^2)$ and requires no trusted setup, however, has an *unbalanced* communication cost. The cost of the broadcaster is approximately n times higher than that of other nodes, leading to a bottleneck at the broadcaster. We

Table 1: Comparison with Existing RBC protocols. The following acronyms are used in the table; q-SDH: q-Strong Diffie-Hellman, DBDH: Decisional Bilinear Diffie-Hellman. The upper bound for rounds means bad-case latency [3], and the lower bound for rounds means lower bound for good-case latency when the broadcaster is honest [2].

Scheme	Communication Cost (broadcaster)	Communication Cost (other node)	Communication Cost (total)	Rounds	Cryptographic Assumption	Setup
Bracha [9]	$O(n\|M\|)$	$O(n\|M\|)$	$O(n^2\|M\|)$	4	None	None
Cachin-Tessaro [10]	$O(\|M\|+\kappa n \log n)$	$O(\|M\|+\kappa n \log n)$	$O(n\|M\|+\kappa n^2 \log n)$	4	Hash	None
Nayak et al. [24]	$O(\|M\|+\kappa n)$	$O(\|M\|+\kappa n)$	$O(n\|M\|+\kappa n^2)$	7	q-SDH+DBDH	Trusted
Das et al. [11]	$O(n\|M\|)$	$O(\|M\|+\kappa n)$	$O(n\|M\|+\kappa n^2)$	4	Hash	None
This work	$O(\|M\|+\kappa n)$	$O(\|M\|+\kappa n)$	$O(n\|M\|+\kappa n^2)$	5	Hash	None
Lower bound	$\Omega(\|M\|+n)$	$\Omega(\|M\|+n)$	$\Omega(n\|M\|+n^2)$	2	—	—

Table 2: Comparison with existing AVID protocols. The following accronyms are used in the table; DL: Discrete Logarithm, CRS: Common Reference String, q-SDH: q-Strong Diffie-Hellman.

Scheme	Dispersal Cost (client)	Dispersal Cost (total)	Retrieval Cost (total)	Storage Cost (total)	Cryptographic Assumption	Setup
Cachin-Tessaro [10]	$O(\|M\|+\kappa n \log n)$	$O(n\|M\|+\kappa n^2 \log n)$	$O(\|M\|+\kappa n \log n)$	$O(\|M\|+\kappa n \log n)$	Hash	None
Hendricks et al. [17]	$O(\|M\|+\kappa n^2)$	$O(\|M\|+\kappa n^3)$	$O(\|M\|+\kappa n^2)$	$O(\|M\|+\kappa n^2)$	Hash	None
Alhaddad et al. [4]	$O(\|M\|+\kappa n \log n)$	$O(\|M\|+\kappa n^2)$	$O(\|M\|+\kappa n \log n)$	$O(\|M\|+\kappa n \log n)$	DL	CRS
Alhaddad et al. [4]	$O(\|M\|+\kappa n)$	$O(\|M\|+\kappa n^2)$	$O(\|M\|+\kappa n)$	$O(\|M\|+\kappa n)$	q-SDH+Hash	Trusted
DisperseLedger [27]	$O(\|M\|+\kappa n \log n)$	$O(\|M\|+\kappa n^2)$	$O(\|M\|+\kappa n \log n)$	$O(\|M\|+\kappa n \log n)$	Hash	None
This work	$O(\|M\|+\kappa n)$	$O(\|M\|+\kappa n^2)$	$O(\|M\|+\kappa n)$	$O(\|M\|+\kappa n)$	Hash	None
Lower bound	$\Omega(\|M\|+n)$	$\Omega(\|M\|+n^2)$	$\Omega(\|M\|+n)$	$\Omega(\|M\|)$	—	—

provide a detailed comparison with the RBC protocol of [11] in Table 1 and discuss other related work in detail in §6.

Cachin et al. [10] presented the first AVID protocol with a total communication cost of $O(n|M|+\kappa n^2 \log n)$ during the dispersal phase and $O(|M|+\kappa n \log n)$ during the retrieval phase. Here κ is size of output of the hash function. Moreover, in their protocol, both dispersing client and retrieving client incurs a communication cost $O(|M|+\kappa n \log n)$ during the dispersal and retrieval phase, respectively. Also, each node needs incurs a storage cost of $O(|M|+\kappa \log n)$. Hendricks et al. [17] gave a protocol with communication cost $O(|M|+\kappa n^3)$ for the dispersal phase and $O(|M|+\kappa n^2)$ for retrieval phase. Very recently, the cost is further improved to $O(|M|+\kappa n^2)$ for dispersal and $O(|M|+\kappa n)$ for retrieval assuming a trusted setup [4]. Also, without a trusted setup, [27, 4] proposed AVID protocols with $O(|M|+\kappa n^2)$ for dispersal and $O(|M|+\kappa n \log n)$ for retrieval phase. Also, in their AVID protocol, both dispersing and retrieving client incur a cost of $O(|M|+\kappa n \log n)$ during the dispersal and retrieval phase, respectively. Moreover, each node needs incurs a storage cost of $O(|M|/n + \kappa \log n)$. We summarize the existing works on AVID in Table 2 and describe them in more detail in §6.

Our results. Our first contribution is a RBC protocol, which we referred to as the BalRBC, has the same total communication cost of $O(n|M|+\kappa n^2)$ as the state-of-the-art [11], but additionally achieves balanced communication cost of $O(|M|+\kappa n)$ at every node, including the broadcaster. In particular, we eliminate the need for the broadcasters to send the entire proposal message to every node.

Our second contribution is a new AVID protocol that does not require any trusted setup and has a communication cost of $O(|M|+\kappa n^2)$ during the dispersal phase. Moreover, the both dispersing client incurs a cost of $O(|M|+\kappa n)$. Similar to existing AVID protocols [10, 17, 27, 4], during the dispersal phase, the client uses error correction code to encode the message into n symbols and send a symbol to each node. Unlike existing protocols, we do not use a Merkle tree for verification and instead reliably broadcast hashes of each symbol and use them for verification. As a result, each node in our AVID protocol incurs a storage cost of $O(|M|/n + \kappa)$. instead of $O(|M|/n + \kappa \log n)$. Finally, using error-correcting codes and online error

correction, we improve the communication cost of retrieval per client to $O(|M|+\kappa n)$.

Our third contribution is two lower bound results. First, we prove that in any deterministic RBC protocol, each node incurs a communication cost of $\Omega(|M|+n)$. Second, we prove that in any deterministic AVID protocol, the dispersal phase has a communication cost of $\Omega(|M|+n^2)$. Hence, both of our protocols have near-optimal communication costs.

Paper organizations. The rest of the paper is organized as follows. In §2 we provide the necessary background. In §3 we discuss our balanced RBC protocol where the broadcaster has a comparable bandwidth cost as other nodes. In §4 we describe our improved AVID protocol. In §5 we show several lower bounds on the communication cost of RBC and AVID. We discuss the related work in §6 and conclude in §7.

2 System Model and Preliminaries

2.1 System Model

We consider a network of n nodes where every pair of nodes is connected via a pairwise authenticated channel. We consider the presence of a malicious adversary $\mathcal{A}$ that can corrupt up to t nodes in the network. The corrupted nodes can behave arbitrarily, and we call a node honest if it remains non-faulty for the entire protocol execution. We assume the network is asynchronous, i.e., $\mathcal{A}$ can arbitrarily delay any message but must eventually deliver all messages sent between honest nodes.

We use $|S|$ to denote the size of a set S. Let $\mathbb{F}$ be a finite field. For any integer a, we use $[a]$ to denote the ordered set $\{1, 2, \ldots, a\}$. We use κ to denote the size of output of the collision-resistant hash function. We also assume that $\kappa > \log(|\mathbb{F}|) \geq \log n$.

2.2 Problem Formulations

Definition 1 (Reliable Broadcast)**.** A protocol for a set of nodes $\{1, ..., n\}$, where a distinguished node called the broadcaster holds an initial input M, is a reliable broadcast (RBC) protocol, if the following properties hold

- *Agreement:* If an honest node outputs a message M' and another honest node outputs M'', then $M' = M''$.
- *Validity:* If the broadcaster is honest, all honest nodes eventually output the message M.
- *Totality:* If an honest node outputs a message, then every honest node eventually outputs a message.

An VID protocol has following two functions: `DISPERSE`(M), which a client invokes to disperse a message M to n nodes, and `RETRIEVE`, which a (possibly different) client invokes to retrieve the message M. Clients invoke `DISPERSE` and `RETRIEVE` for a particular instance of VID, identified by an instance tag. For simplicity, in the paper we will focus on a single instance of an VID and omit the instance tag.

Definition 2 (Verifiable Information Dispersal [10])**.** A verifiable information dispersal (VID) scheme for a message M consists of a pair of protocols `DISPERSE` and `RETRIEVE` which satisfy the following requirements:

- *Termination:* If an honest client invokes `DISPERSE`(M) and no other client invokes `DISPERSE` on the same instance, then every honest node eventually finishes the dispersal phase.
- *Agreement*: If any honest node finishes the dispersal phase, all honest nodes eventually finish the dispersal phase.
- *Availability*: If an honest node has finished the dispersal phase, and some honest client initiates `RETRIEVE`, then the client eventually reconstructs some message M'.
- *Correctness*: If an honest node has finished the dispersal phase, then honest clients always reconstruct the same message M' when invoking `RETRIEVE`. Furthermore, if an honest client invoked `DISPERSE`(M) and no other client invokes `DISPERSE` on the same instance, then $M' = M$.

In this paper, we will propose protocols solving these two problems under asynchronous networks, and also prove corresponding lower bounds for deterministic protocols.

Definition 3 (Communication Complexity)**.** The communication complexity of a protocol measures the total number of bits sent by all honest protocol nodes during the execution of the protocol.

In addition to the standard communication complexity above which measures the total cost of a protocol, we also measure the cost for each honest protocol node, as the per-node communication complexity defined below.

Definition 4 (Per-node Communication Complexity)**.** The per-node communication complexity of any honest protocol node p running a protocol measures the number of bits sent by p, and the number of bits p received from any other honest node, during the execution of the protocol.

Note that, asymptotically, the communication complexity of a protocol equals the summation of per-node communication complexity over all honest nodes. When a protocol has the same per-node communication complexity C for every honest node, we say the protocol has per-node communication complexity C.

2.3 Primitives

Error Correcting Code We use error correcting codes. For concreteness, we will use the standard Reed-Solomon (RS) codes [25]. A (m, k) RS code in Galois Field $\mathbb{F} = \text{GF}(2^a)$ with $m \leq 2^a - 1$, encodes k data symbols from $\text{GF}(2^a)$ into a codeword of m symbols from $\text{GF}(2^a)$. Let $\mathsf{RSEnc}(M, m, k)$ be the encoding algorithm. Briefly, the RSEnc takes as input a message M consisting of k symbols, treats it as a polynomial of degree $k-1$ and outputs m evaluations of the corresponding polynomial.

Let $\mathsf{RSDec}(k, r, T)$ be the RS decoding procedure. RSDec takes as input a set of symbols T (some of which may be incorrect), and outputs a degree $k-1$ polynomial, i.e., k symbols, by correcting up to r errors (incorrect symbols) in T. It is well-known that RSDec can correct up to r errors in T and output the original message provided that $|T| \geq k + 2r$ [22]. Concrete instantiations of RS codes include the Berlekamp-Welch algorithm [26] and the Gao algorithm [15].

Collision-resistant Hash Function We use a cryptographic collision-resistant hash function hash, which guarantees that a polynomially bounded adversary cannot come up with two inputs that hash to the same value, except for a negligible probability.

3 Balanced Quadratic Reliable Broadcast

We present our balanced reliable broadcast (BalRBC) in Algorithm 1 and describe it next. In our RBC protocol, each node, including the broadcaster, incur the same per-node communication cost of $O(|M|+\kappa n)$. Also, our RBC protocol matches the state-of-the-art RBC protocol in terms of total cost [11].

3.1 Our Design

In order to reduce the cost of the broadcaster node, our protocol BalRBC first let the broadcaster encode its message M into n symbols using a $(n, t+1)$ Reed-Solomon code (line 4), and only send the i-th symbol to node i together with the hash digest of the message M (line 5). Note that due to RS code, each symbol has size $|M|/(t+1)$, and therefore the cost of the broadcaster is reduced to $O(n \cdot (|M|/(t+1)+\kappa)) = O(|M|+\kappa n)$ where κ is the size of the hash digest.

The next step is to let all nodes forward their symbols and the hash digest received from the broadcaster. When a node receives a symbol from other nodes, it adds the symbol to a set T_h indexed by the hash digest h. Once enough symbols are collected, nodes use the standard *Online Error Correcting* (OEC) algorithm [5] (line 11-14) to decode the message. Intuitively, the OEC algorithm performs up to t trials of reconstruction, and during the r-th trial, a node uses $2t + r + 1$ symbols to decode. If the reconstructed message M' has

Algorithm 1 BalRBC protocol for long messages

1: *// only broadcaster node*
2: **input** M
3: Let $h := \mathsf{hash}(M)$
4: Let $[m_1, m_2, \ldots, m_n] := \mathsf{RSEnc}(M, n, t+1)$
5: **send** $\langle \texttt{PROPOSE}, m_j, h \rangle$ to node j for each $j \in [n]$

// each node i
6: **input** $P(\cdot)$ *// predicate $P(\cdot)$ returns true unless otherwise specified.*

7: Let $M := \perp$
8: **upon** receiving the first $\langle \texttt{PROPOSE}, m_i, h \rangle$ from the broadcaster **do**
9: **send** $\langle \texttt{SHARE}, m_i, h \rangle$ to all nodes

10: For the first $\langle \texttt{SHARE}, m_j^*, h \rangle$ received from node j, add (j, m_j^*) to T_h *// T_h initialized as* $\{\}$
11: **for** $0 \leq r \leq t$ **do** *// online Error Correction*
12: **upon** $|T_h| \geq 2t + r + 1$ **do**
13: Let $M' := \mathsf{RSDec}(t+1, r, T_h)$
14: **if** $\mathsf{hash}(M') = h$ and $P(M') = true$ **then**
15: $M := M'$
16: **send** $\langle \texttt{ECHO}, m_j, h \rangle$ to node j for each $j \in [n]$ where m_j is the j-th symbol of $\mathsf{RSEnc}(M', n, t+1)$

17: **upon** receiving $2t + 1$ $\langle \texttt{ECHO}, m_i, h \rangle$ for matching messages and not having sent a `READY` message **do**
18: **send** $\langle \texttt{READY}, m_i, h \rangle$ to all

19: **upon** receiving $t + 1$ $\langle \texttt{READY}, *, h \rangle$ messages and not having sent a `READY` message **do**
20: Wait for $t + 1$ matching $\langle \texttt{ECHO}, m_i', h \rangle$
21: **send** $\langle \texttt{READY}, m_i', h \rangle$ to all

22: For the first $\langle \texttt{READY}, m_j^*, h \rangle$ received from node j, add (j, m_j^*) to T_h' *// T_h' initialized as* $\{\}$
23: **for** $0 \leq r \leq t$ **do** *// Error Correction*
24: **upon** $|T_h'| \geq 2t + r + 1$ **do**
25: **if** $M \neq \perp$ **then**
26: **output** M and **return**
27: Let $M'' := \mathsf{RSDec}(t+1, r, T_h')$
28: **if** $\mathsf{hash}(M'') = h$ **then**
29: **output** M'' and **return**

the matching hash digest h, a node successfully reconstructs the message; otherwise, it waits for one more symbol and tries again.

Once a node successfully reconstructs the message M', the rest of the protocol is similar to the four-round RBC of Das et al. [11, Algorithm 4]. Briefly, nodes send `ECHO` messages with their symbols and the hash digest to all nodes (line 16). Also, nodes send `READY` messages once $2t + 1$ matching `ECHO` messages are collected (line 17-18) or upon receiving $t + 1$ `READY` messages (line 19-21). Note that each node needs to wait for $t+1$ matching `ECHO` messages to learn the symbol to be attached in the `READY` message (line 20-21). Finally, nodes use the OEC algorithm to reconstruct the broadcaster's message once receiving enough `READY` messages of the same hash digest.

3.2 Analysis

We next analyze the properties of our BalRBC protocol and its performance.

Lemma 1. *Assuming a collision resistant hash function, if an honest node sends* $\langle \texttt{READY}, m_i, h\rangle$ *where* $h = \mathsf{hash}(M)$*, then* m_i *is the* i^{th} *symbol of* $\mathsf{RSEnc}(M, n, t+1)$*, and furthermore, no honest node sends a* `READY` *message for a different* $h' \neq h$*.*

Proof. First we show no two honest nodes send `READY` messages for different hash digests. Let i be the first honest node that sends a $\langle \texttt{READY}, *, h\rangle$. Then at least $2t+1$ nodes sent $\langle \texttt{ECHO}, *, h\rangle$ to node i. Now, for the sake of contradiction assume that an honest node i' is the first honest node that sends a $\langle \texttt{READY}, *, h'\rangle$ for $h' \neq h$. Again at least $2t+1$ nodes sent $\langle \texttt{ECHO}, *, h'\rangle$ to node i'. Then, by quorum intersection, at least $t+1$ nodes sent `ECHO` message for both h and h'. This is impossible as there are at most t malicious nodes and an honest node sends `ECHO` message at most once.

Now we show if an honest node sends $\langle \texttt{READY}, m_i, h\rangle$ where $h = \mathsf{hash}(M)$, then m_i is the i^{th} symbol of $\mathsf{RSEnc}(M, n, t+1)$. Note that an honest node i sends $\langle \texttt{READY}, m_i, h\rangle$ for $h = \mathsf{hash}(M)$ only upon receiving at least $t+1$ matching $\langle \texttt{ECHO}, m_i, h\rangle$. At least one of these `ECHO` message is from an honest node h. Before the honest node h sends the `ECHO` message, it successfully reconstructed the message M' whose hash digest equals h. Then, by the collision resistance property of the underlying hash function, $M' = M$ and m_i is the i^{th} symbol of $\mathsf{RSEnc}(M, n, t+1)$. □

Lemma 2. *If an honest node* i *receives* $t+1$ `READY` *messages with a matching hash* h*, then node* i *will eventually receive* $t+1$ *matching* $\langle \texttt{ECHO}, m_i, h\rangle$ *messages and hence send* $\langle \texttt{READY}, m_i, h\rangle$*.*

Proof. Let j be the first honest node that sends $\langle \texttt{READY}, *, h\rangle$ message to all. Then, node j must have received at least $2t+1$ `ECHO` messages with matching h. At least $t+1$ of these `ECHO` messages are from honest nodes. All these honest node will send $\langle \texttt{ECHO}, m_i, h\rangle$ to node i. Hence, node i will eventually receive $t+1$ $\langle \texttt{ECHO}, m_i, h\rangle$ messages. □

Theorem 1 (Totality and Agreement)**.** *If an honest node outputs a message, then every honest node eventually outputs a message. If an honest node outputs a message* M' *and another honest node outputs* M''*, then* $M' = M''$*.*

Proof. An honest node outputs a message M only upon receiving at least $2t+1$ `READY` messages with a matching hash $h = \mathsf{hash}(M)$. At least $t+1$ of them are sent by an honest node. Hence, all honest nodes will receive at least $t+1$ `READY` messages with hash h. By lemma 2, eventually all honest nodes will send `READY` messages with hash h. Hence, all honest nodes will receive `READY` messages from all other honest nodes. Furthermore, due to Lemma 1, all these `READY` message contain correct symbols from the codeword $\mathsf{RSEnc}(M, n, t+1)$. Thus, every honest node will eventually output M such that $h = \mathsf{hash}(M)$. □

Theorem 2 (Validity)**.** *If the broadcaster is honest, all honest nodes eventually output the message* M*, given that the predicate* $P(\cdot)$ *evaluates to be true at all honest nodes.*

Proof. When the broadcaster is honest and has input M, it sends the correct symbols and hash to all nodes. Then, all honest nodes send the `SHARE` messages with the correct symbols. Thus, after receiving all `SHARE` message from honest nodes, any honest node can reconstruct M' due to OEC. Moreover, the collision resistance property of the hash function ensures that $M' = M$. Also, the predicate $P(M') = \mathit{true}$ at all honest nodes, therefore, at least $2t+1$ honest nodes will send `ECHO` messages with identical $h = \mathsf{hash}(M)$. Hence, all honest nodes will eventually send `READY` messages for h. By lemma 1 no honest node will send `READY` message for $h' \neq h$. As a result, all honest node will receive at least $2t+1$ `READY` message for h with valid symbols in it, which is sufficient to recover M. □

Next, we will analyze the communication complexity of the protocol.

Theorem 3 (Performance)**.** *Assuming existence of collision resistant hash functions, Algorithm 1 solves RBC with total communication complexity of* $O(n|M|+\kappa n^2)$*, and per-node communication complexity of* $O(|M|+\kappa n)$*, where* κ *is the size of the output of the hash function.*

Proof. In algorithm 1 the broadcaster sends a single `PROPOSE` to all other nodes. Moreover, each honest node sends a single `SHARE`, `ECHO` and `READY` message. Each message in Algorithm 1 is $O(|M|/n+\kappa)$ bits long, since $|m_i|= |M|/(t+1)$ and hash outputs are κ bits long. Hence, each node incurs a per-node communication cost of $O(|M|+n\kappa)$. Hence, the total communication cost is $O(n|M|+\kappa n^2)$. □

4 Improved Asynchronous Verifiable Information Dispersal

We next describe our improved AVID protocol and summarize it in Algorithm 2. As mentioned in §2, the protocol consists of two phases: *dispersal* phase, invoked by the dispersing client who wants to reliably store its message M among n protocol nodes; and *retrieval* phase, invoked by any retrieving client (need not be the one who initiated the dispersal phase), who wants to retrieve the message M.

4.1 Our Design

In the dispersal phase, first, the dispersing client encodes the message M using a $(n, t+1)$ Reed-Solomon code (line 2). Let $M' = [m_1, m_2, \ldots, m_n]$ be the encoded message that consists of n symbols $m_1, m_2, \ldots, m_n$. The dispersing client then computes a vector $H = [h_1, h_2, \ldots, h_n]$ of n elements where the i-th element $h_i = \mathsf{hash}(m_i)$ (line 3). The dispersing client then sends the i-th symbol m_i to i-th node. Additionally, the dispersing client reliably broadcasts H using our BalRBC protocol from §3 (line 4). During the BalRBC, as per the predicate, the i-th node checks whether the i-th element of the hash vector that is being reliably broadcast, is equal to the hash of the symbol it received from the dispersing client (line 5-7). Let H be the output of the validated RBC. Each node then encodes H using a $(n, t+1)$ Reed-Solomon code; let $H' = [h'_1, h'_2, \ldots, h'_n]$ be the output of the Reed-Solomon encoding (line 11). Also, let $h = \mathsf{hash}(H)$. At the end of the dispersal phase, the i-th node outputs $\langle m_i, h'_i, h\rangle$ where $m_i = \perp$ if the i-th node did not receive a valid symbol from the dispersing client.

The main idea of the retrieval phase is to let the retrieving client first recover the vector H and then use it to validate symbols sent by nodes. More specifically, during the retrieval phase, the retrieving client sends `RETRIEVE` request to all nodes (line 13). Upon receiving `RETRIEVE` request from the retrieving client, each node waits till the dispersal phase terminates (line 33). The i-th node then sends the message $\langle \texttt{HASH}, h'_i, h\rangle$ to the retrieving client (line 34). Additionally, if node i received a symbol m_i during the dispersal phase such that $\mathsf{hash}(m_i) = H[i]$, it sends a $\langle \texttt{SYMBOL}, m_i\rangle$ to the retrieving client (line 35-36).

The retrieving client upon receiving a message $\langle \texttt{HASH}, h_j, h\rangle$ from node j adds them to a set T_h (line 15). Additionally, for every $\langle \texttt{SYMBOL}, m_j\rangle$ message it receives, it adds it to the set T_M. The retrieving client then uses T_h and the standard online error correction to recover H (line 17-21). After recovering H, the retrieving client uses it to retrieve the message. In particular, for every tuple $(j, a) \in T_M$, it first checks whether $\mathsf{hash}(a) = H[j]$ and adds the tuple (j, a) to the set T (line 22-24).

The retrieving client waits till $|T|= t+1$ and then interpolates the tuples in T into a polynomial of degree less than or equal to t (line 25-26). Let M' be the interpolated polynomial. The client then checks if there exists any $j \in [n]$ such that $M[j] \neq H[j]$. If such j exists, then the client outputs $\perp$ and returns. Otherwise, the outputs the Reed-Solomon decoding of M' (line 27-31).

4.2 Analysis

We next analyze the properties of our AVID protocol and its performance.

Theorem 4 (Termination and Agreement)**.** *If an honest dispersing client invokes* `DISPERSE`(M) *and no other client invokes* `DISPERSE` *on the same instance, then every honest node eventually finishes the dispersal phase.*

If any honest node finishes the dispersal phase, all honest nodes eventually finish the dispersal phase.

Proof. An honest dispersing client sends the correct symbols $m_1, ..., m_n$ to all nodes where $[m_1, m_2, \ldots, m_n] = \mathsf{RSEnc}(M, n, t+1)$, and reliab ly broadcasts the hash vector $H = [\mathsf{hash}(m_1), \mathsf{hash}(m_2), \ldots, \mathsf{hash}(m_n)]$. By

Algorithm 2 Pseudocode for AVID

// the dispersing client invokes `DISPERSE`*(M)*
input M
Let $M' := [m_1, m_2, \ldots, m_n] := \mathsf{RSEnc}(M, n, t+1)$
Let $H := [h_1, h_2, \ldots, h_n] := [\mathsf{hash}(m_1), \mathsf{hash}(m_2), \ldots, \mathsf{hash}(m_n)]$
Send m_i to node i for each $i \in [n]$, and invoke BalRBC(H) with predicate $P(\cdot)$ described below

// predicate $P(\cdot)$ for node i during BalRBC
procedure $P(H)$
    upon receiving m_i from the dispersing client **do**
        return true iff $\mathsf{hash}(m_i) = H[i]$

// code for node i during the dispersal phase
Wait till BalRBC($\cdot$) terminate
Let $H = [h_1, h_2, \ldots, h_n]$ be the output of BalRBC($\cdot$)
Let $h = \mathsf{hash}(H)$
$[h'_1, h'_2, \ldots, h'_n] := \mathsf{RSEnc}(H, n, t+1)$
Output and store $\langle m_i, h'_i, h \rangle$ for the dispersal phase *// Use $m_i = \perp$ if it does not receive m_i with $h_i = \mathsf{hash}(m_i)$ from the client.*

// the retrieving client invokes `RETRIEVE`
send $\langle$`RETRIEVE`$\rangle$ to all nodes

// retrieving H
Let $T_h := \{\}; T_M := \{\}$
For every $\langle$`HASH`$, h'_j, h\rangle$ received from node j, add (j, h'_j) to T_h
For every $\langle$`SYMBOL`$, m_j\rangle$ received from node j, add (j, m_j) to T_M
for $0 \le r \le t$ **do** *// online Error Correction*
    Wait till $|T_h| \ge 2t + r + 1$
    Let $H = \mathsf{RSDec}(t+1, r, T_h)$
    if $\mathsf{hash}(H) = h$ **then**
        break

// retrieving M after retrieving H
for each $(j, a) \in T_M$ **do**
    if $\mathsf{hash}(a) = H[j]$ **then**
        add (j, a) to T
Wait till $|T| = t + 1$
Interpolate T as a degree-t polynomial
Let M' be the interpolated polynomial evaluated at every element in $[n]$
if $\exists j \in [n]$ such that $\mathsf{hash}(M'[j]) \neq H[j]$ **then**
    output $\perp$ and **return**
else
    output $\mathsf{RSDec}(t+1, 0, M')$ and **return**

// code for node i during `RETRIEVE`
upon receiving $\langle$`RETRIEVE`$\rangle$ from the retrieving client **do**
    Wait till the dispersal phase outputs $\langle m_i, h'_i, h\rangle$
    send $\langle$`HASH`$, h'_i, h\rangle$ to the retrieving client
    if $m_i \neq \perp$ **then**
        send $\langle$`SYMBOL`$, m_i\rangle$ to the retrieving client

the Validity property of the RBC, the RBC will terminate at all honest nodes. Hence, every honest node will finish the dispersal phase.

Nodes terminate the dispersal phase only after the RBC protocol terminates. Thus by the Totality property of the RBC every node will terminates the RBC and thus terminates the dispersal phase. □

Lemma 3. *If the dispersal phase terminates at an honest node, then every honest node will output the same vector of hashes $H = [h_1, h_2, \ldots, h_n]$ for RBC. Furthermore, at least $t + 1$ honest nodes have received a symbol that matches with the corresponding location of H.*

Proof. Nodes terminate the dispersal phase only after the RBC protocol terminates. Thus by the Totality and Agreement property of the RBC every node will receive the same message H. Furthermore, when any honest node finishes the RBC, it has received `READY` messages from at least $2t + 1$ nodes, among which at least $t+1$ are honest. This implies that at least one honest node receives `ECHO` messages from at least $2t+1$ nodes, among which at least $t + 1$ are honest. Before these honest nodes send `ECHO` messages, they have the predicate evaluated to be true, which implies that each honest node j above has received m_j from the client such that $\mathsf{hash}(m_j) = H[j]$. □

Lemma 4. *If an honest node has finished the dispersal phase with H as the output of the RBC, then any honest client can reconstruct the same H after invoking* `RETRIEVE`.

Proof. By Agreement of AVID, all honest nodes eventually finish the dispersal phase once an honest node has finished the dispersal phase. Also, due to Lemma 3 all honest nodes output the same H for RBC when the dispersal phase terminates. Therefore, when an honest client invokes `RETRIEVE`, all $2t + 1$ honest nodes will send the correct $\langle \texttt{HASH}, h'_i, \mathsf{hash}(H)\rangle$ to the client. By OEC (line 17-20) and the collision resistance property of the hash function, the client can successfully decode the same hash vector H. □

Theorem 5 (Availability and Correctness)**.** *If an honest node has finished the dispersal phase, and some honest clients invoke* `RETRIEVE`*, then they eventually output the same message M'. Furthermore, if an honest client invoked* `DISPERSE`(M) *and no other client invokes* `DISPERSE` *on the same instance, then $M' = M$.*

Proof. By Lemma 4, the honest client reconstructs the same hash vector H as the one output by any honest node during the dispersal phase. Moreover, at least $t + 1$ honest nodes have received m_j such that $\mathsf{hash}(m_j) = H[j]$. Therefore, when an honest client invokes `RETRIEVE`, all these $t + 1$ honest nodes will send the correct $\langle \texttt{SYMBOL}, m_i\rangle$ where $\mathsf{hash}(m_i) = H[i]$ to the client. As a result, the client can reconstruct the message M' using these symbols.

Let $f_u(\cdot)$ denote the polynomial of degree t or less a retrieving client u obtains via interpolation. The client u then uses $f_u(\cdot)$ to recover the message M_u only if $\mathsf{hash}(f_u(i)) = H[i]$ for all $i \in [n]$. Hence, due to collision resistance property of the $\mathsf{hash}(\cdot)$, if two retrieving client, u and v outputs messages $M_u \neq \perp$ and $M_v \neq \perp$, respectively, then $M_u = M_v$.

Also, if any honest retrieving client outputs $\perp$, then every honest retrieving client outputs $\perp$. For the sake of contradiction, let us assume that a retrieving client u outputs $\perp$ but another retrieving client $v \neq u$ outputs $M_v \neq \perp$. Let T_u be the set of indices used by retrieving client u to interpolate f_u. Then, there exists a $k \in [n] \setminus T_u$ such that $H[k] \neq \mathsf{hash}(f_u[k])$. Since retrieving client v outputs $M_v \neq \perp$, this implies $f_u[k] = f_v[k]$ for all $k \in T_u$. However, both f_u and f_v have degree t or less and agrees on $t+1$ distinct points. This implies f_u and f_v matches as polynomial and $f_u[k] = f_v[k]$ for all $k \in [n]$, which is a contradiction.

If an honest dispersing client invoked `DISPERSE`(M) and no other dispersing client invokes `DISPERSE` on the same instance, by the Termination property of AVID, all honest nodes eventually finish the dispersal phase with RBC output $H = [\mathsf{hash}(m_1), \ldots, \mathsf{hash}(m_n)]$ where $[m_1, \ldots, m_n] = \mathsf{RSEnc}(M, n, t + 1)$. Also, from Lemma 4 the retrieving client will receive H during the retrieval phase. Then, by collision-resistant property of the hash function, the honest retrieving client use correct symbol to recover the message, hence will recover and output M. □

Theorem 6 (Performance)**.** *The communication cost of a dispersing client during the dispersal phase is $O(|M|+\kappa n)$ and the total communication cost of the dispersal phase is $O(|M|+\kappa n^2)$. Also, each node incurs*

a storage cost of $O(|M|/n + \kappa)$. *Furthermore, the total communication cost for retrieval at a client is* $O(|M|+\kappa n)$.

Proof. During the dispersal phase, the dispersing client only sends a symbol of size $O(|M|/n)$ to each node and reliably broadcasts a message of size κn. Hence, the total communication cost of the dispersing client is $n \cdot O(|M|/n)$ and the cost of RBC which is $O(\kappa n)$ from Lemma 3. Also, each node receives a symbol of size $O(|M|/n)$ and participates in the RBC of a message of size $O(\kappa n)$. Hence, using Lemma 3, the total communication cost of the dispersal phase is $O(|M|+\kappa n^2)$. At the end of the dispersal phase, each node stores two symbol of size $O(|M|/n)$ and $O(\kappa)$, respectively, and a hash output of size κ. Thus, the total storage cost of our AVID protocol is $O(|M|+\kappa n)$.

During retrieval at any retrieving client, each node sends at most two symbols of size $O(\kappa)$ and $O(|M|/n)$ to the retrieving client. Hence, the communication cost of the retrieval at a single retrieving client is $O(|M|+\kappa n)$. □

5 Lower Bounds

In this section, we prove communication complexity lower bounds for deterministic protocols that solve RBC and AVID, as mentioned in Table 1 and 2. To strengthen the result, the lower bounds for RBC are proven under synchrony. The lower bound proofs are inspired by [13].

5.1 Reliable Broadcast

For any deterministic RBC protocol with input M and tolerates up to $\Theta(n)$ Byzantine nodes, it is straightforward to show a lower bound of $\Omega(n|M|+n^2)$ [24] on the communication cost even under synchrony. The $\Omega(n|M|)$ part is because $O(n)$ honest nodes need to receive the message when the protocol terminates, and the $\Omega(n^2)$ part is due to the classic Dolev-Reischuk lower bound [13]. Therefore, both Das et al. [11] and our BalRBC have near-optimal communication cost.

Next, for any deterministic protocol that solves RBC under synchrony, we will show that $\Omega(|M|+n)$ is a lower bound on the communication cost of any protocol node including the broadcaster, which implies our BalRBC has near-optimal per-node cost as well. The argument for broadcaster is straightforward. First, the broadcaster needs to send at least $\Omega(|M|)$ bits for its input message M. Moreover, the broadcaster has to send messages to at least $t + 1$ nodes, otherwise it is possible that no honest nodes receive any information from the broadcaster, and the Validity property of RBC can be violated. Since $t = \Theta(n)$, we conclude that the broadcaster has to send $\Omega(|M|+n)$ bits. For any non-broadcaster node, we show the following lower bound result.

Theorem 7. *Any deterministic protocol that solves RBC must incur a per-node communication cost of* $\Omega(|M|+n)$ *for any honest node.*

Proof. We will prove that any deterministic RBC protocol incurs at least $\Omega(|M|+n)$ per-node communication cost in at least one execution.

The theorem is true for broadcaster node as mentioned previously, thus we will only prove for non-broadcaster node. Consider any non-broadcaster honest node during a failure-free execution where the broadcaster has input M. Since the honest node needs to output M, at least $\Omega(|M|)$ bits needs to be received.

Let $C_{p,E}$ denote the number of messages an honest node p sends to any node and receives from any honest node during an execution E. We show that $C_{p,E} \geq t/2 + 1$ for any honest node p in at least one execution E. Otherwise, suppose there exists a RBC protocol where an honest node q has $C_{q,E} \leq t/2$ for any execution E. Without loss of generality, suppose q outputs 0 if it receives no message during the entire execution but other honest nodes output for RBC. Consider a failure-free execution $E1$ where the honest broadcaster has input 1, and by assumption, $C_{q,E1} \leq t/2$. Let S denote the set of nodes that q receives messages from in $E1$, and $|S| \leq t/2$. Consider execution $E2$ where the honest broadcaster has input 1, and

q is Byzantine and remains silent. Since the broadcaster is honest and has input 1, by Validity, all honest nodes output 1 in $E2$. Then, we construct another execution $E3$ same as $E2$ except that the nodes in S are Byzantine and q is now honest. The nodes in S behave identically as in $E2$, except that they send no message to q. By assumption, $C_{q,E3} \leq t/2$, and the adversary also corrupts the set of nodes R that q sends messages to in $E3$. It is possible since $|S \cup R| \leq |S| + |R| \leq t$. The Byzantine nodes in R behave identically as in $E2$. Since q receives no message in $E3$, q will output 0 in $E3$ by assumption. Other honest nodes will output 1 in $E3$ since they cannot distinguish $E2, E3$. However, the Agreement property of RBC is then violated. Therefore, we prove that $C_{p,E} \geq t/2 + 1$ for any honest node p in at least one execution E, which implies the per-node communication cost of any RBC protocol is $\Omega(n)$.

Therefore, any RBC protocol has a per-node communication cost of $\Omega(|M|+n)$. □

5.2 Asynchronous Verifiable Information Dispersal

For AVID (or even synchronous VID), the communication cost of the dispersing client during dispersal phase is lower bounded by $\Omega(|M|+n)$ by a similar argument – the dispersing client needs to send $\Omega(|M|)$ bits and needs to send messages to at least $t + 1 = \Omega(n)$ nodes. For the total communication cost during dispersal, we will show the following $\Omega(|M|+n^2)$ lower bound for AVID.

Theorem 8. *Any deterministic protocol that solves AVID must incur a total communication cost of* $\Omega(|M|+n^2)$ *during the dispersal phase.*

Proof. Using a similar argument as the cost of broadcaster node in proof of Theorem 7, the dispersing client incurs a cost of $\Omega(|M|+n)$ during the dispersal phase, which is also a lower bound for total cost. For the protocol nodes, we will prove that any deterministic AVID protocol incurs at least $\Omega(n)$ per-node communication cost during the dispersal phase in at least one execution. Then, the total cost of dispersal phase is $\Omega(|M|+n^2)$ since there are n protocol nodes.

Similar to the proof of Theorem 7, let $C_{p,E}$ denote the number of messages an honest node p sends to any node and receives from any honest node during the dispersal phase of an execution E, and we will show that $C_{p,E} \geq t/2+1$ for any honest node p in dispersal phase of at least one execution E. Otherwise, suppose there exists an AVID protocol where an honest node q has $C_{q,E} \leq t/2$ during the dispersal phase for all executions. There are two cases.

- Suppose q terminates the dispersal phase if it receives no message during the dispersal phase. Consider a failure-free execution $E1$ where the honest dispersing client disperses M but its messages are delayed, and by assumption, $C_{q,E1} \leq t/2$. Let S denote the set of nodes that q receives messages from in $E1$, and $|S| \leq t/2$. Then, we construct another execution $E2$ where the dispersing client is Byzantine and remain silent, and node q is also Byzantine and remain silent. Now consider execution $E3$ where the honest dispersing client disperses a message M but its messages are delayed, the nodes in S are Byzantine and q is honest. The nodes in S behave identically as in $E2$, except that they send no message to q. By assumption, $C_{q,E3} \leq t/2$, and the adversary also corrupts the set of nodes R that q sends messages to in $E3$. It is possible since $|S \cup R| \leq |S| + |R| \leq t$. The Byzantine nodes in R behave identically as in $E2$. Since q receives no message in $E3$, q will terminate the dispersal phase in $E3$ by assumption. For other honest nodes, they cannot distinguish $E2, E3$ before receiving any message from the dispersing client. Therefore, by Agreement property of AVID, these honest nodes will terminate the dispersal phase in $E3$ before receiving any message from the dispersing client. In the retrieval phase of $E3$, according to the Availability property, the retrieving client reconstructs some message eventually at some time point τ. Suppose the messages of the dispersing client is delayed beyond time τ in $E3$. Since no honest node receives any message from the dispersing client, during the retrieval phase of $E3$ the dispersed message cannot be reconstructed, violating the Correctness property of AVID.

- Suppose q never terminates the dispersal phase if it receives no message during the dispersal phase. Consider a failure-free execution $E1$ where the honest dispersing client disperses M, and by assumption, $C_{q,E1} \leq t/2$. Let S denote the set of nodes that q receives messages from in $E1$, and $|S| \leq t/2$. Consider

execution $E2$ same as $E1$ except that q is Byzantine and remains silent. Since the dispersing client is honest, all honest nodes eventually terminates the dispersal phase in $E2$. Then, we construct another execution $E3$ same as $E2$ except that the nodes in S are Byzantine and q is now honest. The nodes in S behave identically as in $E2$, except that they send no message to q. By assumption, $C_{q,E3} \leq t/2$, and the adversary also corrupt the set of nodes R that q sends messages to in $E3$. It is possible since $|S \cup R| \leq |S| + |R| \leq t$. The Byzantine nodes in R behave identically as in $E2$. Since q receives no message in $E3$, q will never terminate the dispersal phase in $E3$ by assumption. However, other honest nodes will terminate the dispersal phase in $E3$ since they cannot distinguish $E2, E3$. Then, the Agreement property of RBC is violated, contradiction.

Therefore, $C_{p,E} \geq t/2 + 1$ for any honest node p in at least one execution E, which implies the per-node communication cost of any AVID protocol during dispersal is $\Omega(n)$. Hence, the communication cost of any deterministic AVID protocol is $\Omega(|M| + n^2)$ during the dispersal phase. □

Finally, the total communication cost during retrieval is lower bounded by $\Omega(|M| + n)$, since the client needs to receive at least $\Omega(|M|)$ bits from the honest nodes to obtain M, and the client needs to send messages to at least $t + 1 = \Omega(n)$ nodes for retrieval otherwise it could be the t nodes are Byzantine and ignore the message.

6 Related Work

Reliable Broadcast. The problem of reliable broadcast (RBC) was introduced by Bracha [9]. In the same paper, Bracha provided an RBC protocol for a single bit with a communication cost of $O(n^2)$, thus $O(n^2|M|)$ for $|M|$ bits using a naïve approach. Almost two decades later, Cachin and Tessaro [10] improved the cost to $O(n|M| + \kappa n^2 \log n)$ assuming a collision-resistant hash function with κ being the output size of the hash. Hendricks et al. in [17] propose an alternate RBC protocol with a communication cost of $O(n|M| + \kappa n^3)$ using a erasure coding scheme where each element of a codeword can be verified for correctness. Assuming a trusted setup phase, hardness of q-SDH [6, 7] and Decisional Bilinear Diffie-Hellman (DBDH) [8], Nayak et al. [24] reduced the communication cost to $O(n|M| + \kappa n^2)$.

Recently, Das et al. [11] presents a RBC protocol that has a communication cost to $O(n|M| + \kappa n^2)$ assuming only collision-resistant hash function. However, in their RBC protocol, the broadcaster incurs a higher communication cost than the non-broadcaster nodes. Our protocol maintains the same total communication cost while ensuring that each node, including the broadcaster, incurs asymptotically the same communication cost.

Asynchronous Verifiable Information Dispersal. The first AVID protocol is due to Cachin and Tessaro [10]. In the AVID protocol of [10], during the dispersal phase, each node, including the dispersing client, incurs a communication cost of $O(|M| + \kappa n \log n)$. As a result, it has a total dispersal cost of $O(n|M| + \kappa n^2 \log n)$. The high communication cost of each node is because every node needs to gossip a symbol and associated Merkle path during the dispersal phase. In our AVID protocol, we reduce the communication cost of both dispersing client and other nodes by removing the need for nodes to gossip the symbols and associated Merkle paths. Finally, during retrieval at each retrieving client, it has a communication cost of $O(|M| + \kappa n \log n)$. Again, the factor of $\log n$ is because nodes need to send the Merkle path to the retrieving client, which we obviate in our design.

Hendricks et al. in [17] propose an alternate AVID protocol where during dispersal, the dispersing client incurs a communication cost of $O(|M| + \kappa n^2)$. They improve the communication cost by proposing a non-interactive verification scheme with fingerprinted cross-checksum. In their protocol, only the dispersing client sends the symbols to all nodes, and the nodes only perform an RBC on the fingerprinted cross-checksum but not the symbols. As a result, the remaining nodes incur a communication cost of $O(\kappa n^2)$. Hence, the total communication cost of their protocol during the dispersal phase is $O(|M| + \kappa n^3)$. Also, the total retrieval cost for a single retrieving client is $O(|M| + \kappa n^2)$, as each node sends a $O(\kappa n)$ size finger-printed cross-checksum and an encoded symbol to the retrieving client.

Very recently, Yang et al. [27] presents a new AVID protocol in which, during the dispersal phase, the dispersing client incurs a communication cost of $O(|M|+\kappa n \log n)$. Furthermore, the total communication cost of their dispersal phase is $O(|M|+\kappa n^2)$. The main innovation of the AVID protocol of [27] is that they remove the need for nodes to gossip symbols and Merkle paths during the dispersal phase. They do so by designing a novel retrieval protocol and a RBC on the root of the associated Merkle tree. Nevertheless, during the dispersal phase, the dispersing client still needs to send a Merkle path to every node. Moreover, during retrieval at a retrieving client, each node still sends an encoded symbol and the associated Merkle path to the retrieving client, leading to a communication cost of $O(|M|+\kappa n \log n)$. Our protocol improves the communication costs of both these steps by a factor of $\log n$ using a vector of hashes instead of a Merkle tree and our balanced RBC protocol for long messages.

With trusted setup and assuming hardness of q-SDH [18], the recent work by Alhaddad et al. [4] reduced the dispersing client cost to $O(|M|+\kappa n)$ and the total communication to $O(|M|+\kappa n^2)$ using the KZG [18] polynomial commitment scheme. Our protocol achieves the same cost using only collision-resistant hash function without any trusted setup or additional cryptographic assumptions other than collision-resistant hash functions.

7 Discussion and Conclusion

We have presented an asynchronous Byzantine reliable broadcast (RBC) protocol with a balanced communication cost at all nodes, including the broadcaster. Our balanced RBC protocol immediately implies a balanced communication cost of asynchronous verifiable/complete secret sharing schemes [11]. We also present an asynchronous verifiable information dispersal (AVID) protocol with improved communication and storage cost. Finally, we present lower bound results on the per-node communication cost of any deterministic RBC protocol and the total communication cost of dispersal phase in any deterministic AVID protocol. Our balanced RBC protocol and AVID protocol have near-optimal communication costs. One interesting open problem is to design an AVID protocol that also achieves optimal or near-optimal total storage cost.

References

[1] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, page 363–373, 2021.

[2] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: A complete categorization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC)*, page 331–341, 2021.

[3] Ittai Abraham, Ling Ren, and Zhuolun Xiang. Good-case and bad-case latency of unauthenticated byzantine broadcast: A complete categorization. In *International Conference on Principles of Distributed Systems (OPODIS)*, 2021.

[4] Nicolas Alhaddad, Sisi Duan, Mayank Varia, and Haibin Zhang. Succinct erasure coding proof systems. *Cryptology ePrint Archive*, 2021.

[5] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 52–61, 1993.

[6] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *International conference on the theory and applications of cryptographic techniques*, pages 56–73. Springer, 2004.

[7] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the sdh assumption in bilinear groups. *Journal of cryptology*, 21(2):149–177, 2008.

[8] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *International conference on the theory and application of cryptology and information security*, pages 514–532. Springer, 2001.

[9] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[10] Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pages 191–201. IEEE, 2005.

[11] Sourav Das, Zhuolun Xiang, and Ling Ren. Asynchronous data dissemination and its applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2721, 2021.

[12] Sourav Das, Tom Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. *Cryptology ePrint Archive*, 2021.

[13] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.

[14] Sisi Duan, Michael K Reiter, and Haibin Zhang. Beat: Asynchronous bft made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2028–2041, 2018.

[15] Shuhong Gao. A new algorithm for decoding reed-solomon codes. In *Communications, information and network security*, pages 55–68. Springer, 2003.

[16] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous bft protocols. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 803–818, 2020.

[17] James Hendricks, Gregory R Ganger, and Michael K Reiter. Verifying distributed erasure-coded data. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 139–146, 2007.

[18] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *International conference on the theory and application of cryptology and information security*, pages 177–194. Springer, 2010.

[19] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1751–1767, 2020.

[20] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 887–903, 2019.

[21] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 129–138, 2020.

[22] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes*, volume 16. Elsevier, 1977.

[23] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42, 2016.

[24] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H Vaidya, and Zhuolun Xiang. Improved extension protocols for byzantine broadcast and agreement. In *34th International Symposium on Distributed Computing (DISC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[25] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[26] Lloyd R Welch and Elwyn R Berlekamp. Error correction for algebraic block codes, December 30 1986. US Patent 4,633,470.

[27] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. Dispersedledger: High-throughput byzantine consensus on variable bandwidth networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022.

[28] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew Miller. hbacss: How to robustly share many secrets. In *(To appear) Proceedings of the 29th Annual Network and Distributed System Security Symposium*, 2022.