# All-to-All Encode in Synchronous Systems 

Canran Wang and Netanel Raviv<br>Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis 63130, MO, USA


#### Abstract

We define all-to-all encode, a collective communication operation serving as a primitive in decentralized computation and storage systems. Consider a scenario where every processor initially has a data packet and requires a linear combination of all data packets; the linear combinations are distinct from one processor to another, and are specified by a generator matrix of an error correcting code. We use a linear network model, in which processors transmit linear combinations of their data and previously received packets, and adopt a standard synchronous system setting to analyze its communication cost. We provide a universal algorithm which computes any matrix in this model by only varying intermediate coefficients, and prove its optimality. When the generator matrix is of the Vandermonde or Lagrange type, we further optimize the communication efficiency of the proposed algorithm.


## I. Introduction

The interest in coding for decentralized systems has increased lately, due to emerging applications in blockchains [13], sensor networks [4], and the internet of things [5]. In such systems, raw data is generated independently in distributed source nodes, and then encoded and delivered to distributed sink nodes, without a central authority which orchestrates the operation. Examples include encoding for reliable distributed storage (e.g., with a Reed-Solomon or a random code [6]) or for straggler-resilient distributed computation [7]. To study the communication cost of this setting, we focus on the following fundamental collective communication operation.

Definition 1. (All-to-all encode) Consider a distributed system with $K$ processors and no master processor. Every processor $k \in[0, K-1] \triangleq\{0,1, \ldots, K-1\}$ initially possess an initial packet $x_{k} \in \mathbb{F}_{q}$, where $\mathbb{F}_{q}$ is a finite field with $q$ elements, and obtains a coded packet $\widetilde{x}_{k} \in \mathbb{F}_{q}$ after the communication operation. For $A \in \mathbb{F}_{q}^{K \times K}$ that is known a priori to all processors, the coded packets are defined as

$$
\left(\widetilde{x}_{0}, \ldots, \widetilde{x}_{K-1}\right)=\left(x_{0}, \ldots, x_{K-1}\right) \cdot A
$$

That is, in an all-to-all encode operation, every node has its own data packet, and wishes to obtain a linear combination of all other packets in the system. An algorithm which successfully achieves all-to-all encode for a given $A$ and every $x_{0}, \ldots, x_{K-1} \in \mathbb{F}_{q}$ is said to compute $A$. Computing matrices in this context emerges during the encoding phase in coded decentralized systems, as described shortly.

We adopt the popular communication model of [8], in which the processors are connected by a synchronous network, and messages can pass between any two of them. The system operates in consecutive communication rounds, during each a processor can simultaneously send and receive 1 message, which might contain multiple field elements, through each one
of its $p<K$ ports. Inspired by the well-familiar networkcoding literature [9, 10], we adopt a linear network model in which processors transmit linear combinations of their own data and previously received packets.

Similar to [8], to capture the communication cost we consider the time to pass a message containing $d$ field elements as $\beta+d \cdot \tau$, where $\beta$ and $\tau$ are system parameters; $\beta$ is the startup time of each message delivery, and $\tau$ refers to the perelement cost. We focus on two measures of communication:

- $C_{1}$ : the number of rounds incurred by the algorithm.
- $C_{2}$ : the total number of field elements transferred in a sequence during the operation. That is, $C_{2}=\sum_{t \in[T]} d_{t}$ in a $T$-round algorithm, where $d_{t}$ is the size of message containing the largest number of field elements ${ }^{1}$ transferred in round $t$, among all ports of all processors.
Our main goal in this paper is optimizing the total communication cost, given by $C_{1} \cdot \beta+C_{2} \cdot \tau$. Note that since each round includes sending at least one element from one node to another, it follows that $C_{2} \geq C_{1}$.

Clearly, all-to-all encode strongly depends on the properties of the underlying matrix $A$. Moreover, any algorithm for this problem contains two separate components, scheduling and coding scheme. The former determines which processor communicates with which other processors at each round, and the latter determines the coefficients in the linear combinations that processors transmit to one another. Motivated by this distinction, we address the all-to-all encode problem on two levels, the universal and the specific.

On the universal level, we seek a scheduling by which any matrix $A$ could be computed by only varying the coding scheme, i.e., the coefficients in the transmitted packets throughout the algorithm (see Fig. 1). That is, a universal algorithm is a series of instructions which indicate which processor communicates with which other processors at each round, alongside a mechanism that for every given $A$ determines the coefficients that are used by each processor in order to linearly combine previously received packets in each transmission. Universal algorithms are important in cases where the scheduling must either be determined prior to knowing $A$, must apply in several consecutive computations of different matrices, or its simplicity or uniformity are paramount (see Remark 1 below). Further, universal algorithms can be used as primitives in specific algorithms, as we show in the sequel.

On the specific level, we seek both scheduling and coding scheme that are uniquely tailored towards a specific matrix

[^0]

Round 1

Round 2

Fig. 1. An example of universal algorithm for computing any $A \in \mathbb{F}_{q}^{4 \times 4}$ in 2 rounds, when $p=1$. Every processor $k \in[0,3]$ wants $\widetilde{x}_{k}=$ $a_{0 k} x_{0}+a_{1 k} x_{1}+a_{2 k} x_{2}+a_{3 k} x_{3}$. In the first round, it receives $a_{(k-1) k} x_{k-1}$ from processor $k-1$. In the second round, it receives $a_{(k-2) k} x_{k-2}+$ $a_{(k-3) k} x_{k-3}$ from processor $k-2$. Finally, processor $k$ sums up the 2 received packets with $a_{k k} x_{k}$ and obtains $\widetilde{x}_{k}$.
of interest. Clearly, such specific algorithms are important only if they outperform universal ones, since by definition, every universal algorithm subsumes a specific algorithm for all matrices. We are particularly interested in Vandermonde and Lagrange matrices, that are required in Reed-Solomon and Lagrange coded systems, respectively.

Finally, we emphasize that in either the specific or the universal setting, neither the scheduling nor the coding scheme depend on the input packets $x_{i}$, but are exclusively determined by the matrix $A$, that is known to all processors.

## Our Contributions

- We provide lower bounds for $C_{1}$ and $C_{2}$ in a universal algorithm, and propose prepare-and-shoot, a universal algorithm which is optimal in $C_{1}$ and achieves the lower bound of $C_{2}$ within a factor of $\sqrt{2}$.
- We propose draw-and-loose, a family of specific algorithms for Vandermonde matrices, which optimize the aforementioned universal algorithm in terms of $C_{2}$, and provide similar gains for Lagrange matrices.

Remark 1 (All-to-all encode for decentralized encoding). Apart from its independent interest, the all-to-all encode operation can be used as a primitive in decentralized coded systems. For integers $N$ and $K$ such that $K \mid N$, consider a system with $N$ processors in which processor $i \in[K]$ holds $x_{i}$, where $[K] \triangleq\{1,2, \ldots, K\}$. Each processor $j \in[N]$ requires a different linear combination of the $x_{i}$ 's, defined by the $j$ 'th column of a predetermined generator matrix $G \in \mathbb{F}_{q}^{K \times N}$. The all-to-all encode operation defined herein is applicable to this setting as follows. Partition the processors to $N / K$ subsets of size $K$ each: $\{1, \ldots, K\},\{K+1, \ldots, 2 K\}$, and so on. First, each processor $i \in[K]$ disseminates its $x_{i}$ to the processors $\{\ell K+i\}_{\ell=1}^{N / K-1}$ using a simple tree-structured broadcast protocol with $\log _{p+1}(N / K)$ rounds. Then, each subset runs an all-to-all encode operation to compute the respective $K \times K$ submatrix of $G$.

## II. Related Works

Collective communication operations (e.g., one/all-to-all broadcast, one/all-to-all reduce, scatter, gather, etc.) have been
studied extensively due to their importance in parallel algorithms, see [12] for a thorough introduction to the topic. Yet, to the best of the authors' knowledge, an encompassing treatment of the all-to-all encode operation defined above is conspicuously absent from the literature, and several special cases have been scantly studied in recent years.
Jeong et al. [13] studies the decentralized encoding process of an $[N, K]$ systematic code as part of a coded FFT algorithm, but did not study the all-to-all encode problem. Decentralized encoding has also been studied by [6], in which processors pass on a random linear combination of packets to their neighbors, resulting in an MDS code with high probability. Similar problems have been studied in the signal processing literature under the title Graph Signal Processing [11], with a few substantial differences-a graph structure dictates the network connections, communication proceeds in so called graph-shift operations, and computations are over the real or complex fields.

## III. Lower Bounds

We now propose the lower bounds for $C_{1}$ and $C_{2}$ which apply to any universal algorithm.
Lemma 1. Any universal algorithm has $C_{1} \geq\left\lceil\log _{p+1} K\right\rceil$.
Proof. Since a universal algorithm must apply for all $K \times K$ matrices, it must also apply for matrices with no zero entries, i.e., every processor wants a linear combination of all initial packets. Similar to [8, Proposition 2.1], a packet $x_{k}$ can reach at most $(p+1)^{t}$ processors in the $p$-port model after round $t$, Hence, it takes at least $\left\lceil\log _{p+1} K\right\rceil$ rounds for any packet $x_{k}$ to reach all processors.
Lemma 2. Any universal algorithm has $C_{2} \geq \frac{\sqrt{2 K}}{p}-O(1)$.
Proof. We define baseline algorithms as a class of universal algorithms in which every processor passes exactly 1 field element through each port during every round; clearly, all baseline algorithms have $C_{1}=C_{2}$.

Given a universal algorithm in which processors transfer at most $d_{t}$ packets through one of the ports in round $t$ (and thus $C_{2}=\sum_{t} d_{t}$ ), there exists a corresponding baseline algorithm that "simulates" each round $t$ in the universal algorithm using $d_{t}$ rounds, and thus has the same $C_{2}=\sum_{t} d_{t}$. Therefore, a universal algorithm cannot outperform all baseline algorithms in terms of $C_{2}$. Hence, it suffices to bound $C_{1}$, and hence $C_{2}$, for baseline algorithms.

We provide a lower bound on $C_{1}$ for baseline algorithms by counting possible coding schemes, i.e., the number of ways processors can linearly combine previously received packets during the algorithm. Notice that once the scheduling and coding scheme are fixed, an algorithm cannot compute two distinct matrices ${ }^{2}$. Therefore, this value must be greater or equal to $\left|\mathbb{F}_{q}^{K \times K}\right|=q^{K^{2}}$.

[^1]Let $\mathbf{s}_{k}^{(t)}$ be the vector of packets received by processor $k$ prior to the beginning of round $t \in[T]$. Clearly, we have $\left|\mathbf{s}_{k}^{(t)}\right|=(t-1) p$. In round $t$, for every port $\rho \in[0, p-1]$, the processor creates and sends a packet $y_{k, \rho}^{(t)}$ by summing the initial packet $x_{k}$ with a linear combination of previously received packets ${ }^{3}$, i.e.,

$$
y_{k, p}^{(t)}=x_{k}+\left(\mathbf{g}_{k}^{(t)}\right)^{\top} \cdot \mathbf{s}_{k}^{(t)}
$$

where $\mathbf{g}_{k}^{(t)} \in \mathbb{F}_{q}^{p(t-1)}$ is a coding vector defining the linear combination of elements in $\mathbf{s}_{k}^{(t)}$. The number of possible coding vectors in round $t$ is then $q^{p(t-1)}$.

Note that there are $K$ processors in the $p$-port system, and the baseline algorithm contains $T$ rounds. Further, at the end of the $T$-round baseline algorithm, processor $k$ obtains $\widetilde{x}_{k}$ by linearly combining the $T p+1$ received packets, including the initial packet $x_{k}$. Therefore, the total number of coding schemes is

$$
\begin{equation*}
q^{K(T p+1)} \cdot \prod_{t \in[T]} q^{(t-1) K p^{2}} \geq q^{K \cdot K} \tag{1}
\end{equation*}
$$

Taking logarithm on both sides of Equation (1), we have

$$
K(T p+1)+\sum_{t \in[T]}(t-1) K p^{2} \geq K^{2}
$$

which simplifies to $p^{2} T^{2}-p(p-2) T+2(1-K) \geq 0$, and hence

$$
\begin{aligned}
C_{1}=T & \geq \frac{1}{2}-\frac{1}{p}+\sqrt{\frac{1}{4}-\frac{1}{p}-\frac{1}{p^{2}}+\frac{2 K}{p^{2}}} \\
& =\frac{\sqrt{2 K}}{p}-O(1)
\end{aligned}
$$

Remark 2 (Lower bound for specific algorithms). Clearly, specific algorithms can perform at least as good as universal ones in a any figure of merit; this is since any universal algorithm subsumes specific algorithms for all matrices by definition. Providing bound pertaining to specific algorithms proved to be a difficult task, and several such bounds will appear in future versions of this paper. Yet, it is readily verified that any matrix which contains a non-zero row cannot be computed with $C_{1}<\log _{p+1} K$; this is due to the simple fact that a given packet $x_{k}$ cannot be disseminated to all $K$ processors in less than this many rounds.

## IV. Prepare and Shoot: <br> An Optimal Universal Algorithm

In this section, we propose a universal algorithm that computes any matrix $A$. The proposed algorithm consists of two phases, prepare ( $T_{p}$ rounds) and shoot ( $T_{s}$ rounds). Let $L$ be the maximum integer such that $(p+1)^{L}<K$. If $L$ is even let $T_{p}=L / 2+1$ and $T_{s}=L / 2$. If $L$ is odd

[^2]

Fig. 2. Illustration of the dissemination of $x_{0}$ in the prepare phase in a 2-port system to $m=9$ processors. In the first round, processor 0 broadcasts $x_{0}$ to processor 3 and processor 6 . In the second round, processor $0 / 3 / 6$ passes $x_{0}$ to processor $1 / 4 / 7$ and processor $2 / 5 / 8$, and concludes the prepare phase.
let $T_{p}=T_{s}=(L+1) / 2$. In either case, the proposed algorithm has the optimal $C_{1}=T_{s}+T_{p}=\left\lceil\log _{p+1} K\right\rceil$ (see Lemma 1). To describe the phases, let $m=(p+1)^{T_{p}}, n=(p+1)^{T_{s}}$, and hence $(n-1) m<K \leq n m$. For every $k \in[0, K-1]$, let

$$
\begin{aligned}
\mathcal{R}_{k}^{+} & =\{k+\ell \mid \ell \in[0, m-1]\} \\
\mathcal{R}_{k}^{-} & =\{k-\ell \mid \ell \in[0, m-1]\} \\
\mathcal{S}_{k}^{+} & =\{k+\ell \cdot m \mid \ell \in[0, n-1]\}, \text { and } \\
\mathcal{S}_{k}^{-} & =\{k-\ell \cdot m \mid \ell \in[0, n-1]\}
\end{aligned}
$$

For convenience of notation, subscripts are computed $\bmod K$.
Prepare phase: This phase consists of $K$ one-to- $m$ broadcasts happening in parallel; each disseminates $x_{k}$ from processor $k \in[0, K-1]$ to processors in $\mathcal{R}_{k}^{+}$. In round $t$, for every $k, r \in[0, K-1]$, processor $k$ forwards $x_{r}$ (if present in its internal storage) to processor

$$
k+\rho \cdot \frac{m}{(p+1)^{t}}
$$

through its $\rho$-th port, for every $\rho \in[p]$. See Figure 2 for an illustrative example.

Lemma 3. After $C_{1, \text { prepare }}=T_{p}$ rounds, every processor $k$ has obtained $x_{r}$ for every $r \in \mathcal{R}_{k}^{-}$, with $C_{2, \text { prepare }}=\frac{(p+1)^{T_{p}}-1}{p}$.
Proof. Consider a tree defined recursively: Initially, processor $k$ is added as the root. In every subsequent round $t$, every processor $r$ in the current tree is connected to processor $r+\rho \cdot \frac{m}{(p+1)^{t}}$, for every $\rho \in[0, p-1]$ (i.e., the processors receiving messages from processor $r$ in round $t$ ). The recursion is concluded when $t=T_{p}$, after which the tree contains processors in $\mathcal{R}_{k}^{+}$.

Originally only processor $k$ holds $x_{k}$, and since the algorithm employs every processor to forward every packets it has, it follows that every processor in $\mathcal{R}_{k}^{+}$obtains $x_{k}$ after $T_{p}$ rounds. Therefore, processor $k$ has obtained $x_{r}$ for every $r \in \mathcal{R}_{k}^{-}$. Further, as the number of field elements in a message increase by $p$-fold after each round, we have

$$
C_{2, \text { prepare }}=\sum_{t=1}^{T_{p}}(p+1)^{t-1}=\frac{(p+1)^{T_{p}}-1}{p}
$$

Shoot phase: This phase consists of $K n$-to-one reduce operations happening in parallel, each intended to communicate the correct linear combination of packets to every processor.


Fig. 3. Illustration of the reduce operation from processors $60,3, \ldots, 57$ to processor 0 in the shoot phase in a 2 -port system with $K=65$.

At the beginning of the phase, every processor $k$ defines $n$ variables $w_{k, k}, w_{k, k+m}, \ldots, w_{k, k+(n-1) m}$ using the information received in the prepare phase (i.e., $x_{r}$ for $r \in \mathcal{R}_{k}^{-}$), and the coefficients of the matrix $A$. Intuitively, the variable $w_{k, s}$ contains a linear combination of $x_{r}$ 's at processor $k$, whose final destination is processor $s$; more and more packets will be added to $w_{k, s}$ as the algorithm progresses. Specifically, for $\ell \in[0, n-1]$, initialize $w_{k, k+\ell \cdot m}$ with $\mathbf{x}_{k} \cdot A_{k+\ell \cdot m}$, where $A_{k+\ell \cdot m}$ is the $(k+\ell \cdot m)$-th column of $A$, and the non-zero entries of $\mathbf{x}_{k} \in \mathbb{F}_{q}^{K}$ are indexed by elements in $\mathcal{R}_{k}^{-}$, i.e.,

$$
\mathbf{x}_{k}[r]= \begin{cases}x_{r} & r \in \mathcal{R}_{k}^{-}  \tag{2}\\ 0 & \text { otherwise }\end{cases}
$$

The goal of this phase is to allow every processor $k$ to obtain $y_{k}=\sum_{s \in \mathcal{S}_{k}^{-}} w_{s, k}=\sum_{s \in \mathcal{S}_{k}^{-}} \mathbf{x}_{s} \cdot \mathbf{A}_{k}$, where $w_{s, k}$ refers to the content of that variable at the beginning of this phase.

In round $t \in\left[1, T_{s}\right]$ of the shoot phase, processor $k$ forwards a message to processor $k+\rho m^{t}$, and receives a message from processor $k-\rho m^{t}$, for every $\rho \in[p]$. To describe this process, we now define a series of trees for every processor at every round, and an illustrative example can be found in Figure 3.

For every round $t$, let $\mathcal{T}_{k}^{(t)}$ be a tree defined recursively in $T_{s}-t$ steps, as follows. Initially, processor $k$ is added as the root. In every subsequent step $\tau$, for $\tau \in\left[1, T_{s}-t\right]$, every processor $r$ in the current tree is connected to processor $r+$ $\rho m^{t+\tau}$, for every $\rho \in[p]$. In particular, the tree $\mathcal{T}_{k}^{\left(T_{s}\right)}$ contains only the root $k$. Intuitively, the tree $\mathcal{T}_{k}^{(t)}$ contains processors that the processor $k$ is connecting to, directly or indirectly, in rounds following round $t$.

```
Algorithm 1 Shoot Phase (for processor \(k\) )
    Initialize \(w_{k, k}, w_{k, k+m}, \ldots, w_{k, k+(n-1) m}\).
    for \(t \leftarrow 1,2, \ldots, T_{s}\) do
        for \(\rho=1\) to \(p\) do \(\quad \triangleright\) As a sender
            \(s_{\text {out }}=k+\rho m^{t}\)
            send \(w_{k, r}\) to processor \(s_{\text {out }}\) for every \(r \in \mathcal{T}_{s_{\text {out }}}^{(t)}\)
        for \(\rho=1\) to \(p\) do \(\quad \triangleright\) As a receiver
            \(s_{\mathrm{in}}=k-\rho m^{t}\)
            for \(r \in \mathcal{T}_{k}^{(t)}\) do
                receive \(w_{s_{\text {in }}, r}\) from processor \(s_{\text {in }}\)
                \(\operatorname{assign} w_{k, r} \leftarrow w_{k, r}+w_{s_{\text {in }}, r}\)
    Output \(w_{k, k}\) as \(y_{k}\)
```

At round $t$, processor $k$ sends a message to processor $s_{\text {out }}=$ $k+\rho m^{\tau}$, and receives a message from processor $s_{\text {in }}=k-$ $\rho m^{\tau}$ through the $\rho$-th port for $\rho \in[p]$. The sent message contains $w_{s_{\text {out }}, r}$ for every $r \in \mathcal{T}_{S_{\text {out }}}^{(t)}$. The received message contains $w_{s_{\text {in }}, r}$ for every $r \in \mathcal{T}_{k}^{(t)}$, and the processor $k$ updates its internal storage, letting $w_{k, r} \leftarrow w_{k, r}+w_{s_{\text {in }}, r}$. The details are given in Algorithm 1, and the correctness is as follows.

Lemma 4. After $C_{1, \text { shoot }}=T_{s}$ rounds, every processor $k$ has obtained $y_{k}=\sum_{r \in \mathcal{S}_{k}^{-}} \mathbf{x}_{r} \cdot \mathbf{A}_{k}$, with $C_{2, \text { shoot }}=\frac{(p+1)^{T_{s}}-1}{p}$.
Proof. We show that the variable $w_{k, k}$ stores $y_{k}$ for every $k$ at the end of the algorithm. The proof is based on an recursively defined tree $\mathcal{T}_{k}^{\prime}$ : in step $\tau=0$ processor $k$ is added as the root. In every subsequent step $\tau \geq 1$ every processor $r$ in the current tree is connected to processor $r-\rho m^{\tau}$, for every $\rho \in[p]$. That is, during each step $\tau$, we add processors that sent messages to the existing processors at round $t=T_{s}-\tau+1$.

Observe that the tree $\mathcal{T}_{k}^{\prime}$ contains all processors in $\mathcal{S}_{k}^{-}$ after $T_{s}$ steps of the recursion. For each processor $r \in \mathcal{T}_{k}^{\prime}$ added in step $\tau$, processor $k$ is present in the tree $\mathcal{T}_{r}^{(t)}$. Hence, at round $t$, the variable $w_{r, k}$ is transmitted from processor $r$ to its parent $s$ in $\mathcal{T}_{k}^{\prime}$, and summed with $w_{s, k}$.

Traversing from the processors added at step $T_{s}$, the packets $\left\{\mathbf{x}_{r} \cdot \mathbf{A}_{k}\right\}_{r \in \mathcal{T}_{k}^{\prime}}$ are summed and transmitted to the root $k$, and stored in the variable $w_{k, k}$. Note that tree $\mathcal{T}_{k}^{\prime}$ contains exactly the processors in $\mathcal{S}_{k}^{-}$, and hence $w_{k, k}=y_{k}$ stores $\sum_{r \in \mathcal{S}_{k}^{-}} \mathbf{x}_{r} \cdot \mathbf{A}_{k}$.

Finally, observe that $\left|\mathcal{T}_{k}^{(t)}\right|=\frac{n}{(p+1)^{t}}$, which is the number of field elements sent by processor $k$ at round $t$ through each of its ports. Hence, summing over all rounds, we have

$$
C_{2, \text { shoot }}=\sum_{t=1}^{T_{s}}(p+1)^{T_{s}-t}=\frac{(p+1)^{T_{s}}-1}{p}
$$

Finally, in the most general case where $K<m n$, some overlap of indices need to be resolved, as some computation results have been summed up twice. In particular, observe that $\mathcal{R}_{k}^{-} \cap \mathcal{R}_{k-(n-1) m}^{-}=[k-n m+1, k]$, which is an empty set only if $n m=K$, as we assumed that $(n-1) m<K \leq n m$, and since indices are computed $\bmod K$. Therefore,

$$
\begin{equation*}
y_{k}=\widetilde{x}_{k}+\sum_{r \in[k-m n+1, k]} A_{r, k} x_{r} \tag{3}
\end{equation*}
$$

from which processor $k$ can individually compute $\widetilde{x}_{k}$ with no communication, by computing the r.h.s sum and subtracting from $y_{k}$. This concludes the prepare-and-shoot algorithm, and provides the following by Lemma 3 and Lemma 4.

Theorem 1. The prepare-and-shoot algorithm has $C_{1}=T_{p}+$ $T_{s}=\left\lceil\log _{p+1} K\right\rceil$ and

$$
C_{2}=\left\{\begin{array}{ll}
\frac{2(p+1)^{(L+1) / 2}-2}{p^{p}} & \text { if } L \text { is odd } \\
\frac{(p+1)^{L}{ }^{2+1}-2}{p} & \text { if } L \text { is even }
\end{array} .\right.
$$

Remark 3. According to Lemma 1, the prepare-and-shoot algorithm is strictly optimal in terms of $C_{1}$. In addition,


Fig. 4. Illustration of the trees with $K=9$ and $p=2$. (left) The polynomial tree rooted at $f(z)=x_{0}+x_{1} z+\cdots+x_{8} z^{8}$. The polynomials in the first level are $f_{0}(z)=x_{0}+x_{3} z+x_{6} z^{3}, f_{1}(z)=x_{1}+x_{4} z+x_{7} z^{3}$, etc. see (7). (right) The tree of field elements rooted at $\gamma=1$, with $\gamma_{1}=g^{3}, \gamma_{10}=g$, etc, where $g$ is a generator of $\mathbb{F}_{q}$, see (5).
since $(p+1)^{L}<K$, by Lemma 2 the algorithm is asymptotically optimal in terms of $C_{2}$.

## V. Draw and Loose: An Algorithm for Computing Vandermonde Matrices

In this section, we shift our attention to specific algorithms for computing Vandermonde matrices, for their prevalent use in Reed-Solomon codes. That is, we tailor both the scheduling and the coding scheme for $K \times K$ matrices $A$ such that $A_{i, j}=$ $\alpha_{j}^{i}$ for $i, j \in[0, K-1]$, where $\alpha_{0}, \ldots, \alpha_{K-1}$ are distinct elements of $\mathbb{F}_{q}$ (hence $K \leq q$ ). That is, every processor wishes to obtain $\widetilde{x}_{k}=f\left(\alpha_{k}\right)$, an evaluation of the polynomial $f(z)=$ $\sum_{k \in[0, K-1]} x_{k} z^{k}$ at $\alpha_{k}$.

Inspired by the Fast Fourier Transform algorithm, we show a method that computes the Discrete Fourier Transform (DFT) matrix (a special case of Vandermonde matrix) with the optimal $C_{1}=C_{2}=\log _{p+1} K$ (see Remark 2). Later, this method serves as a primitive for computation of general Vandermonde matrices, and brings a significant gain in $C_{2}$ compared with the universal algorithm described earlier.

## A. Computing a DFT Matrix

Assume that $K=(p+1)^{H}$ for some positive integer $H$, that $K \mid q-1$, and let $\beta=g^{\frac{q-1}{K}}$ be a primitive $K$-th root of unity, where $g$ is a generator of $\mathbb{F}_{q}$. A Discrete Fourier Transform (DFT) matrix $D_{K}$ is a Vandermonde one with $\alpha_{k}=$ $\beta^{k}$, i.e.,

$$
D_{K}=\left[\begin{array}{ccccc}
1 & 1 & 1 & \cdots & 1  \tag{4}\\
1 & \beta & \beta^{2} & \cdots & \beta^{K-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \beta^{(K-1)} & \beta^{2(K-1)} & \cdots & \beta^{(k-1)(K-1)}
\end{array}\right]
$$

The proposed method relies on two complete $(p+1)$-ary trees, a tree of field elements and a tree of polynomials. Each tree is of height $H$, and there are $(p+1)^{h}$ nodes at level $h \in[0, H]$. Therefore, a node at level $h$ can be represented by $h$ digits using $(p+1)$-radix.

The tree of field elements is defined as follows. A node at level $h \in[0, H]$ is identified by $h$ digits $k_{h-1} \ldots k_{0}$ in $[0, p]$ ( $k_{h-1}$ being the most significant), and contains the element

$$
\begin{equation*}
\gamma_{k_{h-1} \cdots k_{0}} \triangleq\left(\beta^{k_{h-1}(p+1)^{h-1}+\cdots+k_{0}}\right)^{(p+1)^{H-h}} \tag{5}
\end{equation*}
$$

It is readily verified that for each of the leaves (at level $H$ ), we have that $\gamma_{k_{H-1} \cdots k_{0}}=\beta^{k}$ is the evaluation point of the processor indexed by $k=k_{H-1}(p+1)^{H-1}+\cdots+k_{0}$. For the root (at level 0 ), which is represented by 0 digits, we have $\gamma=1$. Indices of sibling nodes differ only in the most significant (leftmost) digit, and index of a parent node is given by omitting the most significant digit of its child. That is, nodes $\gamma_{0 k_{h-1} \cdots k_{0}}, \ldots, \gamma_{p k_{h-1} \cdots k_{0}}$ are the children of the same parent node $\gamma_{k_{h-1} \cdots k_{0}}$. By Equation (5), every child is a distinct $(p+1)$-th root of its parent, i.e., for every $\rho \in[0, p]$,

$$
\begin{align*}
& \left(\gamma_{\rho k_{h-1} \cdots k_{0}}\right)^{p+1} \\
& =\left(\left(\beta^{\rho(p+1)^{h}+k_{h-1}(p+1)^{h-1}+\cdots+k_{0}}\right)^{(p+1)^{H-h-1}}\right)^{p+1} \\
& =\left(\beta^{\rho(p+1)^{h}+k_{h-1}(p+1)^{h-1}+\cdots+k_{0}}\right)^{(p+1)^{H-h}} \\
& =\left(\beta^{k_{h-1}(p+1)^{h-1}+\cdots+k_{0}}\right)^{(p+1)^{H-h}} \cdot \beta^{\rho(p+1)^{H}} \\
& =\gamma_{k_{h-1} \cdots k_{0}} \cdot\left(g^{\frac{q-1}{K}}\right)^{\rho K}=\gamma_{k_{h-1} \cdots k_{0}} \tag{6}
\end{align*}
$$

The tree of polynomials is defined recursively from the root labelled by $f(z)$. For a non-leaf node labelled by $f_{k_{0} \cdots k_{h-1}}(z)=b_{0} z^{0}+\cdots+b_{D-1} z^{D-1}$ at level $h$, its $\rho$-th child node is defined as

$$
\begin{equation*}
f_{k_{0} \cdots k_{h-1} \rho}(z)=\sum_{d=\rho \bmod (p+1)} b_{d} z^{\frac{d-\rho}{p+1}} \tag{7}
\end{equation*}
$$

Therefore, every non-leaf node is a combination of its children evaluated at $z^{p+1}$, i.e.,

$$
\begin{equation*}
f_{k_{0} \cdots k_{h-1}}(z)=\sum_{\rho \in[0, p]} z^{\rho} f_{k_{0} \cdots k_{h-1} \rho}\left(z^{p+1}\right) \tag{8}
\end{equation*}
$$

Note that the leaf node $f_{k_{0}, \ldots, k_{H-1}}=x_{k^{\prime}}$ is the initial packet of the processor indexed by $k^{\prime}=k_{H-1}+\cdots+$ $k_{0}(p+1)^{H-1}$. That is, the $(p+1)$-radix representations of $k=k_{H-1}(p+1)^{H-1}+\cdots+k_{0}$ (defined above) and $k^{\prime}$ have reversed order. See Figure 4 for illustrations of both trees.

The proposed algorithm is defined recursively using the above trees. Define $Q(k, t)=f_{k_{H-1} \cdots k_{t}}\left(\gamma_{k_{t-1} \cdots k_{0}}\right)$, and hence every processor $k$ initially has $Q(k, 0)=f_{k_{H-1} \cdots k_{0}}(\gamma)=$ $f_{k_{0}^{\prime} \cdots k_{H-1}^{\prime}}(\gamma)=x_{k}$ at the beginning of the algorithm, and wishes to obtain $Q(k, H)=f\left(\gamma_{k_{H-1} \cdots k_{0}}\right)=f\left(\beta^{k}\right)=\widetilde{x}_{k}$.

Assume that processor $k$ has $Q(k, t)$ after round $t$, and we show how it can obtain $Q(k, t+1)$ in one round. By Equation (8) and Equation (5) we have

$$
\begin{align*}
Q(k, t+1) & =f_{k_{H-1} \cdots k_{t+1}}\left(\gamma_{k_{t} \cdots k_{0}}\right) \\
& =\sum_{\rho \in[0, p]}\left(\gamma_{k_{t} \cdots k_{0}}\right)^{\rho} f_{k_{H-1} \cdots k_{t+1} \rho}\left(\left(\gamma_{k_{t} \cdots k_{0}}\right)^{p+1}\right) \\
& =\sum_{\rho \in[0, p]}\left(\gamma_{k_{t} \cdots k_{0}}\right)^{\rho} f_{k_{H-1} \cdots k_{t+1} \rho}\left(\gamma_{k_{t-1} \cdots k_{0}}\right) \\
& =\sum_{\rho \in[0, p]}\left(\gamma_{k_{t} \cdots k_{0}}\right)^{\rho} Q\left(k_{\rho}^{(t)}, t\right) \tag{9}
\end{align*}
$$

where $k_{\rho}^{(t)}$ is represented by $k_{H-1} \cdots k_{t+1} \rho k_{t-1} \cdots k_{0}$ in $(p+$ 1)-radix, i.e., it differs from $k$ only in the $t$ 'th digit. Therefore, the desired $Q(k, t+1)$ is a linear combination of $Q\left(k_{0}^{(t)}, t\right), \ldots, Q\left(k_{p}^{(t)}, t\right)$. Written in matrix form we have

$$
\left[\begin{array}{c}
Q\left(k_{0}^{(t)}, t+1\right)  \tag{10}\\
\vdots \\
Q\left(k_{p}^{(t)}, t+1\right)
\end{array}\right]=A_{k}^{(t)} \cdot\left[\begin{array}{c}
Q\left(k_{0}^{(t)}, t\right) \\
\vdots \\
Q\left(k_{p}^{(t)}, t\right)
\end{array}\right]
$$

where $A_{k}^{(t)}=\left[\begin{array}{ccc}\left.\gamma_{\left(0 k_{t-1} \cdots k_{0}\right.}\right)^{0} & \cdots & \left(\gamma_{0 k_{t-1} \cdots k_{0}}\right)^{P} \\ \vdots & \ddots & \vdots \\ \left.\gamma_{\left(p k_{t-1} \cdots k_{0}\right.}\right)^{0} & \cdots & \left(\gamma_{p k_{t-1} \cdots k_{0}}\right)^{P}\end{array}\right]$.
At round $t+1$ of the proposed algorithm, every processor $k$ broadcasts $Q(k, t)$ to the $p$ processors having the same index except for the $t$-th digit. In the same round, processor $k$ receives $Q\left(k_{\rho}^{(t)}, t\right)$ for every $\rho \in[0, p]$ from these processors (including itself). Then, processor $k$ obtains $Q(k, t+1)$ by linearly combining the received packets based on Equation (9).

Recall that every processor $k$ has $Q(k, 0)$ at beginning, it obtains the coded packet $\widetilde{x}_{k}=Q(k, H)$ after $H$ rounds by repeating the above operation $H$ times. Since exactly one packet is transmitted through each of the $p$ ports during each operation, we have the following theorem.

Theorem 2. The above algorithm for computing a DFT matrix has $C_{1}=C_{2}=H=\log _{p+1} K$, which is strictly optimal.

Remark 4. As shown in Remark 2, the proposed algorithm has the strictly optimal $C_{1}$ value. Further, the $C_{2}$ value is also optimal, since during each round only 1 packet is communicated through each port. This is an exponential improvement over the universal algorithm.

Next, we emphasize the invertibility of the presented algorithm in the following lemma; this will be useful in the sequel.

Lemma 5. The above algorithm can be used to compute the inverse of a DFT matrix, with the same $C_{1}$ and $C_{2}$.
Proof. Since the matrix $A_{k}^{(t)}$ defined in Equation (11) is an invertible Vandermonde matrix, it follows that each step of the induction is invertible. That is, the processor $k$ in possession of $Q(k, t+1)$ can obtain $Q(k, t)$ in one round of communication, sending and receiving 1 packet through each of its ports, for every $t \in[0, H]$. Therefore, a processor $k$ in possession of $Q(k, H)$ can obtain $Q(k, 0)$ in $H$ rounds, with the optimal $C_{1}$ and $C_{2}$ as shown in Theorem 2.

## B. Generalization

The aforementioned algorithm computes a unique Vandermonde matrix with strictly optimal $C_{1}$ and $C_{2}$, but requires that $K=(p+1)^{H}$ for some $H$ and that $K \mid q-1$. In cases where $K \nmid q-1$ and $K \leq q-1$, let $H$ be the maximum integer such that $(p+1)^{H}$ divides $\operatorname{gcd}(K, q-1)$, and denote $K=M \cdot(p+1)^{H}$. We use the above DFT algorithm as a primitive for improved $C_{2}$ with respect to prepare-and-shoot in the computation of multiple other Vandermonde matrices.

Let $Z=(p+1)^{H}$, and denote processor $P_{i, j}=j+Z \cdot i$ by two indices $j \in[0, Z-1]$ and $i \in[0, M-1]$. We define the evaluation point of processor $P_{i, j}$ as $\alpha_{i, j}=\alpha_{i} \cdot \beta_{j}$, where $\alpha_{i}=g^{\varphi(i)}$ and $\beta_{j}=g^{j \cdot \frac{q-1}{z}}$, with $g$ being a generator of $\mathbb{F}_{q}$ and $\varphi$ being any injective map from $[0, M-1]$ to $[0,(q-1) / Z-1]$, which exists since $q-1 \geq Z M=K$. These definitions guarantee that no two evaluation points are identical. In addition, since there exists $\binom{(q-1) / Z}{M}$ possible choices for $\varphi$, it follows that the proposed algorithm computes this many different Vandermonde matrices up to permutation of columns.

Recall that the coded packet $\widetilde{x}_{i, j}$ desired by processor $P_{i, j}$ is an evaluation of $f(z)$ at $\alpha_{i, j}$. Moreover, we have

$$
\begin{align*}
f\left(\alpha_{i, j}\right) & =\sum_{k=0}^{K-1} x_{k}\left(\alpha_{i, j}\right)^{k}=\sum_{k=0}^{K-1} x_{k} \alpha_{i}^{k} \beta_{j}^{k} \\
& =\sum_{\ell=0}^{Z-1} \beta_{j}^{\ell} \sum_{k=\ell \bmod Z} x_{k} \alpha_{i}^{k} \beta_{j}^{k-\ell} \tag{12}
\end{align*}
$$

Since $g^{q-1}=1$, it follows that $\beta_{j}^{k-\ell}=g^{j \cdot \frac{q-1}{Z}(k-\ell)}=1$ whenever $k=\ell \bmod Z$, and hence

$$
\begin{equation*}
\widetilde{x}_{i, j} \stackrel{(12)}{=} \sum_{\ell=0}^{Z-1} \beta_{j}^{\ell} \sum_{k=\ell \bmod Z} x_{k} \alpha_{i}^{k}=\sum_{\ell=0}^{Z-1} \beta_{j}^{\ell} f_{\ell}\left(\alpha_{i}\right) \tag{13}
\end{equation*}
$$

where $f_{\ell}(z)=\sum_{w=0}^{M-1} x_{w, \ell} z^{\ell+Z \cdot w}$. In matrix form,

$$
\widetilde{X} \triangleq\left[\begin{array}{ccc}
\widetilde{x}_{0,0} & \cdots & \widetilde{x}_{0, Z-1}  \tag{14}\\
\vdots & \ddots & \vdots \\
\widetilde{x}_{M-1,0} & \cdots & \widetilde{x}_{M-1, Z-1}
\end{array}\right]=F \cdot D_{L}
$$

where $D_{L}$ is an $Z \times Z$ DFT matrix, and

$$
F=\left[\begin{array}{ccc}
f_{0}\left(\alpha_{0}\right) & \cdots & f_{Z-1}\left(\alpha_{0}\right)  \tag{15}\\
\vdots & \ddots & \vdots \\
f_{0}\left(\alpha_{M-1}\right) & \cdots & f_{Z-1}\left(\alpha_{M-1}\right)
\end{array}\right]
$$

Notice that the $j$-th column of $F$ satisfies

$$
\begin{align*}
& {\left[f_{j}\left(\alpha_{0}\right), \cdots, f_{j}\left(\alpha_{M-1}\right)\right]^{\top}=}  \tag{16}\\
& {\left[\begin{array}{cccc}
\alpha_{0}^{j} & & \\
& \ddots & \\
& & \alpha_{M-1}^{j}
\end{array}\right] \cdot\left[\begin{array}{ccc}
\alpha_{0}^{Z \cdot 0} & \cdots & \alpha_{0}^{Z \cdot(M-1)} \\
\vdots & \ddots & \vdots \\
\alpha_{M-1}^{Z \cdot 0} & \cdots & \alpha_{M-1}^{Z \cdot(M-1)}
\end{array}\right] \cdot\left[\begin{array}{c}
x_{0, j} \\
\vdots \\
x_{M-1, j}
\end{array}\right]}
\end{align*}
$$

The protocol proceeds in two phases.
Draw Phase: The objective of this phase is for every processor $P_{i, j}$ to obtain $f_{j}\left(\alpha_{i}\right)$. As shown above (16), $f_{j}\left(\alpha_{0}\right), \ldots, f_{j}\left(\alpha_{M-1}\right)$ are given by multiplying $\left[x_{0, j}, \ldots, x_{M-1, j}\right]^{\top}$ by a Vandermonde matrix, denoted by $V$, and a diagonal matrix $\operatorname{diag}\left(\alpha_{0}^{j}, \cdots, \alpha_{M-1}^{j}\right)$. Therefore, this problem can be solved in parallel by $Z$ all-to-all encode operations. For every $j \in[0, Z-1]$, processors $\quad P_{0, j}, \ldots, P_{M-1, j} \quad$ collectively compute the matrix $V$ using prepare-and-shoot (Section IV). Once completed, every processor $P_{i, j}$ locally multiplies the resulting packet with $\alpha_{i}^{j}$ and obtains $f_{j}\left(\alpha_{i}\right)$, which is the element in the $i$-th row and $j$-th column of the matrix $F$.

Loose Phase: As shown in Equation (14), the coded packets $\widetilde{x}_{i, 0}, \ldots, \widetilde{x}_{i, Z-1}$ are linear combinations, defined by the DFT matrix $D_{L}$, of the elements $f_{0}\left(a_{i}\right), \ldots, f_{Z-1}\left(a_{i}\right)$ in the $i$-th row of the matrix $F$. For every $j \in[0, Z-1]$, processors $P_{i, 0}, \ldots, P_{i, Z-1}$ collectively compute $D_{L}$ using the specialized algorithm a for DFT matrix (Section V-A). After $H$ rounds, every processor $P_{i, j}$ obtains the coded packet $\widetilde{x}_{i, j}$.

Let $\Psi(M)$ be the $C_{2}$ in the prepare-and-shoot algorithm to compute any $M \times M$ matrix. Observe that the draw phase takes $C_{1}=\left\lceil\log _{p+1} M\right\rceil$ rounds and $C_{2}=\Psi(M)$ communication, and the loose phase takes $C_{1}=H$ rounds and $C_{2}=H$ communication. Therefore, we have the following.

Theorem 3. The draw-and-loose algorithm can compute $\binom{(q-1) / Z}{M}$ different Vandermonde matrices (up to permutation of columns) with $C_{1}=\left\lceil\log _{p+1} K\right\rceil$ and $C_{2}=$ $H+\Psi(M)$. In particular, if $M \leq p+1$ then $\Psi(M)=1$, i.e., $C_{1}=C_{2}=\left\lceil\log _{p+1} K\right\rceil$.

Remark 5. Note that draw-and-loose can compute any Vandermonde matrix. Yet, significant gains w.r.t prepare-and-shoot will be possible in cases where $H$ is large. In particular, in cases where $H=0$ the draw-and-loose algorithm does not provide gains over the universal prepare-and-shoot.

Lemma 6. Similar to Lemma 5, the above algorithm can be used to compute the inverse of a Vandermonde matrix, with the same $C_{1}$ and $C_{2}$.

Proof. The invertibility of the loose phase is given in Lemma 5. In the draw phase, since the Vandermonde matrix $V$ defined in Equation (15) is invertible, this step can be inverted by computing the inverse of $V$ using prepare-and-shoot.

Together, the inverse of a Vandermonde matrix can be computed by first inversing the loose phase, and computing the inverse of $V$ with prepare-and-shoot.

## VI. Computing Lagrange Matrices with Invertible Draw-And-Loose

Lagrange matrices were recently popularized for their use in coded computing [7]. For sets $\left\{\alpha_{i}\right\}_{i=1}^{K}$ and $\left\{\omega_{i}\right\}_{i=1}^{K}$, each with $K$ distinct elements in $\mathbb{F}_{q}$, let

$$
A=\left[\begin{array}{ccc}
\Phi_{1}\left(\alpha_{1}\right) & \ldots & \Phi_{1}\left(\alpha_{K}\right) \\
\vdots & \ddots & \vdots \\
\Phi_{K}\left(\alpha_{1}\right) & \ldots & \Phi_{K}\left(\alpha_{K}\right)
\end{array}\right], \Phi_{k}(z)=\prod_{j \neq k} \frac{z-\omega_{j}}{\omega_{k}-\omega_{j}} .
$$

In this section we sketch an extension of draw-and-loose which computes Lagrange matrices. Evidently, computing a Lagrange matrix can be described as follows. Every processor $k$ has $x_{k}=f\left(\omega_{k}\right)$; they together form a point-value representation of a polynomial $f(z)=\sum_{k \in[0, K-1]} f_{k} z^{k}$ of degree $K-1$ at $\omega_{0}, \ldots, \omega_{K-1}$. Every processor $k$ wants $\widetilde{x}_{k}=$ $f\left(\alpha_{k}\right)$, i.e., another point-value representation of $f(z)$ on $\alpha_{0}, \ldots, \alpha_{K-1}$.

Therefore, computing a Lagrange matrix is possible by two consecutive computations. First, compute the inverse of a Vandermonde matrix $V\left(\omega_{1}, \ldots, \omega_{K}\right)$ (Lemma 6) in order to obtain the coefficients of the polynomial $f$. Second,
compute the Vandermonde matrix $V\left(\alpha_{1}, \ldots, \alpha_{K}\right)$ in order to evaluate $f$ at $\alpha_{k}$ for every $k \in[K]$. This yields the following, in which $C_{i}(x)$ is the $C_{i}$ measure of draw-andloose over $V\left(x_{1}, \ldots, x_{k}\right)$, for $i \in\{1,2\}$ and $x \in\{\alpha, \omega\}$.

Theorem 4. The above algorithm computes a Lagrange matrix with $C_{1}=C_{1}(\omega)+C_{1}(\alpha)$ and $C_{2}=C_{2}(\omega)+C_{2}(\alpha)$.
Proof. Written in matrix form, the initial packets

$$
\left(x_{0}, \ldots, x_{K-1}\right)=\left(f_{0}, \ldots, f_{K-1}\right)\left[\begin{array}{ccc}
\omega_{0}^{0} & \cdots & \omega_{n-1}^{0}  \tag{17}\\
\vdots & \ddots & \vdots \\
\omega_{0}^{n-1} & \cdots & \omega_{n-1}^{n-1}
\end{array}\right]
$$

are linear combination of $f_{0}, \ldots, f_{K-1}$ defined by the Vandermonde matrix $V\left(\omega_{0}, \ldots, \omega_{K-1}\right)$. By the invertibility of draw-and-loose (lemma 6), this step can be completed with the same $C_{1}$ and $C_{2}$ introduced in Theorem 3, and every processor $k$ obtains $f_{k}$. Next, the coded packets

$$
\left(\widetilde{x}_{0}, \ldots, \widetilde{x}_{K-1}\right)=\left(f_{0}, \ldots, f_{K-1}\right)\left[\begin{array}{ccc}
\alpha_{0}^{0} & \cdots & \alpha_{n-1}^{0}  \tag{18}\\
\vdots & \ddots & \vdots \\
\alpha_{0}^{n-1} & \cdots & \alpha_{n-1}^{n-1}
\end{array}\right]
$$

are linear combination of $f_{0}, \ldots, f_{K-1}$ defined by the Vandermonde matrix $V\left(\alpha_{0}, \ldots, \alpha_{K-1}\right)$. Since every processor $k$ has obtained $f_{k}$, this phase can be completed using Vandermonde algorithm.

## VII. DISCUSSION AND FUTURE WORK

Directions for future work include extending the results to other specific matrices, e.g., Cauchy matrices and Moore matrices, and to further study algorithms and lower bounds for Vandermonde and Lagrange matrices. Also, incorporating computation and storage constraints of processors is an interesting direction for future research.

## REFERENCES

[1] S. Li, M. Yu, C.-S. Yang, A. S. Avestimehr, S. Kannan, and P. Viswanath, "Polyshard: Coded sharding achieves linearly scaling efficiency and security simultaneously," IEEE Transactions on Information Forensics and Security, vol. 16, pp. 249-261, 2020.
[2] C. Wang and N. Raviv, "Low Latency Cross-Shard Transactions in Coded Blockchain," in IEEE International Symposium on Information Theory, pp. 26782683, 2021.
[3] N. A. Khooshemehr and M. A. Maddah-Ali, "The Discrepancy Attack on Polyshard-ed Blockchains," in IEEE Int. Symp. Information Theory, pp. 2672-2677, 2021.
[4] B. Li, J. Yang, H. Yang, G. Liu, R. Ma, and X. Peng, "Decode-and-forward cooperative transmission in wireless sensor networks based on physical-layer network coding," Wireless Networks, pp. 1-7, 2019.
[5] Y.-J. Chen, and L.-C Wang, "Privacy protection for Internet of drones: A network coding approach," IEEE Internet of Things Journal, vol 6, no. 2,pp 1719-1730, 2018.
[6] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Decentralized Erasure Codes for Distributed Networked Storage," IEEE Transactions on Information Theory, vol. 52, no. 6, pp 2809-2816, 2006.
[7] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in International Conference on Artificial Intelligence and Statistics, pp. 1215-1225, 2019.
[8] J. Bruck, C-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby, "Efficient algorithms for all-to-all communications in multiport message-passing systems," IEEE Transactions on parallel and distributed systems, vol. 8, no. 11, pp 1143-1156, 1997.
[9] R. Ahlswede, N. Cai, S.-Y. R .Li, and R. W. Yeung, "Network information flow," IEEE Transactions on information theory 46, no. 4, pp. 1204-1216, 2000.
[10] S.-Y. R. Li, R. W. Yeung, and N Cai, "Linear network coding," IEEE transactions on information theory, vol .49, no. 2, pp 371-381, 2003.
[11] S. Segarra, A. G. Marques, and A. Ribeiro, "Optimal graph-filter design and applications to distributed linear network operators," IEEE Transactions on Signal Processing, vol. 65, no. 15, pp. 4117-4131, 2017.
[12] V. Kumar, A. Grama, A. Gupta, and G. Karypis, Introduction to parallel computing, vol. 110, Redwood City, CA, USA: Benjamin Cummings, 1994.
[13] H. Jeong, T. M. Low, and P. Grover, "Masterless coded computing: A fully-distributed coded FFT algorithm," in 2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton), pp. 887894, 2018.
[14] C. Fragouli, J.-Y. Le Boudec, and J. Widmer, "Network coding: An instant primer," ACM SIGCOMM Computer Communication Review 36, no. 1, pp 63-68, 2006.


[^0]:    ${ }^{1}$ Since the operation proceeds in rounds, the largest message in an individual round determines the time duration of this round.

[^1]:    ${ }^{2}$ Recall that an algorithm computes a matrix $A$ only if it computes the product $x A$ for every $x=\left(x_{1}, \ldots, x_{k}\right) \in \mathbb{F}_{q}^{K}$. Hence, if $x A=x B$ for every $x \in \mathbb{F}_{q}^{K}$, then $\operatorname{ker}(A-B)=\mathbb{F}_{q}^{K}$, which implies $A=B$.

[^2]:    ${ }^{3}$ Since the receiver will linearly combine the received packets in future rounds of network coding, the initial packet $x_{k}$ has coefficient 1 to avoid overcounting the coding schemes.

