Distributed Error Correction Coding Scheme for Low Storage Blockchain Systems

Huihui Wu, Alexei Ashikhmin, *Fellow, IEEE*, Xiaodong Wang, *Fellow, IEEE*, Chong Li, *Senior Member, IEEE*, Sichao Yang, and Lei Zhang

***Abstract*—This article presents a novel way to reduce blockchain nodes' memory requirements using error correcting codes. In particular, LDPC codes are taken as examples to explicitly demonstrate the scheme. The proposed coding scheme encodes data across multiple blocks, respectively, block headers, in the blockchain. This leads to a significant reduction in required memory at each node. We then apply the proposed coding technique to blockchains organized in two different ways. Our first scheme has the same protocol for mining, broadcasting, and verification of blocks, as Bitcoin-type blockchains. Our scheme is different in that *full nodes* do not have to store all blocks. Instead they will need to store only one block of a group of t blocks. In the second scheme, we consider a new block verification protocol and an account-based model under the assumption that transmission between any two nodes can be established, as well as the broadcast transmission. Our block verification protocol uses the Byzantine fault tolerance algorithm and requires sending a newly mined block to only a small number of verification nodes, instead of broadcasting it to the entire network, which leads to a reduction of the network load.**

***Index Terms*—Block verification protocol, blockchain, Byzantine fault tolerance (BFT) algorithm, distributed storage, error correcting codes, low-density parity-check codes.**

I. Introduction

BLOCKCHAIN is a distributed system that builds consensus among the untrusted participants. One of its most popular applications is the cryptocurrency. Since the launch of the Bitcoin in 2009 [1], many cryptocurrency systems have been developed, e.g., Ethereum [2], Ripple [3], Cardano [4], Zcash [5], etc. Moreover, blockchain systems are also useful in various applications, such as medical data management [6], smart contracts [7], and so on. It is believed that the blockchain technique has the potential to revolutionize the digital world [8].

Nowadays, the Internet of Things (IoT) connects billions of devices, and each of them generates and exchanges massive numbers of data. Accordingly, the security issues for such a large IoT network needs to be addressed, which is a challengeable work. First, devices can collude to collapse the IoT or launch cyber attacks independently. Second, data confidentiality, integrity, and authentication are also vital and fundamental issues. Moreover, typically IoT networks rely on a central cloud service provider [9], whose failure would be a disaster. The aforementioned concerns call for the application of blockchain into IoT networks. For example, the blockchain has been applied in smart grid [10], [11], and IoT data storage [12], and more applications can be found in [13].

It is also important to note that since devices in the IoT network may have very limited memories, it looks natural to design blockchains with only light nodes with small memory capabilities. Design and analysis of such blockchains are considered in Sections V and VI, respectively.

Currently, one premise of blockchain systems is that each full node needs to store the entire blockchain, which will eventually lead to a lack of storage resources. Specifically, the total size of the Bitcoin blockchain amounts to around 198 GB [14], including all block headers and transactions, to the date of February 8, 2019. Therefore, the scalability problem has become one of the major concerns in the blockchain community, as this is vital for the large-scale application of blockchain.

One solution for the increasing storage problem is to reclaim the disk space [1] by pruning the old transactions in the blockchain. However, one noticeable drawback of this solution lies in the loss of transaction data, and hence leads to the data unavailability problem. An alternative approach is to deploy light nodes which have been adopted in Bitcoin, Ethereum, Zcash, Byteball [15], and perhaps some other blockchains. The light nodes store only block headers instead of full blocks. This results in that the light nodes heavily rely on the trusted full nodes for the block acquisition and verification, which may not always be a desirable approach.

Recently, novel designs employing coding techniques have been proposed [16]–[18]. Dai *et al.* [16] used network coding, while we consider erasure correcting codes. We believe that both approaches are interesting research topics, since in different applications, they may have their own advantages and disadvantages. We also would like to note that the

Manuscript received October 25, 2019; revised January 29, 2020 and February 24, 2020; accepted March 10, 2020. Date of publication March 19, 2020; date of current version August 12, 2020. This work was supported in part by the Columbia-IBM Blockchain Center and in part by Nakamoto & Turing Labs. *(Corresponding author: Xiaodong Wang.)*

Huihui Wu was with the Department of Electrical Engineering, Columbia University, New York, NY 10027 USA. He is now with the Department of Electrical and Computer Engineering, McGill University, Montreal, QC H3A 0E9, Canada (e-mail: huihui.wu.phd@gmail.com).

Alexei Ashikhmin is with the Communications and Statistical Sciences Research Department, Nokia Bell Laboratories, Murray Hill, NJ 07974 USA (e-mail: alexei.ashikhmin@nokia-bell-labs.com).

Xiaodong Wang is with the Department of Electrical Engineering, Columbia University, New York, NY 10027 USA (e-mail: wangx@ee.columbia.edu).

Chong Li, Sichao Yang, and Lei Zhang are with Nakamoto & Turing Labs, New York, NY 10036 USA (e-mail: chongl@ntlabs.io; ysc@ntlabs.io; leizha@ntlabs.io).

Digital Object Identifier 10.1109/JIOT.2020.2982067

performance evaluation on blockchain platforms is not given in [16]. Raman and Varshney [17] proposed a secure distributed coding scheme combining the components of Shamir's secret key sharing and private key encryptions. However, they suggest to use a black box for all the necessary computations related to the block recovery algorithm, and practical implementation of such a black box is not provided. So, it is even not clear whether such a black box is feasible or not. In contrast, in this article, we explicitly describe all the block encoding and recovery algorithms. In [18], it is proposed to use an erasure code to encode and distribute each individual block, which leads to a reduction of the required memory. The authors suggest to use random linear codes for encoding each block individually and assign to each node only a small part of a codeword. This approach leads to a significant reduction in node memory requirements and allows one to restore a block even if some blockchain nodes are not available for some reason. On the other hand, when a new node joins the blockchain, it needs for each of the previously generated blocks to: 1) collect some symbols from neighboring nodes; 2) conduct decoding in order of recovering the block; and 3) conduct encoding in order to generate a new parity check symbol for this block. All these steps, especially step 2), require large computations. Taking into account that the number of blocks can be very large, this creates significant problems with the practical implementation of this approach. In contrast, in this article, when a new node joins the blockchain, it simply requests some copies of the memories from several neighboring nodes. This looks to be a much more practical approach since it does not require any computations. We also believe that our approach leads to lower storage requirements for each node since we suggest to encode a group of blocks.

In this article, in contrast to the above works, we study the possibility of encoding data across several blocks and using error correction codes that afford simple decoding. Specifically, LDPC codes [19] are taken to describe explicitly the coding scheme. This allows us to reduce the needed computational complexity compared, in particular, with [18], in which inversions of big matrices are required. We design our encoding scheme to minimize the number of nodes that should be contacted and the needed computational complexity for restoring a particular blockchain block. In the Example at the end of Section III-C, we show that the expected values of these two parameters in our scheme are typically quite low. We further propose a new simple account-based model which further reduces the required memory by removing all full nodes. We also suggest to replace the storage of block headers by their hashes.

Another bottleneck in the modern blockchain systems lies in the large network load. In this article, we propose some ideas that may lead to a reduction of the network load. In particular, only the block headers are broadcasted over the network, rather than the whole block.

The contribution of this article can be summarized as follows.

1) A novel error correcting coding-based distributed storage scheme is presented, which can be performed by the full nodes in the current blockchain systems. Moreover, by deploying LDPC codes, the nodes are naturally embedded with error correction and malicious node detection properties.
2) A novel blockchain with all light nodes is proposed for transaction applications using an account-based model. Another assumption in this system is that a communication link between any two nodes can be established. The proposed system reduces both storage cost and communication cost in a blockchain.
3) A novel block verification protocol with the help of the Byzantine fault tolerance (BFT) algorithm [20] is introduced. The new verification protocol involves only a small number of verification nodes, which lowers drastically the verification cost in blockchain systems.

The remainder of this article is organized as follows. Section II gives a brief introduction on the Bitcoin blockchain while Section III presents the details of the proposed distributed blockchain coding scheme. Section IV describes the proposed blockchain with full nodes and Section V proposes a novel blockchain with all light nodes using the account-based model. A novel block verification algorithm involves a small number of nodes that is also provided in this section. Moreover, the performance analysis of the proposed blockchain coding scheme is illustrated in Section VI. Finally, Section VII concludes this article.

II. Preliminaries on Bitcoin Blockchain

The Bitcoin blockchain [1] is a decentralized ledger that is used for organizing the cryptocurrency system in which online payments between any two parties are conducted without a central financial center. This section briefly describes the main part of the Bitcoin protocol.

The Bitcoin ledger consists of blocks, and each block is composed of transactions and a header. Transactions and blocks are generated and processed by nodes, which are some computational devices, e.g., computers and mobile phones. In Bitcoin, there are two categories of nodes—full nodes and light nodes. Full nodes store the entire blockchain (all generated blocks) while light nodes save only the block headers.

Each transaction is created and digitally signed by a node, for instance, a payment from node a to node b would be created and signed by a. Then, it is broadcasted to all nodes over the blockchain network. A full node can conduct mining. Mining consists of putting several transactions together into a block and conducting some computational protocol, which will be discussed below in this section. Next, the block is broadcasted to all full nodes and each of them checks if the block was formed correctly (in particular, if all its transactions are valid), (see [21] and [22]). If the block is valid, then each full node includes it in its ledger. Note that the mining node gets a bitcoin reward for generating a block. This reward plays as a strong incentive for full nodes to conduct mining.

When a light node, for some reason, wants to verify a transaction, it requests information on transactions from some neighboring full nodes and further conducts the simplified payment verification (SPV) protocol, (see [1] for details).

Remark 1: The authors could not find in the literature an incentive for a full node to convey this information to the light node. This is an interesting point since typically all operations in distributed systems should have incentives. What would happen, for instance, if the owner of a full node modifies the Bitcoin software (that runs on this node) so that the node would stop answering light nodes' requests? This can be done, for example, in order to save computational and electrical resources of the node. Since this will not result in any penalty for this node, many other full nodes may follow the same "lazy" behavior, which may result in poor blockchain performance.

The block header of a block contains the following items: version, timestamp, the hash of previous block header, Merkle root, current target, and nonce. We briefly discuss these items as follows.

The "version" field indicates which version of the software is used, "timestamp" field is the time (in UNIX format) when a full node starts mining the block, and the hash of previous block header is the result of hashing output of the previous block header.

We recall that a hash function maps input data of arbitrary size into a sequence of fixed length and any change of the input data will change the output. In Bitcoin, the SHA256 hash function is used, which generates a 256-b output for any input. The very important feature of a hash function is that it is very difficult to invert, i.e., to find an input corresponding to a given output.

In order to produce a block, a full node first selects several valid transactions together, then it forms a Merkle tree by repeatedly hashing pairs of the transactions until there is only one hash left, which is called the Merkle root. More details on the Merkle tree can be found in [1] and [23].

"The current target" is a 256-b sequence that is used to adjust the difficulty of block mining. The lower is the current target, the more difficult it is to find a nonce for a new block. The target adjusts every 2016 blocks in order to make sure that a block is mined in around 10 min.

Finally, a very important component of a block header is the so-called "nonce," which is a 32-b sequence whose value is searched by full nodes. In order to successfully mine a block, each full node competes for solving a difficult Proof-of-Work (PoW) problem (1), i.e., searching for a nonce for the block header, such that the following inequality holds

$$\text{SHA256(SHA256(block header))} < \text{the current target} \quad (1)$$

where the 80-B block header is a concatenation of fields "version" (4 B), hash of previous block header (32 B), Merkle root (32 B), "timestamp" (4 B), "current target" (4 B) and the nonce (4 B). Note that once the valid transactions are selected, the only variable in (1) is the value of "nonce," which needs to be searched.

Next, a generated block is comprised of the block header and the selected transactions and a special *coinbase transaction*. This transaction is a bitcoin reward for the full node that produced the block. This reward serves as an incentive for a full node to conduct the mining.

Once a block is produced, the full node broadcasts it over the network. When it arrives at a full node, the node will verify using a predefined set of rules, (see [22] for details), before adding it to its own blockchain. Note that full nodes have a strong incentive for conducting the verification. Indeed, assume that the block was not valid, but a node a skipped the verification and simply included the block into its blockchain. However, other nodes make verification, figure out that block is invalid, and do not include it into their blockchains. Thus, node a will have a blockchain that is different from blockchains of other nodes and this will preclude node a from subsequent mining in the future.

It is also worth noting that if two full nodes generate two blocks A and B at the same time, then some full nodes include block A on their blockchains while others include block B. However, as time goes on, only one blockchain containing either block A or block B will be the longest, which will be considered as the valid blockchain. If say, block A happened to belong to the longest blockchain, then the branch that starts at B becomes an orphan and no more new blocks will be appended to it.

To summarize, the blocks are linked together to form a chain secured by the PoW, which assures that large computing resources are invested in building such a chain. Any attempt to modify anything recorded in the chain would, therefore, require even more computing resources than those that have already been expended. Moreover, only those transactions included in a block and further linked into the longest blockchain are eventually being recorded in the public ledger.

It is easy to see that the Bitcoin blockchain has a huge redundancy, since each full node (and there are a large number of these nodes) keeps the full blockchain, so multiple copies of the same blockchain are stored. This dramatically increases the storage cost at the full nodes and limits the large-scale application of the blockchain. In the following sections, we develop countermeasures for this storage problem.

III. Reduction of Required Memory at Nodes

A. Distributed Block Encoding and Block Recovering

This section presents the proposed distributed blockchain coding scheme with generic error correction codes. There are many different powerful families of codes capable of recovering erasures. In particular, one can use algebraic codes, e.g., Bose–Chaudhuri–Hocquenghem (BCH) and Reed–Solomon codes, convolutional codes, iteratively decodable codes, such as LDPC and Turbo codes, as well as modern polar codes and many others. In this article, we first formulate results assuming generic erasure recoverable codes, followed by the application of LDPC codes in full detail. We start by defining some necessary notations.

First, assume that the total number of nodes in the blockchain is n and these nodes are labeled with indices $\{1, 2, \ldots, n\}$. The indices can be assigned to nodes according to their blockchain joining timestamps. We assume that blocks in the ledger are enumerated by integers, so $B^{(j)}, j \geq 1$, is the jth block. We combine each set of t consecutive blocks into groups $G_m = [B^{((m-1)t+1)}, \ldots, B^{(mt)}], t \geq 1$, where m is

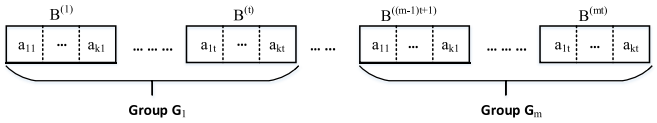

Fig. 1. Illustration of comprising a group of blocks.

the index of the group. We will say that a linear code over the finite field $\mathbb{F}_q$ that encodes t symbols $(\phi_1, \ldots, \phi_t)$ into N symbols $(v_1, \ldots, v_N)$ is an $[N, t]_q$ code. We will always use systematic encoding, that is $v_i = \phi_i$ for $i = 1, \ldots, t$.

It is well known that the performance of long error correcting codes is better than the performance of short codes. For this reason, we assume $N \approx n$. As we will show in Section III-C (see Algorithm 2), the complexity of restoring only one particular block does not depend on N, thus keeping N large improves the performance and does not increase the decoding complexity. The encoding complexity grows with N, but since it is small in any case, the encoding of codes with large N should not be a problem for practical realizations. Note that the number of nodes n can change, but it is not mandatory to keep $N = n$. If at some moment $N < n$, then simply some $n-N$ nodes will not obtain any symbols of a codeword. Other solutions are also discussed in Section VI-G. If $N > n$, then this is also not a problem for our scheme, as we explain this in the sequel. In the rest of this article, to make the presentation shorter, we assume that $N \geq n$.

We represent each block of G_m by k symbols over $\mathbb{F}_q$. We assume that $k \log_2(q)$ is larger or equal to the bit size of any block. If for some block we need only $k' < k$ symbols, then for the $k-k'$ tail symbols we use zeros. Hence, each G_m contains exactly kt symbols. This procedure is shown in Fig. 1, in which the symbols are denoted by a_{ij}, $1 \leq i \leq k, 1 \leq j \leq t$. Note that these symbols of course are different for different groups. So strictly speaking for group G_m, we had to write $a_{ij}(m)$, but in what follows, in order to make notation short, we drop the index m.

Next, we arrange these kt symbols of group G_m into a k-by-t matrix, such that the jth column contains symbols that form the jth block of G_m, i.e.,

$$\begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1t} \\ a_{21} & a_{22} & \ldots & a_{2t} \\ \ldots & \ldots & \ldots & \ldots \\ a_{k1} & a_{k2} & \ldots & a_{kt} \end{bmatrix}. \tag{2}$$

Further, we use a systematic encoder of our $[N, t]_q$ code to encode each row in (2). The resulted codewords are rows of the following matrix:

$$\begin{bmatrix} v_{11} & v_{12} & \ldots & v_{1t} & \ldots & v_{1n} \ldots & v_{1N} \\ v_{21} & v_{22} & \ldots & v_{2t} & \ldots & v_{2n} \ldots & v_{2N} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ v_{k1} & v_{k2} & \ldots & v_{kt} & \ldots & v_{kn} \ldots & v_{kN} \end{bmatrix}. \tag{3}$$

Note that for $1 \leq j \leq t$, column $\mathbf{v}_j$ is the jth block of G_m [24]. This encoding step can be implemented either by a single node or by each full node independently. In the first case, the node performing the encoding will distribute triplets $(\mathbf{v}_j, j, m)$, $1 \leq j \leq n$, to the corresponding nodes. In the second case, each node, say node j, performs independently the encoding of $\mathbf{v}_j$ and keeps the triplet $(\mathbf{v}_j, j, m)$, and it does not compute other $\mathbf{v}_i$, $i \neq j$. This drastically reduces the required memory. More details will be elaborated in Sections IV and V.

Algorithm 1 Recovery of a Group G_m Containing Block $B^{(i)}$

1: Node j enlarges the set L by randomly adding some nodes $i_1, \ldots, i_u$ to set L, which are still not in L;
2: Node j requests triplets $(\mathbf{v}_{i_a}, i_a, m), a = 1, \ldots, u$, from these nodes and those nodes that are active send their triplets to node j;
3: Node j tries to decode k codewords that form the matrix (3), treating missing symbols (the symbols kept by nodes that have not been contacted yet, or symbols of nonactive nodes) as erasures;
4: If all k decodings are successful, then this means that node j has reconstructed the entire group G_m. If some of the decodings are not successful then we go to Step 1;

In a similar way, the aforementioned block encoding algorithm can also be applied for encoding a group of t_h block headers. As the size of a block header is smaller than that of a block, and hence $t_h \geq t$ block headers can be put into group G'_m.

Note that with this scheme, a particular node, say node j, has in its possession only columns $\mathbf{v}_j$ of (3) (one column for each group $G_m, m \geq 1$). If node j needs block $B^{(i)}$, it can use the following block recovery method.

First node j computes indices $m = \lceil (i/t) \rceil$ and $r = i - \lfloor (i-1)/t \rfloor t$. These indices show that $B^{(i)}$ is contained in vector $\mathbf{v}_r$ of group G_m. If $r = j$, then node j does not have to do anything else, since it already has in its memory this $\mathbf{v}_r$. If $r \neq j$, then node j broadcasts a request for the triplets $(\mathbf{v}_r, r, m)$. If node r is reachable and if node j trusts it, then it gets the required block $B^{(i)}$ from node r. However, if for some reason, node r is not active (and so not reachable) or node r is compromised, then $B^{(i)}$ has to be recovered in another way. In this situation, the node j forms the set of nodes $L = \{j\}$ and conducts Algorithm 1.

Note we are interested in keeping the size of the final set L being as small as possible, since it means that only a small number of $\mathbf{v}_{i_a}$ will be sent to node j and therefore the network load will be kept small. More discussions will be given in Section VI-H.

Example: Fig. 2 depicts the proposed Algorithm 1 with a simple example. Assume an error correction code with parameters $[N, t]_q = [10, 3]_q$ is utilized and the number of nodes in the blockchain is $n = 10$. Moreover, assume that the size of each block in the blockchain is less than $\log_2 q$ bits (this may not hold in real blockchains, but only used for this example), which implies $k = 1$, i.e., each block can be represented by one symbol. Moreover, each group contains $t = 3$ blocks, and hence matrices (2) and (3) shrink to row vectors with dimensions of 1×3 and 1×10, respectively. Let node 4 want to recover block $B^{(6)}$. So it computes $m = \lceil (6/3) \rceil = 2$ and $r = 6 - (\lfloor (6-1)/3 \rfloor)3 = 3$ and broadcasts request for vector $\mathbf{v}_3$ from group G_2. Let us assume, however, that node 3 is not active or reliable.

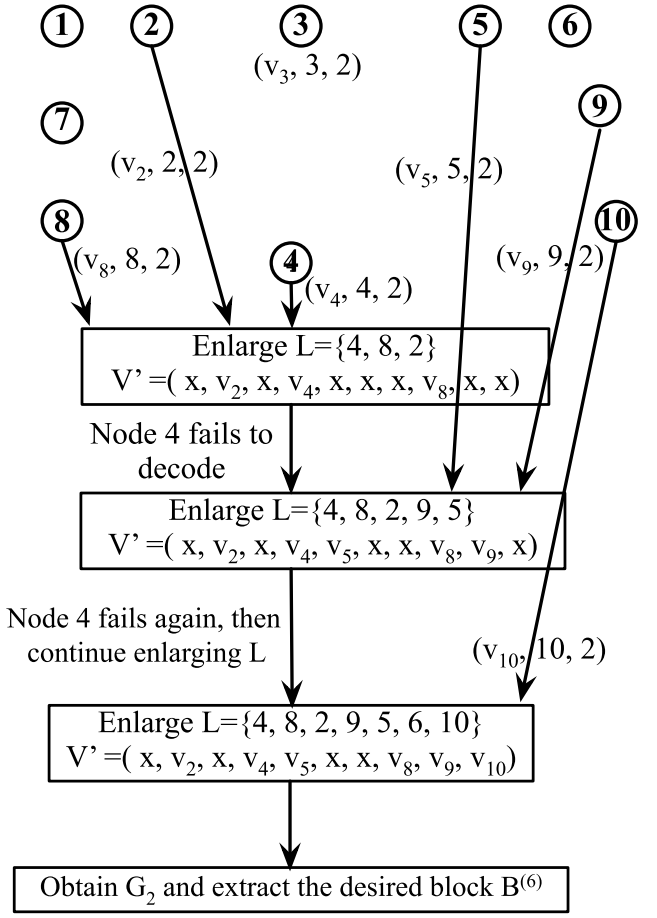

Fig. 2. Illustration of the proposed recovery Algorithm 1, where "**x**" denotes an unavailable symbol.

Then, node 4 randomly enlarges the set by adding two nodes as $L = \{4\} \cup \{8, 2\}$, and nodes 8 and 2 return the corresponding triplets $(\mathbf{v}_8, 8, 2)$ and $(\mathbf{v}_2, 2, 2)$ (these triplets are shown in Fig. 2) to node 4 for assembling $\mathbf{V}'$ (this is shown in the first rectangle in Fig. 2, where "**x**" denotes an erasure). Subsequently, node 4 tries to decode $\mathbf{V}'$, but it fails to reconstruct G_2.

Next, node 4 enlarges the set L again by contacting other two nodes, e.g., $L = \{4\} \cup \{8, 2\} \cup \{9, 5\}$ and updates $\mathbf{V}'$, as shown in the second rectangle in Fig. 2. However, node 4 still fails to decode $\mathbf{V}'$. Subsequently, node 4 continues enlarging the set L as $L = \{4\} \cup \{8, 2\} \cup \{9, 5\} \cup \{6, 10\}$. We assume that node 6 is not active, and so node 4 gets only triplet $(\mathbf{v}_{10}, 10, 2)$. The updated vector $\mathbf{V}'$ is shown in Fig. 2. This time node 4 successfully recovers the desired group G_2. Finally, node 4 extracts $\mathbf{v}_3$ from G_2.

Note that in Algorithm 1, in order to reconstruct one block, node j has to reconstruct the entire group G_m that contains that block, which looks being a waste of network resources. In the sequel, we take LDPC codes as examples, in order to illustrate the proposed block encoding and recovery algorithms in detail. We will show in Section III-C that LDPC codes allow us to contact only those nodes that with high probability will allow reconstruction of only the needed block. This drastically reduces the time and computational complexity needed for the reconstruction and the network load.

Before ending this section, let us elaborate more on the needed code parameters. The code rate of an $[N, t]_q$ code is defined as $R = t/N$. Let us assume that our code can recover a code vector if at least $\alpha t, \alpha > 1$, code symbols are not erased. The value of α depends on the choice of our code. In particular, one can make α being arbitrarily close to 1 by choosing a sufficiently long LDPC code. Let us further assume that the number of nodes and the code length is the same, i.e., $n = N$, and that with a high probability not more than the fraction ϵ of n nodes are not active. Then, with high probability our code will recover all k codewords in (3) if $N(1-\epsilon) \geq \alpha t$, and hence the code rate should satisfy $R \leq (1-\epsilon)/\alpha$.

To get an idea on reasonable values for k and q, we note that if the size of a block is in the range $[k_{low}, k_{up}]$ in terms of symbols over $\mathbb{F}_q$, then it makes sense to take $k = k_{up}$. It is well known that the size of any Galois field is a power of a prime number. For real-life implementation, it is convenient to use $q = 2^b$, where b is an integer. Below we consider two examples of Bitcoin and Ethereum blockchains.

It is known that a block size limit of 1 MB in the Bitcoin system was introduced by Satoshi Nakamoto in 2010. Currently, a block size limit is upper bounded by 4 MB with the segregated witness (SegWit) soft fork in the Bitcoin system [25]. Therefore, we have $k_{up} = 4 \times 2^{23}/b$ symbols over $\mathbb{F}_{2^b}$ for Bitcoin blockchain. In the Ethereum blockchain, there is no block size limit. By observing the average block size in Ethereum [26], it can be noted that there is no block size exceeding 35 KB currently. Consequently, the value $k_{up} = 35 \times 2^{13}/b$ symbols over $\mathbb{F}_{2^b}$ can be utilized in the Ethereum blockchain. For choosing a meaningful value for q, we first note that typically the larger is q the better is the code performance. However, when q is already large, any additional increase of it hardly improves the code performance. So, we think that devoting one or two bytes for q is a good choice. Thus, we take $q = 2^8$ or 2^{16}. This leads us to $k = 4 \times 2^{20}$ or 4×2^{19} for Bitcoin and $k = 35 \times 2^{10}$ or 35×2^9 for Ethereum blockchains.

Additionally, we would also like to point out that the use of LDPC codes allows us to find the best tradeoff between the size q of Galois field and the number k of codewords that are used to encode one group of blocks, for the purpose of minimizing the overall complexity. This is based on the following observation.

Assume we have two LDPC codes defined by the same parity-check matrix $\mathbf{H}$ (i.e., by the same Tanner graph), one is a binary LDPC code C_1 while the other one is an LDPC code C_2 over $\mathbb{F}_q$. It is known that the decoding performance of C_2 would beat C_1, if we use maximum likelihood decoding. However, if the message passing decoding algorithm is utilized for an erasure channel, which is the case in the proposed blockchain coding scenario, then this decoder has identical performance to the so-called peeling decoder [27]. Therefore, the probability of decoding error does not depend on the alphabet size q of a Galois field, and thus the codes C_1 and C_2 will have the same error correcting performance. As a consequence, if we choose q being small then k grows and *vice versa*. Thus,

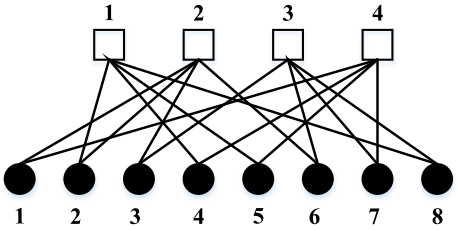

Fig. 3. Tanner graph representation of the parity-check matrix in (4).

one can find an optimal tradeoff between the values of q and k for minimization of the decoding complexity, which depends on both of these parameters.

B. Preliminary on LDPC Codes

This section gives a brief introduction to LDPC codes and their application in the proposed blockchain encoding. An algorithm of recovering an individual block is provided in Section III-C.

An $[N, t]_q$ LDPC code is defined by an $(N-t) \times N$ low-density parity-check matrix $\mathbf{H} \in \mathbb{F}_q^{(N-t)\times N}$. A vector $\mathbf{c} \in \mathbb{F}_q^N$, is a codeword if and only if $\mathbf{Hc} = \mathbf{0}$. It is convenient to associate with $\mathbf{H}$ a bipartite graph consisting of a set of N variable nodes and $N-t$ check nodes. Variable nodes are connected by edges with check nodes. Each column of $\mathbf{H}$ corresponds to a variable node and each row in $\mathbf{H}$ corresponds to a check node. If the entry $h_{ij} \neq 0$, then the jth variable node is connected to the ith check node by an edge and that edge has label h_{ij}.

The weight of a column (row) in $\mathbf{H}$ is defined as the number of nonzero elements in the column (row). An LDPC code is said to be (d_v, d_c) regular if the weights of all columns and the weights of all rows are d_v and d_c, respectively.

An example of a 4×8 parity-check matrix $\mathbf{H}$ over the finite field $\mathbb{F}_4$ of a code with $R = 1/2$ is given in (4). The finite field $\mathbb{F}_4$ consists of elements $\{0, 1, \Omega, \bar{\Omega}\}$, and summation and multiplication rules for those elements (see [28] for details on finite fields).

It can be noted that the weight of each column is 2 while the weight of each row is 4, and thus this is $(2, 4)$ regular code.

$$\mathbf{H} = \begin{bmatrix} 0 & \Omega & 0 & \Omega & \Omega & 0 & 0 & \Omega \\ \Omega & \bar{\Omega} & \Omega & 0 & 0 & \bar{\Omega} & 0 & 0 \\ 0 & 0 & \bar{\Omega} & 0 & 0 & \Omega & \bar{\Omega} & \bar{\Omega} \\ \bar{\Omega} & 0 & 0 & \bar{\Omega} & \bar{\Omega} & 0 & \Omega & 0 \end{bmatrix}. \tag{4}$$

The matrix in (4) can also be represented by a Tanner graph. Fig. 3 shows the Tanner graph corresponding to (4), where squares denote the check nodes and circles represent the variable nodes.

The Tanner graph can be used for iterative decoding algorithms, such as belief propagation or message passing decoding. These algorithms have low complexity, typically $O(N \log N)$, and provide low decoding error probability, (see [29] for decoding algorithms). Additionally, more information regarding the designs and applications of LDPC codes can be found in [30]–[32] and references therein.

It is important to note that irregular LDPC codes are much more efficient than the regular LDPC codes. An irregular LDPC codes has various column or row weights in the parity-check matrix $\mathbf{H}$, and it is defined by the degree distributions $\lambda(x)$ and $\rho(x)$

$$\lambda(x) = \sum_{i=2}^{d_v} \lambda_i x^{i-1}, \quad \rho(x) = \sum_{i=2}^{d_c} \rho_i x^{i-1}$$

where λ_i (respectively ρ_i) denotes the fraction of the edges incident to variable (respectively check) nodes of degree i. For instance, the $(2, 4)$ regular LDPC code in (4) can also be represented by $\lambda(x) = x$ and $\rho(x) = x^3$.

An important subclass of irregular LDPC codes is formed by protograph-based LDPC codes. A protograph $\mathbf{B}_H$ is a Tanner graph with a relatively small number of variable nodes and check nodes, and the parity-check matrix $\mathbf{H}$ of the final LDPC code is constructed from $\mathbf{B}_H$ in a "copy-and-permute" way. First, $\mathbf{B}_H$ is copied l times and then the edges of each copy are permuted among all the l copies. More information about protograph-based LDPC codes can be found in [32] and references therein, and the simulation results will be given in Section VI-H.

As we noticed already in Section III-A, a properly designed LDPC code can recover all t information symbols if αt code symbols are not erased. In particular, an optimized choice of $\lambda(x) = x$ and $\rho(x) = x^3$ leads to LDPC codes with $\alpha \to 1$ as the code length N tends to infinity, (see [31], [33]). So for recovering all blocks of a group G_m, a particular node, say node c, should contact arbitrary set of αt nodes and collect their triplets $(\mathbf{v}_j, j, m)$. Next, node c forms k vectors, say $\mathbf{y}_i, i = 1, \ldots, k$, using the obtained symbols $v_{ij}, i = 1, \ldots, k$, and using erasures x for missing symbols and decodes vectors $\mathbf{y}_i$ by an iterative decoding of the LDPC code. If decodings of $\mathbf{y}_i$ are unsuccessful, node c will request some additional triplets $(\mathbf{v}_j, j, m)$ from nodes that have not been contacted yet.

It is interesting to note that since erasures x in vectors $\mathbf{y}_i$ appear on exactly the same positions, one can show (we omit details) that either all k decodings will be successful or they all will fail. Hence, node c may first conduct decoding of only $\mathbf{y}_1$ and conduct other $k-1$ decodings only if it is successful. This will greatly reduce the computational complexity.

C. Block Encoding and Individual Block Recovering

Often a node does not need all t blocks from a group G_m. Instead it may need block $B^{(a)}$ from G_r and block $B^{(j)}$ from G_m, and so on. So, it looks natural to try to find a decoding approach for this task that would have significantly smaller complexity than the decoding of an entire group G_m. Below we present such an approach in the case of LDPC codes.

Using Gaussian elimination and column permutation one can transform matrix $\mathbf{H}$ to the systematic form $\mathbf{H}' = [\mathbf{P}^T \ \mathbf{I}_{N-t}]$ and further find a generator matrix of this LDPC code in the systematic form $\mathbf{G} = [\mathbf{I}_t \ \mathbf{P}]$. The generator matrix $\mathbf{G}$ allows one to conduct a systematic encoding of vector $\mathbf{a}_i = [a_{i1}, \ldots, a_{it}], \ i = 1, \ldots, k$, as

$$(v_{i1}, \ldots, v_{iN}) = \mathbf{a}_i \mathbf{G}, i = 1, \ldots, k. \tag{5}$$

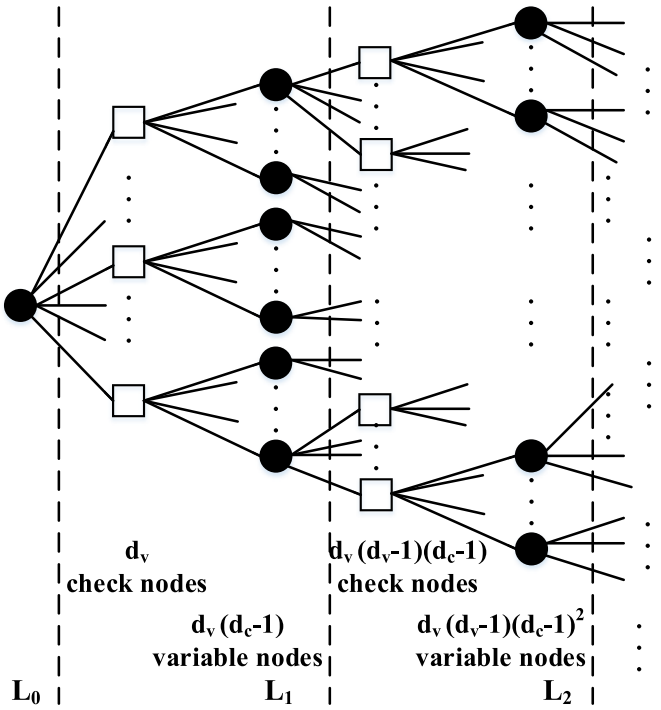

Fig. 4. Expanded Tanner graph for a (d_v, d_c) regular LDPC, in terms of one coded symbol.

These vectors are the rows of matrix (3). If the encoding is conducted by each full node independently from others, node j simply computes $\mathbf{v}_j = \mathbf{a}_i\mathbf{g}_j$, where $\mathbf{g}_j$ is the jth column of $\mathbf{G}$.

As we discussed in Section III-A, recovering a single block is equivalent to recovering the appropriate column in (3).

Recall that an LDPC code can be represented by a Tanner graph, and let us consider a (d_v, d_c) regular LDPC code. Let us take the jth variable node and set $L_0 = \{j\}$ (we will say that L_0 is the 0th extension layer). We will use this variable node as a root to expand the graph as follows. First, we follow the edges that are incident to this variable node and come to d_v check nodes and use their indices to form extension layer L_1. Next, we follow the edges that are incident to the check nodes from L_1, excluding the edges coming from layer L_0, and come to $d_v(d_c-1)$ variable nodes and include their indices into extension layer L_1. We proceed in this way further and form layers L_2, L_3, and so on. This procedure is shown in Fig. 4. It is not difficult to show that the number of check nodes $N_c(r)$ and variable nodes $N_v(r)$ in L_r are

$$N_c(r) = d_v(d_c-1)^{r-1}(d_v-1)^{r-1} \tag{6}$$

$$N_v(r) = d_v(d_c-1)^{r}(d_v-1)^{r-1}. \tag{7}$$

Recall that a cycle in the bipartite graph is a path that starts and finishes at the same variable node. The length of a cycle is equal to the number of the involved edges. Good LDPC codes do not have short cycles. It is easy to see that if our LDPC code does not have cycles of length $2r$, then all variable nodes from layers $L_0,\ L_1, \ldots, L_r$ are distinct.

Let us assume that node c needs to recover the $((m-1)t+j)$th block, that is the jth block in G_m. Recall that this is equivalent to recovering symbols $v_{ij},\ 1 \le i \le k$ in matrix (3). In Algorithm 2, we consider how this can be done for $i = 1$. For all other i's, the algorithm is exactly the same.

Algorithm 2 Recovery of Symbols v_{1j}

Node c expands the Tanner graph using $L_0 = \{j\}$;
Node c broadcasts request for v_{1j};
if Node j is accessible **then**
  Node c gets v_{1j};
  Return;
else
  Node c sets the counter $r = 0$;
  while The value v_{1j} is not recovered **do**
    Node c sets $r \leftarrow r + 1$ and broadcasts requests for the symbols v_{1u} with $u \in L_r$;
    Node c gets symbols v_{1u} only for those $u \in L_r$ that are accessible (perhaps not all nodes from L_r);
    Node c uses the received v_{1u} to form set S_r;
    Node c uses symbols from $S_1, \ldots, S_r$ as input for the **Single Symbol Decoding Algorithm** defined in Algorithm 3;
    if The value v_{1j} is reconstructed **then**
      Break;
    end if
  end while
end if

Below we present an algorithm of recovering only a single coded symbol which corresponds to recovering one block from a group G_m. This single symbol decoding algorithm is similar to the standard erasure decoder of LDPC codes [27]. However, it has a smaller complexity. Since this algorithm is important for blockchain applications, we present it in detail. We will use Fig. 4 to give the following informal description of it.

We will say that $d_c - 1$ edges on the right-hand side of a check node of degree d_c are inputs and the single edge on its left-hand side is the output. We will use similar terminology for variable nodes. We further assume that all the variable and check nodes are enumerated and that the jth variable node corresponds to the jth column of $\mathbf{H}$ and the ith check node corresponds to the ith row of $\mathbf{H}$. The single symbol decoding algorithm is described in Algorithm 3.

Note that the complexity of this decoding is small if r is small. The complexity does not exceed

$$C(r) = \sum_{j=1}^{r} N_c(r)(d_c - 2) \tag{8}$$

summations over $\mathbb{F}_q$, where $N_c(r)$ is defined in (7). Additionally, we also need d_v multiplications at each check node of degree d_c and one multiplication at each variable node. If we assume that the exponential form of elements of $GF(q)$ is used, then multiplications have very low complexity. Hence, in what follows we will count only summations.

Note that with the help of the special format in (3), when node c requests the symbols in set L_{r+1}, the corresponding nodes can send the entire column containing the requested symbols [recall that each node stores one column of the matrix in (3)]. In such a way, the desired block can be recovered by

Algorithm 3 Single symbol decoding algorithm

INPUT: $S_1, \ldots, S_r$;
for Layer L_r **Down to** L_1 **do**
  Node c uses symbols from S_r, the other symbols corresponding to non-active blockchain nodes are replaced with erasure x;
  for Each check node i **do**
    Denote by $c_{i,1}, \ldots, c_{i,d_c-1}$ and by c_{i,d_c} the nonzero entries of the i-th row of $\mathbf{H}$ corresponding to the inputs of the node, and the output of the node respectively;
    if All $d_c - 1$ inputs, say $v_1, \ldots, v_{d_c-1}$, of this check node are not erasures **then**
      Compute the output as $w = (\sum_{t=1}^{d_c-1} c_{i,t} v_t)/c_{i,d_c}$;
    else
      Set $w = x$;
    end if
  end for
  for Each variable node j **do**
    Denote by $c_{j,1}, \ldots, c_{j,d_v-1}$ and by c_{j,d_v} the nonzero entries of the j-th column of $\mathbf{H}$ corresponding to the inputs of the node, and the output of the node respectively;
    if At least one input, say w_t, of the $d_v - 1$ inputs, $w_1, \ldots, w_{d_v-1}$, is not erased **then**
      Set the output as $v = w_t/c_{j,t}$;
      /* it is not difficult to show that all non erased input symbols always have the same value */
    else
      Set $v = x$;
    end if
  end for
end for

contacting a small number of nodes but performing the LDPC decoding algorithm k times.

Similar to the algorithm of reconstructing the whole group of blocks, we are interested in maintaining the value of r as small as possible. In other words, expand as less as possible the layers of the Tanner graph for a given root, since this is equivalent to keeping low network load. The following lemma shows how many nodes we typically have to connect and how much computations we have to do in order to recover an individual block.

Lemma 1: We assume that all blockchain nodes have the same level of unavailability and that the nodes are independent. If a given (d_v, d_c) regular LDPC code does not have cycles of length less than or equal to r, and if the probability of a blockchain node being unaccessible is ϵ, then the probability that we will need to contact more than $N_r = 1+\sum_{y=1}^{r} d_v(d_v-1)^{y-1}(d_c-1)^y$ nodes can be expressed recursively as

$$P^{(0)} = \epsilon,\ P^{(r)} = \epsilon\left(1 - \left(1 - \hat{P}^{(r-1)}\right)^{d_c-1}\right)^{d_v},\ r \geq 1 \quad (9)$$

where

$$\hat{P}^{(0)} = \epsilon,\ \hat{P}^{(r)} = \epsilon\left(1 - \left(1 - \hat{P}^{(r-1)}\right)^{d_c-1}\right)^{d_v-1},\ r \geq 1.$$

The expected number E_{nodes} of nodes that should be contacted and the expected number of needed summations E_{cmplxt} are

$$E_{\text{nodes}} = 1 - \epsilon + \sum_{r=2}^{\infty}\left(P^{(r-1)} - P^{(r)}\right)N_r,$$
$$E_{cmplxt} \quad = \sum_{r=2}^{\infty}\left(P^{(r-1)} - P^{(r)}\right)C(r)$$

where $C(r)$ is defined in (8).

Proof: According to the check and variable nodes processing described in Algorithm 3, we have that if p is the probability of erasure at the input of check (variable) node then

$$\Pr(\text{erasure at the output of check node}) = 1 - (1-p)^{d_c-1} \quad (10)$$

$$\Pr(\text{erasure at the output of variable node}) = p^{d_v-1}. \quad (11)$$

For the root node, which has d_v inputs, we have

$$\Pr(\text{erasure at the output of variable node}) = p^{d_v}. \quad (12)$$

The probability of erasure of the root is ϵ, hence we have $P^{(0)} = \epsilon$. If we start with L_1, then according to (12) and (10), we have $P^{(1)} = \epsilon(1-(1-\epsilon)^{d_c-1})^{d_v-1}$. Here ϵ can be thought of as the probability of erasure at the output of a variable node of L_1. Hence, we have

$$P^{(1)} = \epsilon\left(1 - \left(1 - \hat{P}^{(0)}\right)^{d_c-1}\right)^{d_v}.$$

Next, if we start with L_2, then for the same reasons

$$P^{(2)} = \epsilon\left(1 - \left(1 - \hat{P}^{(1)}\right)^{d_c-1}\right)^{d_v}$$

where $\hat{P}^{(1)}$ is the probability of erasure at the output of a variable node of L_1. Using (10) and (11), we get that this probability is

$$\hat{P}^{(1)} = \epsilon\left(1-(1-\epsilon)^{d_c-1}\right)^{d_v-1} = \epsilon\left(1 - \left(1 - \hat{P}^{(0)}\right)^{d_c-1}\right)^{d_v-1}.$$

Assuming further, in the same way, that we start with layers L_3, L_4, and so on, we obtain (9).

Using (9), we get that

$$Q^{(r)} = \Pr(\text{we have to start decoding from } L_r)$$
$$= P^{(r)} - P^{(r+1)}.$$

The probability that the root is not erased itself is $1-\epsilon$. These facts lead to the expression for E_{nodes}. Arguments for E_{cmplxt} are similar. ■

Example: The above lemma can be interpreted as the fact that with high probability, the desired symbol can be recovered by contacting a small number of auxiliary nodes in the blockchain. Table I shows the probability that we need to contact more than r layers (i.e., more than N_r nodes) for recovering a given symbol in the case of a regular (3, 4) LDPC code for different values of ϵ. In particular, if $\epsilon = 0.2$, then the probability that node c will need to contact more than $N_2 = 64$ nodes is 5.05×10^{-4}. Note that the value $\epsilon = 0.2$ is equivalent to saying that 20% nodes in the network are not accessible, which is a very large fraction. In real life, we expect that ϵ

TABLE I
PROBABILITY THAT WE NEED TO REQUEST MORE THAN r LAYERS, I.E., REQUESTING DATA FROM MORE THAN N_r NODES, FOR SUCCESSFULLY RECOVERING A DESIRED SYMBOL, USING A $(3, 4)$ REGULAR LDPC CODE

ϵ or $P^{(0)}$	N_0	$P^{(1)}$	N_1	$P^{(2)}$	N_2	$P^{(3)}$	N_3	$P^{(4)}$	N_4
0.01	1	2.62×10^{-7}	10	1.85×10^{-16}	64	9.28×10^{-35}	388	0	2332
0.15	1	8.62×10^{-3}	10	4.22×10^{-5}	64	1.08×10^{-9}	388	7.08×10^{-19}	2332
0.2	1	2.32×10^{-2}	10	5.05×10^{-4}	64	2.73×10^{-7}	388	8.02×10^{-14}	2332
0.25	1	4.83×10^{-2}	10	3.05×10^{-3}	64	1.51×10^{-5}	388	3.86×10^{-10}	2332

TABLE II
EXPECTED NUMBER OF NODES THAT SHOULD BE CONTACTED IN ORDER TO RECOVER A PARTICULAR BLOCK $B^{(j)}$ AND THE EXPECTED NUMBER OF SUMMATIONS OVER $\mathbb{F}_q$ NEEDED FOR THIS RECOVERY IN THE CASE OF A $(3, 4)$ REGULAR LDPC CODE

ϵ	E_{nodes}	E_{cmplxt}
0.01	1.09	$0.06k$
0.15	2.83	$2.82k$
0.2	4.22	$6.87k$
0.25	6.07	$12.78k$

will be significantly smaller. For instance, when $\epsilon = 0.01$, the probability that node c will need to contact more than $N_1 = 10$ nodes is only 2.62×10^{-7}.

Furthermore, Table II shows the expected number of auxiliary nodes and the expected number of needed summations for different values of ϵ. One can see that on average those numbers are quite small. As we wrote in Section I, this was one of the main goals addressed in this article.

More numerical results can be found in Section VI-H.

IV. PROPOSED BLOCKCHAIN WITH FULL NODES

This section addresses the application of the proposed blockchain encoding scheme into the current blockchain systems. Specifically, full node j can perform the proposed distributed coding scheme and keeps only symbols $v_{ij},\ 1 \le i \le k$, instead of storing all blocks of the corresponding group G_m.

We assume that each full node is equipped with an encoder and decoder of the selected $[N, t]_q$ LDPC, and hence it is capable to generate the required parity-check and generator matrices $\mathbf{H}$ and $\mathbf{G}$. We also assume that there is a mechanism for assigning an index $j,\ 1 \le j \le n$ to each full node according to its timestamps of joining the blockchain.

Similar to the standard Bitcoin blockchain, we assume that each block is broadcasted over the entire blockchain. Suppose that at some moment group G_{m-1} is encoded, as it was described in the previous sections, and blocks $B^{((m-1)t+1)},\ B^{((m-1)t+2)}, \ldots$, are continued being generated and broadcasted to all full nodes. Each full node keeps all those blocks in its memory. At some moment, a full node, say node c, generates and broadcasts block $B^{(mt)}$. At this moment, all full nodes receiving this block realize that it is time to perform the encoding algorithm for group G_m. Hence, each full node individually conducts the encoding, and keeps only one triplet $(\mathbf{v}_j, j, m)$ and further gets rid of blocks $B^{((m-1)t+1)}, \ldots,\ B^{((m-1)t+t)}$. Moreover, a group of t_h block headers can be encoded and distributed in the same way.

In real life, some delays with block propagation to different nodes are possible. For this reason, to ensure the same enumeration of blocks in blockchains of different nodes, the above protocol can be shifted back in time. For instance, we can start to perform the encoding algorithm for group G_m only when block $B^{((m+1)t)}$ is generated (not $B^{(mt)}$). This delay will ensure that all nodes keep blocks $B^{((m-1)t+1)}, \ldots, B^{(mt)}$ in the same place of their blockchains.

Each full node also stores the hashes of block headers, i.e., $h^{(i)} = \text{hash}(H^{(i)})$, where $H^{(i)}$ is the header of block $B^{(i)}$. The verification rules of a new block can be found in [22], including one important step of checking if there exists a duplicate of new block $B^{(i)}$. By keeping all the hashes of previous block headers, i.e., $h^{(j)},\ j < i$, the above checking procedure can be done by detecting if any value of $h^{(t)},\ t < i$ equals to the value of $h^{(i)}$. If $h^{(t)} \ne h^{(i)}\ \ \forall t < i$, then it is believed that there is no duplicate of block $B^{(i)}$. However, if it happens that $h^{(d)} = h^{(i)}$, for some $d,\ d < i$, then block $B^{(d)}$ needs to be recovered using Algorithm 2 in Section III-C, in order to compare the detailed contents of $B^{(d)}$ and $B^{(i)}$.

Note that in Bitcoin-type blockchain systems, the unspent transaction output (UTXO) model is utilized, and the verification of a specific transaction involves multiple blocks. To do so, one has to recover all the relevant blocks. It is worth noting that only small expected numbers of auxiliary nodes and summation operations are required to recover a single symbol, as shown in Table II. A fact implies that the extra complexity of recovering an individual block is small. On the other hand, using the proposed block encoding and recovery algorithms, this extra complexity exchanges a considerable reduction of nodes' storage requirements. This article presents a tradeoff between the coding complexity and storage complexity, and more efficient encoding and recovery algorithms are to be investigated.

For a blockchain with all light nodes, considered in the following section, we propose to use the account-balance table. The use of this table allows simple transaction verification without checking all the previous transactions of a particular node.

V. PROPOSED BLOCKCHAIN WITH ALL LIGHT NODES

This section investigates a novel blockchain system that consists of only light nodes, utilizing the account-based blockchain model. We assume that each node simply maintains an account-balance table. If some node, say node c, generates and signs a transaction that involves the transfer of b bitcoins (or simply coins, if this is not Bitcoin blockchain) to node d, then at the moment of including this transaction

into a block, blockchain nodes check that the cth entry of the account-balance table is larger or equal to b. If this is not the case, such a transaction will be considered as invalid and not allowed for including into a block.

Another assumption of this blockchain is that a communication link between any two nodes can be established. Similar to the previous sections, we assume that all the nodes are labeled with increasing indices $1, 2, \ldots, n$, in terms of their blockchain joining timestamps.

A. System Description

Assume that a node a generates a block $B^{(i)}$ at some time, $((m-1)t+1) \leq i < mt$, and keeps $B^{(i)}$ in its memory. Node a further sends $B^{(i)}$ only to M verification nodes (a way for choosing such nodes is presented in Section V-B). At the same time, the block header $H^{(i)}$ will be broadcasted to the entire network, that is, to all n nodes. Each node, across the blockchain, waits for the verification results from M verification nodes and determines whether the block header $H^{(i)}$ should be accepted or not. If $H^{(i)}$ is accepted, then only its hash $h^{(i)}$ is kept at each node. The same procedure will be repeated until the production of the block $B^{(mt)}$.

At some moment a node, say node c, mines the block $B^{(mt)}$. If this block is approved by M verification nodes, then node c assembles a group of t blocks $B^{((m-1)t+1)}, \ldots, B^{(mt)}$ by requesting the previous $(t-1)$ blocks from the corresponding miners. Next, node c performs the encoding algorithm using LDPC codes, as described in Sections III-A and III-B. Finally, the mining node c sends triplets $(\mathbf{v}_j, j, m), j = 1, \ldots, n$, to the corresponding nodes j. If some nodes are offline, then the corresponding triplets will be lost. After distributing all the coded triplets, blocks $B^{(i)}$, $((m-1)t+1) \leq i \leq mt$, will be removed from their corresponding miners.

Additionally, node c will also be responsible for encoding a group of t_h block headers. For easy implementation, one may enforce $t_h = t$, i.e., the block encoding and the block header encoding procedures will be conducted at the same pace.

Finally, note that in the proposed scheme, only the miner keeps the block until it is involved in the block coding procedure. In case the miner leaves the blockchain network in the duration, we propose to use the following strategy. The scheme will require M verification nodes to keep copies of the block that they verified. If the miner leaves the network, then one of the M verifiers will take its role to conduct encoding. In particular, it is natural to assume that the representative of M verification nodes (discussed later in Section V-B3) can take this role.

We point out that the coding procedure differs from that for the blockchain with full nodes in the following aspects: 1) only one miner conducts the encoding scheme and 2) only the block header instead of the entire block is broadcasted over the network. It can be noted that for the all-light-node blockchain, the coding scheme is tailored to reduce the network communication cost and the number of nodes conducting the encoding scheme, while still maintaining the blockchain feasibility. It is also important to remind that erasure correcting codes, in particular, LDPC are vitally important in a blockchain with only light nodes since light nodes do not have enough memory for storing the entire ledger. An erasure code (in particular an LDPC code) allows one to drastically reduce the needed memory of light nodes.

B. Block Verification Algorithm With M Nodes

In the currently used blockchain systems, every full node has to verify a newly mined block, which is very computationally consuming and requires large network load, since the block should be broadcasted to all full nodes, whose number could be very large. In this section, we present a novel block verification algorithm that employs only M verification nodes, which greatly reduces the network load.

1) Verification Criteria: As we indicated at the beginning of Section V, we assume that nodes maintain the account-balance table, which consists of the public keys of nodes and their current balances. This table allows a node to check the validity of a transaction. A big advantage of such a table is that its size does not grow with time, but depends only on the number of nodes in the blockchain, which typically does not actively grow. As a result, the entire table can be kept in the memory of small devices such as cell phones.

It is not clear at this moment, whether this table would be sufficient for running smart contracts and other advanced functions. We leave this question for future research. Therefore, in this article, we assume only transaction-oriented blockchains, such as Bitcoin for example. For these types of blockchain, possession of the account-balance table should be enough for checking the validity of transactions.

This table allows checking a transaction according to the following two criteria.

1) The transaction is signed correctly by the sender.
2) The sender's account has sufficient balance for the transaction.

Furthermore, the block verification can be completed by any node with low complexity, by checking the following items.

1) Check if a duplicate of this block already exists.
2) Check if the PoW is correct.
3) Check if each transaction is valid.

In order to conduct item 1), a node compares the value of hash $h^{(i)}$ of header $H^{(i)}$ with hashes $h^{(t)}$ of $H^{(t)}$, $t < i$, and if necessary compares corresponding blocks in the same way as it was described at the end of Section IV.

2) Fake Transactions Insertion: We believe that any protocol in a decentralized blockchain must have an incentive. We discussed in Section II the incentive for full nodes to conduct verification of newly mined blocks. We also pointed out there is a problem with identifying an incentive for a full node to share information with light nodes, which potentially can be a problem. For this reason, we would like to create an incentive for each of M verification nodes to conduct honest verification instead of simply saying that a newly mined block is valid.

The block verification procedure is very simple and cheap, and for this reason, even a small verification reward should be good enough as an incentive. We propose to organize such kind of incentive below.

We propose that the miner randomly inserts a predefined number w of fake transactions into the block that it is going to mine. The purpose of these w fake transactions is to give other nodes a tool to detect whether a particular verification node indeed conducted verification. In order to do this, the miner may copy some transactions from those that are not included in a block and make any change on their addresses. Therefore, the fake transactions will not be validated, and further has no impact on the account-balance table. Note that w fake transactions will not be included for computing the Merkle root in the block header. Furthermore, the miner creates a binary sequence $\mathbf{s}_0$ in which 1's correspond to valid transactions and 0's to fake transactions. Finally, the miner computes and broadcasts the hash

$$h_0 = \text{hash}(\mathbf{s}_0). \tag{13}$$

Each verification node, say node j, is supposed to verify all transactions and create a binary sequence $\mathbf{s}_j$ whose entry indicates which transactions are valid and which are not. The verification node is also supposed to compute $h_j = \text{hash}(\mathbf{s}_j)$. Without looking into the details of transactions, node j is not capable to create $\mathbf{s}_j$ and h_j so that $\mathbf{s}_j = \mathbf{s}_0$ and further $h_j = h_0$. In addition, the verification node makes other checks, namely:

1) the solution to the required PoW is correct;
2) there is no previous block duplicating the current block;
3) the number of invalid transactions is the same as w and all the other transactions are all valid;
4) $h_0 = h_j$;
5) the account-balance update information is correct.

The reason for 3) and 4) is the following. It is known that mining takes a lot of computational resources, and hence a miner wants to produce a correct block (in order to get mining reward), since otherwise it will be rejected by other nodes. Therefore, it is reasonable for a miner to give a correct value of w. Furthermore, if the number of invalid transactions is larger than w (implies some extra transactions are wrong), then the block is invalid. Moreover, if the number of invalid transactions is less than w, then perhaps the miner is lying. Finally, if the number of invalid transactions match the value of w, then item 4) guarantees the correct order of w fake transactions in $\mathbf{s}_j$.

Finally, after all these checks, verification node j makes a decision $u_j = 1, 0$ whether the block is valid or not.

Note that by comparing h_0 and h_j, any other node can check whether node j indeed conducted verification or not, and according to this checking, it can decide whether to assign a reward to node j or not (see details on this below).

3) M Verification Nodes Selection: The PoW protocol is widely used in many blockchain systems for several reasons. In particular, it is used in order to prevent cyber attacks such as a denial-of-service attack. The PoW in most blockchain systems is based on searching for a nonce, say β, that leads to a hash value that satisfies certain inequality, (see Section II).

We propose that miner uses β as a seed of a random integer generator, which randomly outputs γM node indices, where γ is a small integer, like $\gamma = 5$. Let those indices form a set V, $|V| = \gamma M$. The number of verification nodes can be chosen according to the expected fraction of malicious nodes. The larger number of such nodes is expected, the larger should be M (see the end of this Section V-B on choosing M). For the moment, we assume that $M = 99$.

Next, the miner broadcasts β to all nodes. We specified in Section V-A that the miner broadcasts the header of the mined block. Hence, each node can check whether the broadcasted β is valid and that it satisfies the PoW. If β is correct, each node uses it to generate the same set $V(\beta)$ of verification nodes as the set V used by the miner. Note that finding another nonce, say β', for a given block, is a time and computationally consuming task. Therefore, a dishonest miner will be able to generate only a small number n_β of nonces for a given block. It is reasonable to assume that $n_\beta = 1, 2$, or maybe 3. The generation of additional nonces does not help much to a dishonest miner. Indeed, if the miner uses any of those nonces, say nonce β', then it will get another list $V(\beta')$ of γM random nodes. However, these verification nodes are still random and they are not chosen according to the preferences of the miner. Hence, it is not difficult to see that the probability p_{wrong} of wrong majority voting decision, defined in (18) in Section V-B, will be at maximum $n_\beta p_{\text{wrong}}$. Since p_{wrong} is typically very small, the factor n_β will not play a significant role.

Let us now assume that a dishonest miner broadcasts a nonce β, but uses a list $\tilde{V}$ comprised of verifiers that are controlled by itself. In this case, since $V(\beta) \neq \tilde{V}$, all the network nodes will reveal this and will not accept this block (see Section V-B6 for details).

Next, the miner sends a random number α to nodes from V and asks them to solve a small PoW, i.e., find a value Q satisfying

$$\text{hash}(\alpha,\ \text{the node's public key}, Q) < \text{threshold}. \tag{14}$$

We recall that the attacker in a Sybil attack forges a large number of pseudonymous nodes and utilizes them to make a disproportionately large influence. If some node uses such a Sybil attack, then it can be capable of inserting many of those pseudonymous nodes into set V. However, since we require that each node from V solves a small PoW, this approach becomes very expensive for an attacker since it will have to spend too many resources for solving multiple PoWs (14). Thus, this prevents the event that many nodes from V are controlled by one user.

Each node has the incentive to get the verification reward, so we can expect that many nodes from V will try to solve the assigned PoW and send their solutions to the miner. Let such nodes form set $V' \subset V$. Next, among nodes from V', the miner selects the first M verification nodes who sends back the correct solution for PoW (14), which will form set $V,''$ $|V''| = M$. The miner further assigns the node, say j_{rep} with the smallest index among nodes from V'' as the representative of V''.

We note again that by doing this, the miner makes sure that M verification nodes are online and the Sybil attack is unlikely. If the number of nodes who reply to the miner is less than M, then the value of multiplier γ needs to be increased.

4) Block Distribution: We denote by T_{m-1} the account-balance table that was created during generation of group G_{m-1}.

The miner sends the full block, e.g., $B^{(i)}$, only to M selected verification nodes from V''. Additionally, the miner also broadcasts the indices of nodes from V'', h_0, the block header $H^{(i)}$, and the difference Δ_m between its current table T_m and T_{m-1}. The purpose of this is that all nodes can create the same table T_m, using Δ_m and T_{m-1} that is the same at all nodes. This ensures that when the verification nodes check transactions from $B^{(i)}$, they use the same balance table T_m.

Each node outside of V'' will wait for the vote vector (described below) from the representative node j_{rep} for further process.

5) Verification Protocol Using the Byzantine Fault Tolerance Algorithm: The Byzantine type consensus algorithms find numerous applications within the blockchain technology, (see [34]–[36] and references therein). In particular, Das *et al.* [34] proposed new and nontrivial modifications of the Byzantine consensus algorithms for blockchains that run computationally intensive contracts. In this article, we use a relatively simple algorithm described in [20].

Recall that the BFT algorithm [20] considers the scenario in which M participants possess individual messages $a_j, j = 1, \ldots, M$ (for instance $a_j = 0, 1$) and send each other messages b_j. It is assumed that a receiver always knows from whom it receives a particular message. The messages b_j can be of the form "participant j has data c_j," or "participant j received from participant i message e_i." It is shown in [20] and in a number of other papers that, if there are more than $2M/3$ active and honest participants (who always send only true messages), then after $\lfloor (M-1)/3 \rfloor + 1$ rounds,[1] all the honest participants will possess the same vector $(f_1, \ldots, f_M)$ and $f_j = a_j$.

In our scheme, each node from V'' form message a_j in the following way. If node j is honest and $B^{(i)}$ is valid, then

$$a_j = \big(\mathbf{s}_j, u_j, j, \text{ signature of node } j\big). \tag{15}$$

If node j is honest, but $B^{(i)}$ is invalid, then

$$a_j = \big(\text{reason why block is invalid}, u_j, j, \quad \text{signature of node } j\big). \tag{16}$$

Note that node j picks up a reason why block is invalid from a predefined sets of such reasons. For instance, such set can contain reasons: the number of invalid transactions is larger than w, or this block is a copy of a previous block.

Next, the nodes run the BFT algorithm, and after $\lfloor (M-1)/3 \rfloor + 1$ rounds, each honest verification node possesses the same vote vector, i.e.,

$$\mathbf{f} = (f_1, f_2, \ldots, f_M) \tag{17}$$

with $f_j = a_j$. Finally, the representative node j_{rep} will simply broadcast the vote vector $\mathbf{f}$. Note that each vote f_j is signed by node j and thus the representative node cannot modify it.

6) Validity Decision of Block: Now each node in the blockchain receives the following items: the block header $H^{(i)}$ and account-balance update information Δ_m from the miner, the hash value h_0 from the miner, and the vote vector $\mathbf{f}$ from the representative verification node. Subsequently, each node will check the following items.

1) The index j of the verifier belongs to set V.
2) The signature of each vote in vector $\mathbf{f}$ is correct.
3) The number of zeros in $\mathbf{s}_j$ is w.
4) $h_j = h_0$.
5) The decision u_j from node j is 1 (valid) or 0 (invalid).

Note that if the item 1) is violated, then each node could broadcast a warning message so that most honest nodes will not accept the block. Otherwise, each node makes a decision on the validity of the block $B^{(i)}$ by applying the majority vote rule into the vote vector $\mathbf{f}$.

If the block is valid, then each node accepts the block header $H^{(i)}$ and computes the new account-balance table T_m using T_{m-1} and Δ_m. Furthermore, M verification nodes are divided into two sets by each node. The first set V_1 consists of those verification nodes whose votes are the same as the final decision, while the second set V_0 contains the remaining verification nodes. Finally, only verification nodes from V_1 will be awarded.

If the block is invalid, then the block header and the account-balance update information Δ_m will be rejected. In this case, the nodes from V_1 will still be rewarded, as they discovered the invalidity of the block.

It is important that the verification nodes from V_0 will get no reward (disregarding whether the block is accepted or not), since they did not conduct proper block verification. This creates an incentive to conduct verification properly.

The reward for the nodes from V_1 can be implemented with the following two approaches. In the first approach, the next mining node, say node d, should create reward transactions for these verification nodes and broadcast them. Each reward transaction transfers some coins, for example, "current mining reward$/(10M)$," from d to the nodes in V_1, where "current mining reward" is the mining reward for the new miner. In the second approach, node d will create reward transactions with "current mining reward$/(10M)$" coins from the air, such as coinbase transactions for mining reward in Bitcoin-type blockchains, and further broadcasts them. The only difference between the above two schemes is that whether verification nodes are paid by the new miner or by the system.

Next, if verification nodes that check the block mined by d see that not all the nodes from V_1 have been awarded, then they will not validate the new block and node d will not get its mining reward.

Finally, it is important to choose M so that the probability of making a wrong decision on a newly mined block is small, even if there are b dishonest nodes in the blockchain. Assume $M+1$ can be divided by 3. Then, the Byzantine-based decision will be wrong if $M' > (M+1)/3$ verification nodes are dishonest and all of them make the same decision. The last assumption is not very likely since this means that all M' randomly chosen dishonest nodes should have a common control, which is hardly true in real life. Still we make this worst scenario assumption. Under those assumptions, the probability of wrong majority voting decision is

$$p_{\text{wrong}} = \sum_{M'=(M+1)/3}^{M} \binom{n-b}{M-M'} \binom{b}{M'} \Big/ \binom{n}{M}. \tag{18}$$

[1] It is shown in [20, Section 3] that in a general case, when m out of n nodes are dishonest and $n \geq 3m+1$, $m+1$ rounds are enough for achieving a consensus.

Algorithm 4 Block Verification Algorithm With M Verifiers

for Each node that mines a new block **do**
  Insert randomly w fake transactions, compute h_0 using (13);
  Generate γM randomly selected indexes, using nonce β as the seed of a random numbers generator;
  Send a random number α to the γM selected nodes and ask them to solve the PoW (14);
  Appoint M verification nodes from those nodes who firstly send back the solutions of (14), and designate one of them as a representative;
  Broadcast the nonce β;
  Broadcast block header $H^{(i)}$, the account-balance update information and h_0 across the network;
  Send the full block $B^{(i)}$ to the M verification nodes;
end for
for The M verification nodes **do**
  Each verification node checks the block independently and generates message a_j according to (15) or (16);
  Run the Byzantine fault tolerance algorithm;
end for
The vector $\mathbf{f}$ defined in (17) is broadcasted by the representative of M verification nodes;
for Each node in the blockchain **do**
  if The index j of the verifier does not belong to set V **then**
    Reject the block header and the account-balance update information;
    Broadcast a warning message that the verifier j is illegal;
  else
    if The majority in the vote vector $\mathbf{f}$ is positive **then**
      Accept the block header and update the account-balance table;
      Create reward transactions for the verification nodes who vote positive;
    else
      Reject the block header and the account-balance update information;
      Create reward transactions for the verification nodes who vote negative;
    end if
  end if
end for

This expression can be used for choosing an appropriate M. For example, if $n = 1000$ and $b = n/100$, then it is enough to choose $M \geq 32$ to get $p_{\text{wrong}} = 0$. Similarly, for $b = n/50$, it is enough to choose $M \geq 62$ to get $p_{\text{wrong}} = 0$, while for $b = n/20$, it is enough to choose $M \geq 152$ to get $p_{\text{wrong}} = 0$. This probability can be considered as virtually zero probability (the chance that some other terrible event would happen in this world is significantly higher).

Finally, the proposed block verification algorithm is summarized in Algorithm 4.

VI. Analysis of the Proposed Blockchain Coding Scheme

A. Data Availability

The proposed distributed block encoding scheme assumes that we replace storing t blocks by storing only one vector $\mathbf{v}_j$ defined in (3) and the hashes of the block headers, and the size of $\mathbf{v}_j$ is equal to the size of one block. However, at any time, any block or block header can be recovered by any node with the proposed low complexity block recovery algorithm, as described in Section III-C. Additionally, for a blockchain with a larger number of nodes, any node can be offline or experiences data loss while still maintaining a reliable block availability.

B. Malicious Nodes Detection

We would like to point out that the proposed scheme can also be potentially used for the detection of malicious nodes. For instance, consider the scenario where node c wants to recover group G_m and hence requests certain vectors $\mathbf{v}_j$ from corresponding nodes. Assume further that several malicious nodes send back fake vectors $\mathbf{v}'_j$. Despite of this, if the number of such fake vectors is not too large, node c still will be capable to correctly reconstruct blocks from G_m and also detect the fake vectors $\mathbf{v}'_j$. The reason for this is that the LDPC code can correct not only erasures, which correspond to unaccessible nodes, but it can also identify errors, which correspond to fake $\mathbf{v}'_j$ and correct them. After this, node c can broadcast the information about malicious nodes over the network and possible penalties can be applied to the malicious nodes.

Such kind of algorithms need further studies and will be considered in future work.

C. Replay Attack Prevention

In a replay attack, an attacker could copy a whole transaction T_1 that has already been accepted and then resend it at a later time (we refer to this copy as T_2). This could happen if network congestion occurs such that T_2 arrives before T_1 at one node. In this case, the timestamp will help. Specifically, the timestamp of T_2 is definitely later than T_1, then finally when T_1 arrives, the node will replace T_2 by T_1, as the two transactions are exactly the same except the timestamp.

Another situation occurs when the blockchain has forks. For example, an attacker could copy the digital signature from an accepted transaction on one chain and then resend it to the other new chain, and the new chain may also accept it. To address this concern, we could append a new field "fork" in the account-balance table for the all-light-node blockchain. The "fork" field will indicate the "public-key balance" pair belongs to which blockchain branch.

D. Robustness to Adversarial Attacks

Let us assume that among the network nodes, there are τ malicious nodes that send to other nodes wrong code symbols. So, if the code symbols stored by a malicious node is say $\alpha \in \mathbb{F}_q$, then it sends to other nodes $\beta \in \mathbb{F}_q$ instead. This is equivalent to saying that malicious nodes introduce

errors, and now our LDPC code should correct both erasures and errors. LDPC codes are capable of doing this. A small problem, however, is that the most powerful soft-decision decoding algorithm of LDPC codes needs to know the channel probability of error. This probability is τ/n, however, since the number τ is not known to us we cannot find this probability. However, an LDPC code still can correct errors using one of the following two approaches.

First, we can use the hard-decision decoding of LDPC codes. There are a large number of papers on hard-decision decoding of LDPC codes, starting from the original Gallager paper [19], in which he considered bit-flipping decoding algorithms A and B, and up to modern papers (see [37]–[39] and references therein). Such algorithms do not require symbol error probability at all, and they still result in an excellent performance, as it was observed by R. Gallager 56 years ago.

Second, we can use a soft decision decoding as follows. We assume that the number of malicious nodes is never greater than $\tau_{\max}$ and construct LDPC codes capable of correcting errors and erasures in the channel with error probability $\tau_{\max}/n$ and some given probability of erasures. Let the probability of decoding error of this code be P_{de}, and this P_{de} be small enough for our purposes. If now the actual number of malicious nodes $\tau < \tau_{\max}$, we still can use the same soft-decision decoding assuming the same channel error probability. Let in this case, the decoding error probability be P'_{de}. Despite that we use in the channel error probability $\tau_{\max}/n$ instead of τ/n, we will have most likely $P'_{\text{de}} < P_{\text{de}}$, since the total number of errors τ is smaller than $\tau_{\max}$. This approach will be worked out in full detail in future work.

E. Storage Comparison

This section compares the required storage cost for different blockchain systems. Note that only the storage space related to blocks and block headers are counted.

Assume that at a given time instant, the total number of blocks in the blockchain is mt and each block has the size of k symbols over $\mathbb{F}_q$. Moreover, the total number of nodes in the network is n.

In a standard (uncoded) blockchain system with n full nodes (since only full nodes store blocks), the total storage usage amounts to $nmtk$ symbols.

Perard *et al.* [18] proposed the blockchain that consists of low-storage nodes and conventional full nodes and light nodes. The low-storage nodes conduct the encoding scheme using erasure codes. In the scheme of [18], each low-storage node would require only z ($z < k$) symbols for each block. Let us assume further there are n' low-storage nodes and n conventional full nodes in the blockchain proposed in [18]. As a result, the storage space required with the scheme of [18] is $n'mtz + nmtk$ symbols. When $z = 1$, the minimum storage needed is thus $n'mt + nmtk$ symbols.

In the proposed blockchain system using the distributed block coding scheme, assume an $[N, t]$ LDPC code is employed and each group is comprised of t blocks. Consequently, the overall (for all nodes together) required storage resource for all the blocks is only mnk symbols. If the same LDPC code is applied in encoding a group of t block headers and each block header is partitioned into k_h symbols, then extra mnk_h symbols are needed. Furthermore, assume a hash function with a fixed output of d symbols ($d \ll k$) is used, as each node also stores the hashes of all block headers, then the additional storage for these hashes is $mntd$ symbols. In total, the required storage space is $mn(k+k_h+td)$ symbols.

Compared with the uncoded blockchain, the storage saving is up to $[tk/(k + k_h + td)]$ times, or alternatively $[(1 - \epsilon)kN]/[\alpha k + \alpha k_h + (1 - \epsilon)Nd]$, by substituting the relations $R = t/N \leq (1 - \epsilon)/\alpha$, which is defined at the end of Section III-A. Moreover, it is worth pointing out that the storage saving would be significant, especially when t is large, i.e., when a large number of blocks is included in a group.

On the other hand, if the relation $N > n + n'$ holds for the selected LDPC codes, then the overall required storage amounts to $m(n + n')(k + k_h + td)$ with extra n' nodes in the proposed blockchain. As a consequence, if $n' \leq ([n(tk - k - k_h - td)]/[k + k_h + td - t])$ holds [or alternatively $n' \leq ([n((1-\epsilon)N(k-d)-\alpha k-\alpha k_h)]/[\alpha k+\alpha k_h+(1-\epsilon)N(d-1)])]$, then the overall required storage with the proposed blockchain system is smaller than the counterpart scheme in [18].

F. Communication Cost Reduction

We point out that in addition to the substantial storage cost reduction, the proposed blockchain with all light nodes also reduces the communication cost. Since in this scheme, only the block headers are broadcasted in the network, as opposed to the entire blocks. The only disadvantage of this scheme is that the nodes can reconstruct full blocks with some delay. At the same time, if a node needs a specific block before the block encoding is conducted, then this node can request the desired block immediately from its miner.

G. Compatibility With Increasing Number of Nodes

As the parameter pair $[N, t]$ is fixed in the proposed block coding scheme, the event that $n > N$ could happen, i.e., the number of nodes in the blockchain is larger than the length of the coded symbols. To deal with the above situation, we consider the following two cases: 1) n is slightly larger than N and 2) n is significantly larger than N.

To conquer the first problem, each triplet $(\mathbf{v}_j, j, m)$ can be sent to node j and node $N + j$. In this way, if a node needs to recover a block, the probability of receiving the same vectors $\mathbf{v}_j$ from different nodes is small, and thus it has a small side effect on the block recovery performance. For the second problem, each node in the blockchain may employ multiple LDPC codes with different codeword lengths, i.e., various values of N.

H. Experimental Results

This section assesses the performance of the proposed distributed block coding scheme, i.e., the storage usage at each node of the proposed blockchain system and the performance of the block recovery algorithm are evaluated, in a blockchain simulation platform using the account-balance table.

TABLE III
DATA STRUCTURE OF A BLOCK IN THE SIMULATION PLATFORM

Block index	Time when the block was mined	Node that mines the block
Previous block index	Node that mines the previous block	Total number of blocks in the blockchain
Transaction send node	Transaction receive node	Transaction amount
...	...	...
...	...	...

In this section, a $(d_v, d_c) = (3, 4)$ regular LDPC code with $[N, t]_q = [4000, 1000]_{2^{16}}$ is utilized as an example. Additionally, the protograph-based LDPC codes with various codeword lengths (these codes are taken from [40]) are also used to evaluate the performance of recovering a group of blocks. The codes are constructed in the following way. Specifically, a binary parity-check matrix $\mathbf{H}_1 \in \mathbb{F}_2^{(N-t)\times N}$ is first obtained using the progressive edge-growth (PEG) algorithm [41]. The purpose of utilizing the PEG algorithm is to reduce the number of short cycles in the constructed LDPC codes and hence guarantees the good decoding performance compared with the randomly constructed codes. Moreover, the PEG algorithm is popularly employed in LDPC codes construction, and the readers may refer to [32] and [41] and references therein for more instructions.

At the second step, as the positions of the nonzero elements (i.e., "1"s) in $\mathbf{H}_1$ has been optimized by the PEG algorithm, then the parity-check matrix $\mathbb{F}_{2^{16}}^{(N-t)\times N}$ is obtained by replacing these "1"s in $\mathbf{H}_1$, with the randomly selected nonzero elements in $\mathbb{F}_{2^{16}}$. Note that other methods have been investigated for constructing nonbinary LDPC codes, such as [42]–[44] and references therein, but the design of such codes is beyond the scope of this article.

The proposed distributed block coding scheme is implemented on a platform simulating the proposed blockchain with all light nodes. In this platform, each block is represented by a r-by-3 matrix, as demonstrated in Table III, where the "time" field is represented by a counter instead of the real time. In addition, the transactions at the beginning of a block are always reward transactions, i.e., a coinbase transaction for the current miner, followed by the reward transactions for the verification nodes.

Each entry in a block is originally stored in a decimal format, which is further converted to one symbol over $\mathbb{F}_{2^{16}}$ (i.e., four hexadecimal symbols). Moreover, we assume there are $n = 4000$ nodes in the blockchain and each group contains $t = 1000$ blocks. We also assume that $t_h = 1000$ block headers are encoded as a group. Finally, the number of verification nodes is $M = 100$. In the simulation, the block header consists of the first two rows in Table III.

Additionally, for the encoding procedure, each block is partitioned into $k = 900$ symbols, while each block header consists of six symbols over $\mathbb{F}_{2^{16}}$. Note that a hash function with the output of $d = 4$ symbols over $\mathbb{F}_{2^{16}}$ (i.e., 64 b) is utilized to compute the hash of each block header.

Next, the simulation platform is run to generate 6000 blocks in total (i.e., $mt = 6000$) and on average each block contains 870 symbols. Therefore, the required storage is 5.22×10^6 symbols at each node, assuming each node stores all the full blocks in the conventional uncoded blockchain.

However, for the proposed blockchain with all light nodes, by applying the distributed block coding scheme, the required storage at each node is only 29 436 symbols, i.e., 5400 coded symbols are related to all the blocks, 36 coded symbols are associated with all the block headers, and 24 000 symbols correspond to the hashes of all block headers. Therefore, a storage reduction by a factor of 177.3 is achieved.

Note that for a selected $[N, t]_q$ LDPC code, the maximum number of blocks included in a group is fixed at t. Furthermore, if larger values of $[N, t]$ are used, then the average storage savings at each node is also larger. Table IV illustrates the average storage savings at each node, for different numbers of blocks in a group, assuming the LDPC codes with parameters $[N, 2000]_{2^{16}}$, $[N, 3000]_{2^{16}}$, and $[N, 6000]_{2^{16}}$ are utilized, respectively. It can be observed that more storage reduction can be achieved with increasing t. Note that the storage reduction increases slowly with increasing t, which is due to the fact that the storage space for hashes of all block headers is mandatory (in this case, 24 000 symbols are inevitable).

Additionally, we point out that the storage reduction can also be more significant with an increasing number of blocks. Furthermore, considering a large number of nodes exist in a practical blockchain, the reduced storage space with the proposed blockchain is huge.

Next, let us evaluate the performance of reconstructing a whole group of blocks with the protograph-based LDPC codes optimized in [40]. We will show that the decoding performance

TABLE IV
COMPARISON OF STORAGE REDUCTION AT EACH NODE FOR DIFFERENT VALUES OF t

t	Number of symbols at each node	Average storage reduction
1000	29436	177.3
2000	26718	195.4
3000	25812	202.2
6000	24906	209.6

TABLE V
PERFORMANCE OF RECOVERING A GROUP OF BLOCKS USING THE PROTOGRAPH-BASED LDPC CODES FROM [40]

Code parameters $[N_a, t]$	Probability of node loss ϵ	Probability of successful recovery
$\mathcal{C}_1 : [4128, 1032]$	0.68	1.0
	0.69	0.99976
	0.70	0.99527
	0.71	0.94824
	0.72	0.73505
$\mathcal{C}_2 : [12000, 3000]$	0.70	1.0
	0.71	0.99995
	0.72	0.97091
	0.725	0.84513
	0.73	0.53892
$\mathcal{C}_3 : [65536, 16384]$	0.729	1.0
	0.731	0.995
	0.733	0.952
	0.735	0.825
	0.736	0.676

TABLE VI
COMPARISON FOR THE PROBABILITY THAT WE WILL NEED SYMBOLS AT LEVEL L_i FOR RECOVERING A DESIRED SYMBOL, WITH SIMULATION AND PREDICTION RESULTS

ϵ	$P_{sim}^{(0)}$	$P_{ana}^{(0)}$	$P_{sim}^{(1)}$	$P_{ana}^{(1)}$	$P_{sim}^{(2)}$	$P_{ana}^{(2)}$	$P_{sim}^{(3)}$	$P_{ana}^{(3)}$	$P_{sim}^{(4)}$	$P_{ana}^{(4)}$
0.01	0.0113	0.01	0	2.62×10^{-7}	0	1.85×10^{-16}	0	9.28×10^{-35}	0	0
0.05	0.0494	0.05	1.0×10^{-4}	1.45×10^{-4}	0	1.42×10^{-9}	0	1.35×10^{-19}	0	1.24×10^{-39}
0.1	0.1	0.1	1.5×10^{-3}	1.99×10^{-3}	0	1.05×10^{-6}	0	2.95×10^{-13}	0	2.36×10^{-26}
0.15	0.1551	0.15	0.0092	8.62×10^{-3}	0	4.22×10^{-5}	0	1.08×10^{-9}	0	7.08×10^{-19}
0.2	0.1992	0.2	0.0232	2.32×10^{-2}	7.0×10^{-4}	5.05×10^{-4}	0	2.73×10^{-7}	0	8.02×10^{-14}

approaches the theoretical erasure channel capacity. Table V shows the probabilities of successful group recovery with different probabilities of node loss in blockchain systems, for various codeword lengths. Vakilinia *et al.* [40] proposed the design of rate compatible protograph-based raptor-like LDPC codes for binary erasure channels, with rates ranging from (1/6) to (8/9). We construct the codes $\mathcal{C}_1$ and $\mathcal{C}_2$ from [40, relations (1), (15), and (16)] using the PEG algorithm, while code $\mathcal{C}_3$ is derived from [40, relations (1), (17), and (18)]. Note that the codes in [40] consist of punctured symbols. In other words, the original codeword lengths of $\mathcal{C}_1, \mathcal{C}_2$, and $\mathcal{C}_3$ are 4257, 12375, and 67584, respectively, with the first 129, 375, and 2048 symbols punctured. The listed values of code parameter N_a in Table V are the number of symbols that are actually distributed over the blockchain network. Moreover, the probabilities of successful recovery with codes $\mathcal{C}_1, \mathcal{C}_2$, and $\mathcal{C}_3$ in Table V are averaged by running the recovery algorithm for 10^5, 20 000, and 1000 times, respectively.

Theoretically, for erasure channels (this is the case in the blockchain scenario), a code with rate 1/4 can tolerate a probability of node loss 0.75. As can be observed from Table V, the overhead is around $[(1-0.69)/(1-0.75)]-1 \times 100\% \approx 24\%$ for code $\mathcal{C}_1$, and the overhead decreases to around $[(1-0.73)/(1-0.75)]-1 \times 100\% \approx 8\%$ for code $\mathcal{C}_3$ with much longer codeword length. This also implies that the decoding performance improves with the increasing number of nodes in blockchain systems. Furthermore, taking code $\mathcal{C}_1$ as an example, we can enlarge the set of nodes L by randomly adding $\lceil (1-0.71) * 4128 \rceil = 1198$[2] nodes (see the results for code $\mathcal{C}_1$ in Table V) for step 1 of Algorithm 1, and further enlarge the set with extra $\lceil 0.01 * 4128 \rceil = 42$ nodes. In this case, a node has around 94.824% successful probability of recovering a group of blocks, after adding 1198 nodes into set L at its first try, as shown in Table V. If it fails, then the proposed Algorithm 1 will be conducted at most several times, in order to finally reconstruct a group.

Subsequently, let us assess the performance of recovering a single desired symbol. For this, we use a $(d_v, d_c) = (3, 4)$ regular LDPC code with parameters $[N, t]_q = [4000, 1000]_{2^{16}}$. Table VI compares the probability that we will need to contact the symbols at more than i layers, for a successful recovery, and these probabilities are denoted by $P^{(i)}$. The subscripts "*sim*" and "*ana*" are used to distinguish between the simulated results and the analytical results computed from (9). The simulated probability is obtained by averaging over 10 000 simulations. Moreover, recall that the probability of an inactive node is $P_{ana}^{(0)} = \epsilon$.

It can be noted from Table VI that the simulated probabilities match the analytical results quite well. Additionally, even when $\epsilon = 0.2$, a single desired symbol can still be recovered by contacting those symbols at level L_2. Note that for higher values of ϵ, one may need to request the symbols at deeper levels.

Finally, we point out that the performance of recovering a group of blocks and the desired symbol can be further improved, if the LDPC codes with longer codeword length N are utilized and if the LDPC codes are well optimized.

VII. CONCLUSION

This article presents a novel distributed block coding algorithm for reducing the storage cost in a blockchain system. The proposed coding scheme encodes a group of blocks (respectively block headers) using the LDPC codes, and then only one coded vector is kept at each node in the blockchain. Moreover, the corresponding block recovery algorithms are explicitly provided. Furthermore, this article also proposes a novel blockchain with all light nodes, using the account-based model. Additionally, a block verification algorithm is presented, which involves only a small number of verification nodes. The proposed distributed block coding algorithm indeed drastically reduces the storage cost while maintaining the data integrity and availability of the blockchain.

[2] $\lceil x \rceil$ is the ceiling operation.

REFERENCES

[1] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. [Online]. Available: http://bitcoin.org/bitcoin.pdf

[2] G. Wood. (2014). *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. [Online]. Available: https://gavwood.com/paper.pdf

[3] D. Schwartz, N. Youngs, and A. Britto. (2014). *The Ripple Protocol Consensus Algorithm*. [Online]. Available: https://ripple.com/files/ripple_consensus_whitepaper.pdf

[4] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Proc. Annu. Int. Cryptol. Conf. (CRYPTO)*, Aug. 2017, pp. 357–388.

[5] E. Ben-Sasson *et al.* (2014). *ZeroCash: Decentralized Anonymous Payments From Bitcoin*. [Online]. Available: http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf

[6] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "MedRec: Using blockchain for medical data access and permission management," in *Proc. 2nd Int. Conf. Open Big Data (OBD)*, Vienna, Austria, Aug. 2016, pp. 25–30.

[7] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "HAWK: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. 37th IEEE Symp. Security Privacy*, May 2016, pp. 839–858.

[8] S. Underwood, "Blockchain beyond bitcoin," *Commun. ACM*, vol. 59, no. 11, pp. 15–17, Oct. 2016.

[9] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.
[10] J. Gao *et al.*, "GridMonitoring: Secured sovereign blockchain based monitoring on smart grid," *IEEE Access*, vol. 6, pp. 9917–9925, 2018.
[11] F. Lombardi, L. Aniello, S. D. Angelis, A. Margheri, and V. Sassone, "A blockchain-based infrastructure for reliable and cost-effective IoT-aided smart grids," in *Proc. IET Conf. Living Internet Things Cybersecurity IoT*, London, U.K., Mar. 2018, pp. 1–6.
[12] R. Li, T. Song, B. Mei, H. Li, X. Cheng, and L. Sun, "Blockchain for large-scale Internet of Things data storage and protection," *IEEE Trans. Service Comput.*, vol. 12, no. 5, pp. 762–771, Sep. 2019.
[13] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.
[14] *Blockchain*. Accessed: Feb 8, 2019. [Online]. Available: https://www.blockchain.com/en/charts/blocks-size#
[15] A. Churyumov. (2016). *ByteBall: A Decentralized System for Storage and Transfer of Value*. [Online]. Available: https://byteball.org/Byteball.pdf
[16] M. Dai, S. Zhang, H. Wang, and S. Jin, "A low storage room requirement framework for distributed ledger in blockchain," *IEEE Access*, vol. 6, pp. 22970–22975, 2018.
[17] R. K. Raman and L. R. Varshney, "Dynamic distributed storage for blockchains," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2018, pp. 2619–2623.
[18] D. Perard, J. Lacan, Y. Bachy, and J. Detchart, "Erasure code-based low storage blockchain node," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber Phys. Soc. Comput. (CPSCom) IEEE Smart Data (SmartData)*, Jul. 2018, pp. 1622–1627.
[19] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA, USA: MIT Press, 1963.
[20] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *J. ACM*, vol. 27, no. 2, pp. 228–234, Apr. 1980.
[21] *Transaction*. Accessed: Feb 8, 2019. [Online]. Available: https://en.bitcoin.it/wiki/Transaction
[22] *Protocol Rules*. Accessed: Feb 8, 2019. [Online]. Available: https://en.bitcoin.it/wiki/Protocol_rules#.22tx.22_messages
[23] C. Merkle, "Protocols for public key cryptosystems," in *Proc. Symp. Security Privacy*, Apr. 1980, pp. 122–133.
[24] L. Rizzo, "Effective erasure codes for reliable computer communication protocols," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 2, pp. 24–36, Apr. 1997.
[25] *Block Size Limit Controversy*. Accessed: Feb 8, 2019. https://en.bitcoin.it/wiki/Block_size_limit_controversy
[26] *Etherchain*. Accessed: Feb 8, 2019. [Online]. Available: https://www.etherchain.org/charts/blockSize
[27] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 569–584, Feb. 2001.
[28] S. Lin and D. J. Costello, *Error Control Coding*, 2nd ed. London, U.K.: Pearson Educ., 2004.
[29] D. Declercq and M. Fossorier, "Decoding algorithms for nonbinary LDPC codes over GF(Q)," *IEEE Trans. Commun.*, vol. 55, no. 4, pp. 633–643, Apr. 2007.
[30] W. Ryan and S. Lin, *Channel Codes: Classical and Modern*. Cambridge, U.K.: Cambridge Univ. Press, 2009.
[31] G. Liva *et al.*, "Design of LDPC codes: A survey and new results," *J. Commun. Softw. Syst.*, vol. 2, no. 3, pp. 191–211, Sep. 2006.
[32] Y. Fang, G. Bi, Y.-L. Guan, and F. C. M. Lau, "A survey on protograph LDPC codes and their applications," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 1989–2016, May 2015.
[33] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.
[34] S. Das, V. J. Ribeiro, and A. Anand, "YODA: Enabling computationally intensive contracts on blockchains with Byzantine and selfish nodes," in *Proc. Netw. Distrib. Syst. Security Symp. (NDSS)*, San Diego, CA, USA, Feb. 2020, pp. 1–15.
[35] T.-W. Chao, H. Chung, and P.-C. Kuo, "Fair Byzantine agreements for blockchains," Jul. 2019. [Online]. Available: arXiv:1907.03437.
[36] P. Tholoniat and V. Gramoli, "Formal verification of blockchain Byzantine fault tolerance," Oct. 2019. [Online]. Available: arXiv:1909.07453.
[37] H. Cui, J. Lin, and Z. Wang, "An improved gradient descent bit-flipping decoder for LDPC codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 8, pp. 3188–3200, Aug. 2019.
[38] Y. Liu and M. Zhang, "Hard-decision bit-flipping decoder based on adaptive bit-local threshold for LDPC codes," *IEEE Commun. Lett.*, vol. 23, no. 5, pp. 789–792, May 2019.
[39] T. C.-Y. Chang, P. Wang, and Y. T. Su, "Multi-stage bit-flipping decoding algorithms for LDPC codes," *IEEE Commun. Lett.*, vol. 23, no. 9, pp. 1524–1528, Sep. 2019.
[40] K. Vakilinia, D. Divsalar, and R. D. Wesel, "Protograph-based raptor-like LDPC codes for the binary erasure channel," in *Proc. Inf. Theory Appl. Workshop (ITA)*, San Diego, CA, USA, Feb. 2015, pp. 240–246.
[41] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 386–398, Jan. 2005.
[42] E. Boutillon, "Optimization of non binary parity check coefficients," *IEEE Trans. Inf. Theory*, vol. 65, no. 4, pp. 2092–2100, Apr. 2019.
[43] L. Dolecek, D. Divsalar, Y. Sun, and B. Amiri, "Non-binary protograph-based LDPC codes: Enumerators, analysis, and designs," *IEEE Trans. Inf. Theory*, vol. 60, no. 7, pp. 3913–3941, Jul. 2014.
[44] C. Poulliat, M. Fossorier, and D. Declercq, "Design of regular $(2, d_c)$-LDPC codes over GF(q) using their binary images," *IEEE Trans. Commun.*, vol. 56, no. 10, pp. 1626–1635, Oct. 2008.

Huihui Wu received the B.Sc. degree in communication engineering from the Southwest University for Nationalities, Chengdu, China, in 2011, the M.S. degree in communication engineering from Xiamen University, Xiamen, China, in 2014, and the Ph.D. degree in electrical and computer engineering from McMaster University, Hamilton, ON, Canada, in 2018.

From November 2018 to April 2019, he was a Postdoctoral Research Scientist with Columbia University, New York, NY, USA. He is currently a Postdoctoral Researcher with McGill University, Montreal, QC, Canada. His research interests include channel coding, joint source and channel coding, signal quantization, wireless communications, blockchain, and machine learning.

Alexei Ashikhmin (Fellow, IEEE) received the Ph.D. degree in electrical engineering from the Institute of Information Transmission Problems, Russian Academy of Science, Moscow, Russia, in 1994.

He is a Distinguished Member of Technical Staff with the Communications and Statistical Sciences Research Department, Nokia Bell Laboratories, Murray Hill, NJ, USA. He is also an Adjunct Professor with Columbia University, New York, NY, USA, where he teaches courses on quantum computing and error correction, digital communications, and error correcting codes. His research interests include communications theory, massive MIMO systems, theory of error correcting codes and its modern applications, as well as classical and quantum information theory.

Dr. Ashikhmin was a recipient of the 2017 SPS Donald G. Fink Overview Paper Award for the article "An Overview of Massive MIMO: Benefits and Challenges" published in the IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING, and the Bell Laboratories President Award for breakthrough research in wired and wireless communication projects, in 2002, 2010, and 2011. He received the Thomas Edison Patent Award in the Telecommunications for a Patent on Massive MIMO System with Decentralized Service Antennas in 2014 and the IEEE Communications Society Stephen O. Rice Prize for the best paper published in the IEEE TRANSACTIONS ON COMMUNICATIONS in 2004.

Xiaodong Wang (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Princeton University, Princeton, NJ, USA.

He is a Professor of electrical engineering with Columbia University, New York, NY, USA. His research interests fall in the general areas of computing, signal processing and communications, and has published extensively in these areas. Among his publications is a book entitled *Wireless Communication Systems: Advanced Techniques for Signal Reception* (Prentice-Hall in 2003). His current research interests include wireless communications, statistical signal processing, and genomic signal processing.

Dr. Wang received the 1999 NSF CAREER Award, the 2001 IEEE Communications Society and Information Theory Society Joint Paper Award, and the 2011 IEEE Communication Society Award for Outstanding Paper on New Communication Topics. He has served as an Associate Editor for the IEEE TRANSACTIONS ON COMMUNICATIONS, the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, the IEEE TRANSACTIONS ON SIGNAL PROCESSING, and the IEEE TRANSACTIONS ON INFORMATION THEORY. He is listed as an ISI Highly-cited Author.

Chong Li (Senior Member, IEEE) received the B.E. degree in electrical engineering from Harbin Institute of Technology, Harbin, China, and the Ph.D. degree in electrical engineering from Iowa State University, Ames, IA, USA.

He worked with Qualcomm Research, Bridgewater, NJ, USA, on the development of 4G LTE and 5G technologies. He is a Co-Founder of Nakamoto & Turing Labs, New York, NY, USA. He is also an Adjunct Professor with Columbia University, New York, NY, USA. He has holder of over 200 international/U.S. patents (granted and pending). He has been actively publishing papers on top-ranking journals. He has authored the book *Reinforcement Learning for Cyber-Physical Systems* (Taylor & Francis and CRC Press). He has broad research interests, including information theory, blockchain, machine learning, networked control communications theory, and systems design for advance telecommunication technologies (5G and beyond).

Dr. Li served as a reviewer and the technical program committee for most prestigious journals and conferences in communications and control societies.

Sichao Yang received the M.S. degree from the Department of Mathematics and the Ph.D. degree from the Department of Electrical and Computer Engineering from the University of Illinois at Urbana–Champaign, Champaign, IL, USA.

He is a Co-Founder and the CEO of Nakamoto & Turing Labs, New York, NY, USA, a research lab engaged in scientific and engineering research in the fields of blockchain and distributed computing technologies. His Ph.D. research is on game theory with its application on resource allocation and optimization in distributed networks. He was a Senior Staff Engineer with Qualcomm Inc., San Diego, CA, USA, the world's leading wireless communications chip and service provider. During his tenure in Qualcomm, he was one of the key contributors to the design and development of the fourth and fifth generation mobile communication. He also served as the Technical Leader in Qualcomm's research on vehicular network and autonomous driving and holds many invention patents.

Dr. Yang has many journal and conference papers and served as a Reviewer for the *Mathematics of Operations Research*, the IEEE TRANSACTIONS ON NETWORKING, *Games and Economics Behavior*, and other magazines. He also gave invited talks and tutorial sessions on IEEE conferences, including IEEE GLOBECOM and IEEE Summit on Communications Futures.

Lei Zhang received the B.E. and M.S. degrees in electronics engineering from Tsinghua University, Beijing, China, in 2005 and 2007, respectively, and the Ph.D. degree in electrical engineering and computer science from Northwestern University, Evanston, IL, USA, in 2012.

He was with Qualcomm Research, Bridgewater, NJ, USA. He is a Co-Founder of Nakamoto & Turing Labs, New York, NY, USA, and also an Adjunct Professor with Columbia University, New York. His research interests include blockchain systems, information theory, machine learning, and computer vision.

Dr. Zhang was a recipient of the Student Paper Award of the IEEE International Symposium on Information Theory in 2011. He also received the Walter P. Murphy Fellowship and Terminal Year Fellowship from Northwestern University in 2007 and 2011, respectively.