

BLOCK PROPAGATION WITH POISONED TRANSACTIONS IN A BLOCKCHAIN NETWORK

TECHNICAL FIELD

[0001] The present disclosure relates to blockchain networks and, in particular, to the propagation of blocks amongst miner nodes.

BACKGROUND

[0002] In proof-of-work blockchain systems, when a miner finds a valid block it tries to quickly communicate its success to all other miners. This involves propagating information regarding the block through the blockchain network to all the mining nodes. In some cases, this may involve sending the full block data. In some cases, this may involve sending block header and transaction list information. Receiving miners validate the new block by hashing the header and confirming it matches the hash value provided by the successful miner.

[0003] As blocks increase in size and transaction count, delay in block propagation can exacerbate the problems of temporary forks and orphan blocks. These situations are costly to miners and to the system overall. A solution to block propagation delays should be structured to guard against the possibility of undermining consensus rules.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] Reference will now be made, by way of example, to the accompanying drawings which show example embodiments of the present application and in which:

[0005] FIG. 1 shows an example block structure for a blockchain network;

- [0006] FIG. 2 shows a flowchart of an example method of block propagation;
- [0007] FIGs. 3A-3I illustratively show the sequence of messages and operations in an example implementation of block propagation in accordance with the present application;
- [0008] FIG. 4 shows, in flowchart form, one example method for block propagation in a blockchain network;
- [0009] FIG. 5 shows, in flowchart form, one example method of validating blocks in a blockchain network; and
- [0010] FIG. 6 shows, in block diagram form, a simplified example of a mining node.
- [0011] Like reference numerals are used in the drawings to denote like elements and features.

DETAILED DESCRIPTION OF EXAMPLES

[0012] In one aspect, there may be provided a computer-implemented method of block propagation in a blockchain network. The method may include, while hashing a first block header of a first candidate block at a first mining node, sending, to other mining nodes, a first message specifying transactions and their order in a first ordered set contained in the first candidate block; and receiving a second message identifying transactions and their order in a second ordered set contained in a second candidate block being mined by a second mining node.

[0013] In another aspect, there is described a method of block propagation in a blockchain network. The method may include constructing, at a first mining node, a first candidate block containing transactions in a first ordered set, the first candidate block having a first block header; placing one or more invalid transactions in the first ordered set to create a poisoned ordered set; constructing a list identifying the one or more invalid transactions within the poisoned ordered set, and encrypting the list to produce an encrypted list of invalid transactions; and while hashing the first block header, sending, to other mining nodes, a first message containing the poisoned ordered set and the encrypted list of invalid transactions.

[0014] In some implementations, the method may further include the first mining node finding a proof-of-work with respect to the first block header and, as a result, sending a decryption key for the encrypted list of invalid transactions.

[0015] In some implementations, placing one or more invalid transactions includes inserting the one or more invalid transactions into the first ordered set of transactions.

[0016] In some implementations, placing one or more invalid transactions includes, for each invalid transaction in the one or more invalid transactions, replacing a respective valid transaction in the first ordered set. In some cases, wherein the encrypted list of invalid transactions further includes the respective valid transactions. In some cases, the method may further include, at another mining node, constructing the first candidate block by decrypting the encrypted list of invalid transactions, replacing the one or more invalid transactions with the respective valid transactions in the poisoned ordered set in order to reconstruct the first ordered set and generate the first candidate block.

[0017] In some implementations, the one or more invalid transactions may include a transaction that violates at least one validity criteria prescribed by a blockchain protocol for the blockchain network.

[0018] In some implementations, the method may further include, at the first mining node, creating at least one of the one or more invalid transactions.

[0019] In some implementations, the method may further include, at the first mining node, selecting as one of the one or more invalid transactions an invalid transaction received from another mining node.

[0020] In some implementations, the method may further include the first mining node finding a proof-of-work with respect to the first block header and, as a result, sending the proof-of-work and a decryption key for the encrypted list of invalid transactions, wherein the proof-of-work includes a coinbase transaction from the first candidate block, and a nonce and timestamp from the first block header.

[0021] In some implementations, the method may further include, while hashing the first block header, receiving a second message identifying transactions and their order in a second poisoned ordered set derived from a second candidate block being mined by a second mining node, and receiving a second encrypted list, wherein the second encrypted list identifies invalid transactions within the second poisoned ordered set. In some cases, the method further includes validating the transactions in the second poisoned ordered set and identifying a set of one or more invalid transactions in the second poisoned ordered set. In some examples, the method then may further include receiving a coinbase transaction, a decryption key, and second block header information from the second mining node indicating successful completion of a proof-of-work with regard to the second candidate block, wherein the second block header information includes at least a timestamp and a nonce; decrypting the second encrypted list using the decryption key; determining that the second encrypted list identifies the same invalid transactions as in the set of one or more invalid transactions; assembling a second block header based on the coinbase transaction, the second block header information, and the second poisoned ordered set modified to replace the one or more invalid transactions in the second poisoned ordered set; and validating the second candidate block by hashing the second block header to validate the proof-of-work.

[0022] In another aspect, there may be provided a computing device implementing a first mining node for mining blocks on a blockchain. The computing device may include memory, one or more processors, and computer-executable instructions that, when executed, cause the processors to carry out one or more of the methods described herein.

[0023] In yet another aspect, there may be provided a computer-readable medium storing processor-executable instructions for implementing a first mining node for mining blocks on a blockchain network, the processor-executable instructions including instructions that, when executed by one or more processors, cause the processors to carry out at least one of the methods described herein.

[0024] Other example embodiments of the present disclosure will be apparent to those of ordinary skill in the art from a review of the following detailed description in conjunction with the drawings.

[0025] In the present application, the term “and/or” is intended to cover all possible combinations and sub-combinations of the listed elements, including any one of the listed elements alone, any sub-combination, or all of the elements, and without necessarily excluding additional elements.

[0026] In the present application, the phrase “at least one of... or...” is intended to cover any one or more of the listed elements, including any one of the listed elements alone, any sub-combination, or all of the elements, without necessarily excluding any additional elements, and without necessarily requiring all of the elements.

[0027] The present application will refer to hashing or a hash function, which is intended to include any one of a number of cryptographic hash functions that, when applied to an arbitrary set of data or “message”, deterministically produce a unique fixed-length alphanumeric string. The result of a hash function may be called a hash value, fingerprint, hash result, or equivalent. Examples include, but are not limited to, SHA-2, SHA-3, and BLAKE2. Any reference below to a miner hashing a block or a candidate block will be understood to mean applying a cryptographic hash function to the header portion of the candidate block.

[0028] In this document the term ‘blockchain’ is understood to include all forms of electronic, computer-based, distributed ledgers. These include consensus-based blockchain and transaction-chain technologies, permissioned and un-permissioned ledgers, shared ledgers and variations thereof. The most widely known application of blockchain technology is the Bitcoin ledger, although other blockchain implementations have been proposed and developed. While Bitcoin, as exemplified by the Bitcoin SV protocol, may be referred to herein for the purpose of convenience and illustration, it should be noted that the invention is not limited to use with the Bitcoin blockchain and alternative blockchain implementations and protocols fall within the scope of the present invention.

[0029] A blockchain is a peer-to-peer, electronic ledger which is implemented as a computer-based decentralised, distributed system made up of blocks which in turn are made up of transactions. Each transaction is a data structure that encodes the transfer of control of a digital

asset between participants in the blockchain system and includes at least one input and at least one output. Each block contains a hash of the previous block so that blocks become chained together to create a permanent, unalterable record of all transactions which have been written to the blockchain since its inception. Transactions contain small programs known as scripts embedded into their inputs and outputs, which specify how and by whom the outputs of the transactions can be accessed. On the Bitcoin platform, these scripts are written using a stack-based scripting language.

[0030] The blockchain is implemented over a network of nodes. Each node is a computing device with network connectivity and executing software that carries out the applicable blockchain protocol. Nodes validate transactions and propagate them to other nodes in the network. Specialized network nodes, termed “mining nodes” or “miners”, collect a set of unconfirmed transactions, *i.e.* pending transactions, into a block and attempt to “mine” the block. Mining, in these examples, refers to solving a proof-of-work (POW) before any other miner in the network succeeds in solving a proof-of-work for their respective block. In the Bitcoin example, a POW involves hashing a block header containing a nonce until the result is below a threshold value set by a difficulty parameter. The nonce is repeated incremented and the hashing repeated until the result is below the threshold value or until the miner receives notice that another miner has succeeded. Variations in mining process will be familiar to those ordinarily skilled in the art.

[0031] A typical block contains two data structures: a block header and transactions. Figure 1 diagrammatically illustrates an example block structure 100 from the Bitcoin protocol. The block structure 100 includes a block header 102 and a payload 104. The block header 102 in this example includes fields for a version number 106, a previous block hash value 108, a Merkle root 110, a timestamp 112, a target difficulty parameter 114, and a nonce 116. The previous block hash value 108 chains this block to the previous block in the chain, thereby resulting in the “blockchain” structure that links successive blocks together through cryptographic hashes. The Merkle root 110 refers to a Merkle tree structure based on all the transactions contained in the block. The nonce 116 is an arbitrary value that the mining node can repeated increment or decrement to change the contents of the block header 102 in order to produce different hash results when mining.

[0032] The payload 104 includes a transaction count value 118 and the list of transactions 120. The list of transactions 120 may be a list of transaction ID numbers in some implementations.

[0033] When a mining node succeeds in finding a block header that produces a hash result that is below the threshold value, then it proceeds to notify other nodes using an updated inventory message that includes the successful hash result value. The other nodes then request a copy of the new block and independently verify its validity.

Fast Block Propagation

[0034] The blockchain ecosystem is maturing to provide increased usability through a large increase in transaction volume and, as a consequence, block sizes. As blocks become larger, in some cases exceeding 128 MB, propagating a successfully mined new block to other nodes throughout the network takes an increased time. The delay in propagation comes at a cost. First, the miners that are unaware of that a successfully-mined block has been created will continue attempting to mine their own candidate blocks in a wasted effort if the new block proves to be valid. Second, the delay in propagation and validation may result in an increased likelihood of (temporary) forks and orphan blocks.

[0035] It will be appreciated that in many blockchain protocols, the miner does not send a full copy of the block, but instead sends the hash results in an inventory message. The receiving mining nodes determines that they have not seen this alleged new block and send a GETBLOCK or GETDATA message to the successful miner. Rather than sending a full copy of the block, the successful miner send the block header, the transaction count field from the payload, and an ordered list of the transactions included in the block. The ordered list may include a set of the full transaction ID numbers (TXIDs) for the transactions. The TXIDs may, in some embodiments, be a fixed length hash of the transaction. The TXID may be a 256-bit (32-byte) number obtained by hashing the transaction using SHA-256 for instance. A receiving node may reassemble the full block by retrieving the identified transactions from the mempool by their TXIDs.

[0036] Nevertheless, as modern block sizes grow to 128 MB and beyond the size of the data to be transmitted may still be significant if the count of transactions is large. For example, a block containing half a million transactions would have an ordered list of TXIDs that is 16 MB.

[0037] Accordingly, in one aspect, the present application describes methods and systems for blockchain networks in which mining nodes provide other miners with information about their respective candidate blocks while hashing their candidate blocks. In this manner, each miner exploits the time delay between the discovery of successful blocks, which in the example of Bitcoin is about ten minutes, to provide other miners with details regarding the structure and content of candidate blocks. By providing this information in advance, when a successful block is found the successful miner need only send information from the block header and, in some cases, a coinbase transaction in order to ensure all other nodes can assemble and validate the complete new block. This data may be as little as a few hundred bytes. This increases the speed with which a successful new block is propagated through the full network.

[0038] It will be appreciated that unconfirmed transactions are propagated and validated through the network of nodes before they are confirmed by being included in a valid block. While in this unconfirmed state, the unconfirmed transactions are held in memory. This memory may be termed a “mempool”. Each miner may, in some implementations, maintain its own copy of the mempool from which it can select a set of transactions for assembling a candidate block. In some alternative architectures, the mempool may be implemented as a distributed mempool across a number of nodes. In some architectures, specialized nodes may be employed by the blockchain network to manage the mempool and to provide mining nodes with transactions for inclusion in a candidate block. The present application contemplates use of the described methods and devices with any such variations in blockchain network architecture. For the purposes of simplicity of explanation, it will be assumed herein that each mining node maintains its own mempool of unconfirmed transactions.

[0039] The mining nodes use the mempool as a source of new transactions for building a candidate block. The mining nodes may further use the mempool when validating a new block from another miner as each block contains an ordered list of transactions. The ordered list may

identify transactions by a unique transaction identifier, like the TXID. Accordingly, the receiving mining node retrieves the transactions from the ordered list and validates the block against block-level criteria and validates the transactions against transaction-level criteria. In this way the miners guard against double-spends and other attacks.

[0040] To illustrate an aspect of the present application, reference is now made to Figure 2, which shows, in flowchart form, one example method 200 of blockchain mining. The method 200 is implemented by a mining node. The mining node may be implemented on a computing device, which may include one or more processing units. As will be appreciated by those skilled in the art the processing units may include specialized processing units with dedicated hardware designed to carry out the calculation operations associated with blockchain mining with significant speed and efficiency. However, it may also or alternatively include general purpose computing device. The computing device includes processor-executable software that includes processor-readable instructions that, when executed, cause the one or more processing units to carry out the operations described. The computing device includes memory and a network connection with the associated hardware and software for obtaining network connectivity and sending and receiving messages in accordance with the applicable network protocol.

[0041] The method 200 includes selecting a set of transactions from the mempool of unconfirmed transactions to build a candidate block in operation 202. The selection may be based on age of the transactions, fees payable for mining the transactions, or any other factors. The transactions form an ordered list of transactions. The mining node further determines the Merkle root based on the ordered list of transactions and forms the block header, including setting an initial value for the nonce in the header.

[0042] In operation 204, the mining node begins the repeated hashing of the block header and incrementing of the nonce as part of the search for a POW that meets the conditions of the difficulty setting. While engaged in searching for a POW with regard to its candidate block, the mining node sends information about its ordered list of transactions to other mining nodes, as indicated by operation 206. The mining node also receives messages from other mining nodes, while operation 204 is ongoing, that include information about the respective ordered lists of

transactions in the candidate blocks being worked upon by those mining nodes, as indicated by operation 208. It will be appreciated that although operations 204, 206 and 208 are shown in sequence, they generally happen in parallel. The received information about respective ordered lists of transactions from other miners is stored locally by the mining node. The mining node may store the information in a table or other data structure.

[0043] The search for a POW continues until the mining node successfully finds a POW, as indicated by operation 210, or receives notice from another mining node that it has found a POW, as indicated by operation 212. If the mining node finds a POW, then in operation 214 it sends some header information to other mining nodes to propagate the block details. Notably, it does not need to send the payload. It may, however, send a coinbase transaction.

[0044] If the mining node receives header info from another mining node, then in operation 216 it assembles the new block based on the stored ordered list of transactions for that mining node. If a coinbase transaction is received, then that transaction is added to the block and the mining node can construct the Merkle tree to confirm that the Merkle root is valid. The mining node then validates the new block in operation 218 and, if invalid, it may discard the new block and continue searching for its own POW. Note that there may be certain validity checks that, if failed, may prompt the mining node to take other action such as requesting additional data from the successful mining node, as will be described later. If the block is validated, it is added to the blockchain in operation 220 and propagated to further mining nodes. The mining node then returns to operation 202 to build a new candidate block and start the search anew.

[0045] It will be appreciated that the described method of pre-distributing candidate block information so that each miner knows the ordered list of transactions for each other miner, ensures only a minimum amount of information need be propagated after the POW is found. This ensures that mining nodes have the information they require to validate a new block as soon as possible, thereby reducing time wasted in search for a POW when another miner has already succeeded. Mining nodes have every incentive to propagate and validate new blocks as quickly as possible so that they can move onto searching for the next block. Potential delays due to network latency are

of much less concern when pre-distributing ordered lists of transactions during the approximately ten-minute period between successive blocks.

[0046] In one example implementation, the ordered list of transactions from a candidate block may be communicated to another miner by adding ordering information to a transaction data packet. Each miner is able to identify each other miner uniquely. In one example implementation, a message may be sent that uniquely identifies the miner and the block level that the miner is working on, and that lists the TXIDs for that miner's candidate block in order.

[0047] In another example implementation, the messaging protocol may provide for appending transactions to an ordered list for a miner, removing or deleting transactions from an ordered list, replacing a transaction with another transaction, or re-ordering transactions. In the normal course, a miner is likely to use only an "add" message that specifies the ordered list of transactions. However, there are some cases in which a miner may wish to remove or replace a transaction. For example, subsequently-received information may indicate a potential double-spend with regard to one of the transactions or some other potential issue with its validity. As another example, the miner may cycle through all increments of the nonce and may wish to reorder or otherwise adjust the transactions so as to change the Merkle root and thereby change the header to continue mining the block. The precise format the message structures may vary by implementation.

[0048] Further compression of the block propagation messages can be achieved by shortening the TXID string. The TXID in some examples is a 32 byte string. However, in some cases a compressed TXID may be used that relies on sending only a portion of the TXID. In one example, only the first eight bytes are sent.

[0049] Although the first 8 bytes of a TXID is not guaranteed to uniquely identify a transaction, the likelihood of collision in two TXIDs, both within a single message or in general is small. Given that SHA256 outputs are pseudorandom 256-bit strings, the exact likelihood of collision of two random TXID, *i.e.* the likelihood that any two TXIDs have the same first 4 bytes is:

12

$$P(TXID_1[0:8] = TXID_2[0:8]) = \frac{1}{2^{8 \times 8}} = 0.0000000000000000000542$$

[0050] Where $TXID_i[0:n]$ means ‘the first n bytes of $TXID_i$ ’. Furthermore, the likelihood that a message containing N compressed $TXIDs$ will contain one or more collisions may be expressed as:

$$\begin{aligned} P(> 0 \text{ collisions in } N \text{ } TXIDs) \\ &= 1 - P(TXID_i[0:8] \neq TXID_j[0:8] \text{ for } i, j \in \{1, \dots, N\} \text{ } i \neq j) \\ &= 1 - \left(1 - \frac{1}{2^{64}}\right)^{\frac{N(N-1)}{2!}} = 1 - 0.9999999999999999999458^{\frac{N(N-1)}{2!}} \end{aligned}$$

[0051] As an example, if a block were to contain a million transactions such that a message contained a million compressed $TXIDs$ ($N = 1,000,000$), then the likelihood of a collision is:

$$P(> 0 \text{ collisions in } 1,000,000 \text{ } TXIDs) = 0.0000000271$$

[0052] This probability is very small and, even if a collision occurs, the message receiver may simply request the same message in uncompressed form. Alternatively, the sending mining node may first conduct a check to confirm that the ordered set of compressed $TXIDs$ does not contain any collisions before sending and, if any are detected, send the message with uncompressed $TXIDs$. A flag or other signal in the message may indicate whether the $TXIDs$ in the payload are in compressed or uncompressed format.

[0053] In accordance with the process described herein, each miner involved in the blockchain network may receive messages from each other miner specifying an ordered list of transactions contained in that other miner’s candidate block. To track this information, each miner may store the ordered lists in association with the respective miner identities. The miner identities may be determined by IP address or some other unique identifier. In one example implementation, each miner maintains a table or similar data structure in which each row is associated with one of the transactions in the mempool and each column is associated with a miner in the network. Order information may then be stored in the cells of the table indicating for each miner the order in which

the miner has arranged the transactions in its candidate block. It will be understood that not all transactions will have an order value as not all transactions are included in every candidate block. The table below shows one simplified example of a miner ordering data table:

TXID	Miner ID	A	B	C	D
TX_1		1	2	1	1
TX_2		2	1	3	4
TX_3		3	3	2	2
TX_4		4	4	4	5
TX_5		5	5	5	3

[0054] In this simplified example table, there are four miners having miner IDs A, B, C, and D. Each miner has received and validated transactions TX_1 , TX_2 , TX_3 , TX_4 and TX_5 . The table can be updated to add transactions to a miner's ordered list, to replace a transaction in the order with another, or to delete a transaction from the ordered list, which may result in adjusting the ordering of the remaining transactions in that ordered list.

[0055] When a miner, such as miner A, receives a block header message from, for example miner C, indicating that a block has been found, the miner A builds the Merkle tree based on the TXIDs and the order specified for miner C. Miner A hashes the block header to verify the hash value. If the block is verified, then miner A constructs the full block with transaction data in the specified order, adds it to the blockchain, and builds a new candidate block to continue mining.

[0056] In some example implementations, the block header sent by a successful miner does not contain all fields. For example, it may contain only the nonce and the timestamp, along with the hash value obtained by miner C when hashing the full block header. The receiving miner may add the missing fields, such as the version, the prev_block hash, the Merkle root that the miner has calculated, and the difficulty setting. It can then validate the reconstructed header by hashing it can comparing it to the received hash value.

[0057] In some example implementations, the messaging from miners indicating their ordered list of transactions for their respective candidate blocks is sent automatically once a candidate block is built. In some other example implementations, the ordered list information is provided in response to a request from another miner.

[0058] In some implementations, a template identifier $TmID$ may be defined. The template identifier is specific to a miner and to a block level, so that it effectively refers to a specific candidate block, *i.e.* a specific ordered list of transactions. Any messaging from a miner relating to its ordered list of transactions may include the template identifier to ensure that receiving miners associate the changes with the correct specific ordered list of transactions. In one implementation, the template identifier may be a hash of a miner identifier (miner ID) and the prev_block field in the block header, *i.e.* the hash of the previously block:

$$TmID = H(\text{prev block} || \text{miner ID})$$

[0059] This ties the template identifier to a block level and specific miner.

[0060] Each message relating to ordered lists of transactions may include the template identifier, and may further include a sequence number. The sequence number may be an unsigned integer value used to indicate the order of messages from one miner relative to one another. Sequencing is helpful to a message receiver to uniquely determine the $TXID$ ordering for a specific miner in the case that some messages are not received or are received out of order.

[0061] As mentioned above, miner ID may be a mining node's IP address or some other unique identifier for the mining node. In some embodiments, an initial authentication operation may occur when the miner attaches to the network that results in associating a unique miner ID to that mining node. The authentication operation may include a handshaking operation in which each miner provides a public key and digital signature. Authentication is connected to verification of the digital signature. In some cases, the miner ID may be based on the public key, the digital signature, or some combination thereof.

[0062] In some instances, the handshaking operation during the authentication phase may include establishment of a shared secret amongst miners. Many techniques exist for establishing a shared secret. Subsequent messaging, such as messaging relating to the ordered list of transactions, may be encrypted for confidentiality and to reduce the possibility of man in the middle attacks on the mining process.

[0063] In some embodiments, mining nodes carry out an authentication procedure after attaching to the network and opening a peer-to-peer connection with each other. The authentication procedure may involve sending an authentication challenge message and receiving an authentication response. Failure to receive a valid authentication response may result in the mining node closing or dropping the connection to that other mining node. In some cases, the mining node may have a different endpoint for an authenticated connection than its usual public network connection.

[0064] In one example implementation, the authentication challenge message includes a requesting mining node sending a message to the responding mining node. The message is one that the responding mining node has not previously seen. In some cases, the message may be a random code or string. The content of the message is not significant, but the responding mining node should not have seen the content before so that it cannot re-use previously generated material.

[0065] On receiving the authentication challenge message, the responding mining node concatenates the received message with a nonce value. The nonce value may be generated by the responding mining node using any pseudorandom process. The responding mining node then signs the concatenated nonce and message using the private key of a public-private key pair previously published. In these examples, the public-private key pair is related to the miner identity. In some cases, the public key may be the miner ID used in communications on the blockchain network. The requesting mining node may have a whitelist or other record of miner IDs (public keys) for mining nodes attached to the blockchain network.

[0066] Once the responding mining node has signed the concatenated nonce and message using its private key, it then generates and sends the authentication response message. The

authentication response message may include the public key, *e.g.* the miner ID, the nonce value used, and the digital signature generated. The requesting mining node receives the authentication response message and may verify, using the public key, that the mining node is the owner of the public key and the public key matches with a key on the whitelist corresponding to a specific mining node.

[0067] The authentication response message may be sent with a corresponding authentication challenge message, so that the responding mining node may reciprocally authenticate the identity of the requesting mining node.

[0068] In one example, the format of the authentication challenge message may be of the form:

```
###   'authchallenge'
...
version: varint,
message_length: varint,
message: byte[]
...
```

[0069] In one example, the format of the authentication response message may take the form:

```
###   'authresponse'
...
public_key_length: byte,
public_key: byte[],
client_nonce: byte[32],
signature_length: byte,
signature: byte[]
...
```

[0070] In some implementations, the public key is an ECDSA key on the secp256k1 elliptic curve. The signature may be a signature over a concatenation of the nonce value and the content of the message field from the authentication challenge message. The concatenation may be (nonce|message) or (message|nonce), depending on the implementation.

[0071] Reference is now made to Figures 3A-3I, which diagrammatically illustrate example operating state and messaging flow between two miners in accordance with an implementation of the present application.

[0072] In Figure 3A, it will be seen that a first miner 302, Miner A, and a second miner 304, Miner B, are part of a blockchain network, and that there is an existing blockchain 306. The first miner 302 and the second miner 304 both have a locally-stored copy of the blockchain 306 in this example. The blockchain 306 is of a certain “height” or block level.

[0073] As indicated by Figure 3B, each of the miners 302, 304 select a set of transactions from their respective mempools to build a candidate block. Specifically, the first miner 302 builds a first candidate block 308 containing a first ordered set of transactions, and the second miner 304 builds a second candidate block 310 containing a second ordered set of transactions. The first ordered set and the second ordered set may or may not contain the same transactions, and may or may not have them in the same or partially the same order, since the miners 302, 304 are free to select whichever transactions from the mempool that they wish and group them in whatever order they wish.

[0074] As shown in Figure 3C, the miners 302, 304 begin mining the respective candidate blocks 308, 310 in an attempt to find a nonce that will result in a hash of the block header that is below the threshold value set by the difficulty.

[0075] While searching for a successful proof-of-work, the miners 302, 304 exchange information about the ordered set of transactions in their respective candidate blocks 308, 310. As Figure 3D illustrates, the first miner 302 may send an “add” message 312 to the second miner 304 providing the first ordered list of transactions from the first candidate block 308. From that information, the second miner 304 may build a template 314 of the first candidate block 308. In some example implementations, the template 314 may be the first ordered list of transactions in TXID form. In some example implementations, the template 314 may be a table, an example of which is provided above. In yet other example implementations, the template 314 may be a copy

of the full candidate block, excluding certain unavailable header fields, including full transaction data.

[0076] In a similar manner, the second miner 304 may send the first miner 302 an “add” message 316 containing the second ordered list of transactions from the second candidate block 310, as illustrated in Figure 3E. From that data, the first miner 302 builds a template 318 of the second candidate block 310. Now each miner has their own candidate block which they each continue to mine, and a template of the candidate block being worked upon by the other miner.

[0077] Figure 3F shows that the first miner 302 has succeeded in finding a block header hash that meets the proof-of-work requirement. Accordingly, the first candidate block 308 with the most recently tested nonce in its header results in a valid block. Figures 3G and 3H then show that the first miner 302 adds the new block 322 to the blockchain 306 and transmits a message containing the coinbase transaction 320 and a message containing the block header information 324. The block header information 324 includes, at least, the nonce and a timestamp. It may or may not further include the other block header fields, such as the merkle_root field. In some cases, this may be sent as part of the block header information 324, even though it can be calculated by the second miner 304, to enable the second miner 304 to double-check its Merkle root calculation. In some cases, the first miner 302 sends the second miner 304 the full block header for the new block 322. The first miner 302 may also send the second miner 304 the hash value obtained from hashing the block header so that the second miner 304 can not only verify the hash falls below the difficulty threshold but also that it matches the hash that the first miner 302 purports to have found. The hash value may be sent in the coinbase transaction message 320, in the block header information message 324 or in a separate message.

[0078] Once the second miner 304 has the coinbase transaction, it can complete the transaction portion of the template 314 of the first candidate block, and can therefore calculate the Merkle root. From that calculation it can verify that the merkle_root field is accurate, if that field was included in the block header information message 324. If not, then it can complete the merkle_root field and any other missing fields, like the version, prev_block value, and bits field, in order to assemble the block header 326 for the new block 322. The other fields of the block

header 326, like the nonce and timestamp, are provided by the block header information message 324. The second miner 304 may then validate the new block by hashing the assembled block header 326.

[0079] Assuming that the assembled block at the second miner 304 is validated, the second miner 304 then adds that new block 322 to its copy of the blockchain 306, as shown in Figure 3I.

[0080] Although the above-described sequence shows the coinbase transaction message 320 being sent prior to the block header information message 324, it will be understood that in some implementation the block header information message 324 may be sent first, or that both messages may be combined into a single message. Other variations in the order of operations or implementation details of a specific instance may be made without altering the overall functional operation of the described system and method.

[0081] It may be appreciated by those skilled in the art of blockchain networks that in some cases a block may be mined earlier than anticipated. In Bitcoin, for example, the difficulty setting provides for discovery of a valid block approximately every ten minutes, but the timing is not certain. It is possible that a valid block may be found much earlier. Moreover, some messages propagating transactions or transaction ordering information may encounter network delays in propagating through the network.

[0082] Accordingly, the possibility exists, albeit unlikely, that a valid block is found by a miner and the header information is sent to other miners, but at least one of those miner does not have all the transactions included in the block or has an incomplete or incorrect ordered list of transactions for the successful miner. When that miner attempts to validate the Merkle root field of the new block header, a mismatch will be found and the block considered invalid. One option is that the miner considers the block invalid and discards it. However, in another embodiment, the miner may send a request message to the successful miner for the set of Merkle leaves, *i.e.* the ordered list of TXIDs. Upon receiving the sent of leaves, the miner may then determine where the error in Merkle root calculation arose. If the cause is incorrect or incomplete ordering of transactions, the miner may update the ordering and verify the new block. If the cause is a missing

transaction, the miner may request the new transaction and continue the verification process. Note that in some embodiments, a missing transaction may be detected prior to successful discovery of the block if a miner that receives an ordered list of transactions for another miner's candidate block finds that it does not have one of the TXIDs in its mempool. In that case, the receiving miner can request a copy of that transaction and update its mempool.

[0083] In some implementations, one or more miners may wish to maintain some confidentiality over its transaction selection and ordering. In such implementations, it may be possible for a miner to encrypt its ordered list of transactions when distributing it during the mining phase. If a valid block is found, then the miner distributes a decryption key together with the header information and/or the coinbase transaction, then permitting the receiving miners to decrypt the ordered list of transactions and validate the new block.

[0084] The following table describes one set of example messages for one illustrative implementation of the messaging for propagation of ordered transaction lists amongst miners:

Value	Field Size (Bytes)	Data type	Name	Description
0	32 + 4	char[32] + varInt	get_chunk	Request for chunk of <i>TXIDs</i> The first 32 bytes is a template ID and the 4 bytes is a sequence number.
1	32 + 4 + 4 + 32 * N	char[32] + varInt + varInt + char[32]	add	Add a list of <i>TXIDs</i> (chunk) to the message sender's candidate block. The first 32 bytes is the template ID, the second 4 bytes is the sequence number. The third 4 bytes is the length of the rest of the message containing the N <i>TXIDs</i> chunk payload.

2	$32 + 4 + 4 + 4 * N$	$\text{char}[32] + \text{varInt} + \text{varInt} + \text{char}[32]$	cadd	Add a list of transaction hashes (chunk) to the message sender's candidate block. The first 32 bytes is the template ID, the second 4 bytes is the sequence number. The third 4 bytes is the length of the rest of the message containing the N compressed <i>TXIDs</i> chunk payload.
3	$32 + 4 + 4 + 32$	$\text{char}[32] + \text{varInt} + \text{varInt} + \text{char}[32]$	replace	Replace a transaction hash in the list of message sender's transactions. The first 32 bytes is the template ID, the next 4 bytes is the sequence number. The next 4 bytes indicates the position of the tx hash within the chunk. The last 32 bytes is the replacement tx hash
4	$4 + 4 + 4$	$\text{char}[32] + \text{varInt} + \text{varInt}$	creplace	Replace (compressed) transaction hash in the message sender's candidate block. The first 4 bytes is the tx hash and the last 4 bytes is the sequence number indicating the position of the tx in the message sender's block.
5	$4 + 4$	$\text{varInt} + \text{varInt}$	delete	Delete the tx hash in the position indicated by the 4-byte int

6	80	char[32]	block_header	Block header data packet. This notifies a peer of a new block header. The block header fields are converted to string variables and concatenated into a single 80-byte string. This message is sent in conjunction with a valid coinbase_tx message.
7	111-120	char[32]	coinbase_tx	Message containing the (raw) coinbase tx for a successful miner. This message is sent in conjunction with a block_header message.

[0085] A receiving miner modifies the candidate block template corresponding to the template identifier in the message. Messages received out of sequence, according to sequence number, for a particular template identifier may be queued until intervening messages are received.

[0086] In all likelihood, the receiving miner has the applicable transactions corresponding to the TXIDs in its mempool, but if a transaction is missing, then it can request a copy of any missing transactions.

[0087] It will be appreciated that the above-described illustrative messaging protocol is one example implementation.

Block Propagation with Poisoned Transactions

[0088] The early propagation of candidate block details from one miner to another miner may result in another miner relying on the one miner's validation work. That is, each mining node is supposed to validate transactions before adding them to the mempool, and to construct a candidate block from those validated transactions. If a first mining node constructs a candidate block from validated transactions, and then sends the block header data and the ordered set of transaction identifiers to a second mining node, that second mining node could conceivably choose

to rely on the fact that the underlying transactions are valid because the first mining node would have already done that validation work or the first mining node would not have included them in the candidate block that it is mining. This allows the second mining node to avoid the computational cost of validation work. If enough mining nodes start to piggyback on the validation work of other mining nodes, the consensus-based security of the blockchain network may be undermined.

[0089] Accordingly, in one aspect the present application provides for methods and systems of block propagation structured to prevent such reliance. In particular, a mining node that creates an ordered set of transactions for a candidate block may place one or more invalid transactions in the ordered set that is propagated to other mining nodes. If the candidate block is successfully mined, then the mining node reveals which of the transactions in the ordered set are invalid.

[0090] In this manner the second mining node cannot rely on the propagated block information from the first mining node as identifying validated transactions, since the information has been seeded with at least one invalid transaction. The invalid transaction may be referred to as a “poisoned” transaction, and the set of ordered transactions modified to include an invalid or poisoned transaction may be referred to as a “poisoned” ordered set.

[0091] The first mining node does not mine a candidate block containing the poisoned ordered set. Instead, it constructs the candidate block containing an ordered set of validated transactions. It is the block header for that valid candidate block that the first mining node attempts to mine through iterating through nonce values, and changing timestamp and or coinbase transaction data. However, the block data that is forwarded to other mining nodes during the pre-proof-of-work block propagation is the poisoned ordered set information.

[0092] In one implementation, the first mining node may reveal the index or indices for the invalid transactions after successfully mining a block by sending that index/indices data with the POW solution data, *e.g.* the nonce, timestamp and coinbase data for the solution. In another implementation, the first mining node may prove advance knowledge of the invalid transactions

through providing the list of index/indices of the invalid transactions in advance. The list may be encrypted to prevent the receiving mining nodes from repairing the poisoned ordered set immediately, and the first mining node may provide a decryption key for decrypting the encrypted list if the first mining node finds a POW solution for the candidate block.

[0093] The invalid transactions added to the poisoned ordered set may be transactions created by the mining node that are constructed to violate at least one validation criterion. In some cases, the invalid transaction may be one that the mining node has received from another mining node in that other mining node's poisoned ordered set. By re-using an invalid transaction that was used by other mining nodes, the mining node avoids recipients identifying invalid transactions solely on the basis that they have only been seen from that mining node as the single source.

[0094] The encrypted list may identify the invalid transactions in the poisoned ordered set by their index in the set. In some implementations, the encrypted list further provides a replacement valid transaction for each invalid transaction. By replacing the invalid transactions with valid transactions, the poisoned ordered set is restored to the ordered set that matches the candidate block. The receiving node may then fully construct the Merkle tree and validate the POW solution. By providing replacement transactions instead of simply removing the invalid transactions, the structure of the Merkle tree is maintained so that at least some of the validation and Merkle hash work can be done by the receiving node in advance of receiving the POW solution.

[0095] Reference is now made to FIG. 4, which shows, in flowchart form, one example method 400 for block propagation in a blockchain network. The method 400 in this example is implemented by a mining node. The mining node may be a network-connected computing device configured to mine blocks in accordance with a governing blockchain protocol. The mining of block may include building a candidate block and then hashing the candidate block header until a POW solution is found or notification is received that another mining node has succeeded in mining its own candidate block. The method 400 may be implemented using processor-executable instructions stored in memory that, when executed by a processor, cause the processor to carry out the operations described below.

[0096] In operation 402, the mining node builds a candidate block containing validated transactions. The transaction may be validated transactions stored in the mining node's mempool of unconfirmed but validated transactions. It may also begin mining the candidate block as soon as the candidate block is constructed, as indicated by operation 404. Mining involves hashing the candidate block header, determining whether the hash result is lower than the difficulty threshold and, if not, then iterating a nonce value in the header or updating the timestamp or changing the coinbase transaction to change the content of the header, and then repeating. It will be understood that the candidate block contains an ordered set of transactions.

[0097] In operation 406, the mining node replaces one or more of the valid transactions in a copy of the ordered set of transactions with one or more invalid transactions to create a poisoned ordered set of transactions. The invalid transactions may be created by the mining node or may be copied from invalid transactions sent by another mining node. The mining node then creates a list of indices identifying the invalid transactions in the poisoned ordered set of transactions, as indicated by operation 408.

[0098] In this example, the encrypted list may also include replacement data. The replacement data may be TXIDs for the valid transactions that the invalid transactions have replaced. In some cases, the replacement data may be the valid transactions themselves. The replacement data allows a recipient to "repair" the poisoned ordered set of transactions by removing the invalid transactions and replacing them with the respective valid transactions so as to reconstruct the correct ordered set of transactions that is the basis for the candidate block. The list of indices of invalid transactions is encrypted by the mining node.

[0099] In some other examples, the encrypted list does not contain the replacement data, and the replacement data is provided by the mining node with the POW solution if the mining node manages to find a POW solution.

[0100] In operation 410, the mining node propagates the poisoned ordered set of transactions and the encrypted list to other mining nodes. The mining node may employ a

messaging protocol such as that described above for signaling transactions to be added, deleted, or replaced in the poisoned ordered set.

[0101] Having signaled its poisoned ordered set to other mining nodes, the mining node continues to search for a POW solution to the candidate block, as indicated by operation 412. It also monitors for notification that another mining node has found a POW solution to its own candidate block, as indicated by operation 414. If another node finds a valid block, then the mining node returns to operating 402 to build a new candidate block and try again. If the mining node finds a POW solution, then, in operation 416, it propagates the solution and a decryption key for the encrypted list. The other nodes on the blockchain network may then decrypt the encrypted list, repair the ordered set of transactions to build the candidate block, finish validation of the candidate block, and further propagate the solution and decryption key.

[0102] FIG. 5 shows an example method 500 of block validation by a node in a blockchain network. The method 500 in this example may be implemented by a mining node. The mining node may be a network-connected computing device configured to mine blocks in accordance with a governing blockchain protocol. The method 500 may be implemented using processor-executable instructions stored in memory that, when executed by a processor, cause the processor to carry out the operations described below.

[0103] The method 500 begins with receiving candidate block template data and an encrypted list from a mining node in operation 502. The candidate block template data may be data detailing an ordered set of transactions. In this example, the mining node is configured to send a 'poisoned' ordered set of transactions, meaning that some of the transactions are invalid transactions. The candidate block template data may be received based on a message protocol like the one described above in which transactions may be added to, removed from, or replace within a candidate block template associated with a specific mining node. The encrypted list, once decrypted, will identify the poisoned transactions within the candidate block template data.

[0104] In operation 504, the node may determine whether it has copies of the identified transactions. Copies of the transactions may be included in the messages themselves in some cases,

or may be obtained from the node's local mempool. In some cases, the node may not have a copy of a transaction in the candidate block template and may, in operation 506, request copies of any missing transactions from the mining node.

[0105] In operation 508, the node validates any transactions that it had not already validated. If any are identified as being invalid, the node may flag those or otherwise note them as invalid, as indicated by operation 510. If the mining node is using the poisoned transaction block propagation protocol, which may be signaled in the messaging relating to propagation of candidate block templates, then the node will be on notice that invalid transactions may be intentionally placed in the ordered set. In addition, the node may have received the encrypted list, which implicitly signals that there will be at least one invalid transaction in the ordered set.

[0106] In this example, in operation 512, the node receives a notification that the mining node has found a POW solution. The notification may include the POW solution, including the nonce used, the timestamp, and any other data needed by the node to reconstruct and validate the candidate block. The notification further includes the decryption key so that the encrypted list can be decrypted, which the node does in operation 514. In some example implementations, the notification further includes replacement data indicating what valid transactions should be used in place of the invalid transactions. In some cases, the replacement data is included in the encrypted list and revealed once decrypted.

[0107] The decrypted list reveals the indices of the invalid transactions. The node also has replacement data either from the decrypted list or from the POW solution notification. In operation 516, the node determines whether the indices identified in the decrypted list match the transactions that the node identified as invalid in operations 508 and 510. If not, then in operation 518 the block is rejected as invalid. That is, if the node identified an invalid transaction in the candidate block template data and that same transaction is not identified as invalid in the decrypted list, then the block contains an invalid transaction and should be rejected. In some cases, if the decrypted list identifies a transaction as invalid that the node determined was valid, then the replacement may be made and the block may still be accepted provided it meets all validation criteria.

[0108] In operation 520, the node replaces the invalid transactions with valid transactions based on the replacement data in the decrypted list. It then completes validation of the block in operation 522. In particular, with the valid transactions inserted in place of the invalid transactions, the node may complete building of those portions of the Merkle tree, find the Merkle root, complete the block header, hash the header, and confirm that the hash is below the difficulty setting.

[0109] In some implementations, the poisoned transactions indices in the decrypted list are an array of varint indexes. The value 0 may be reserved as a dummy entry that can be ignored by the node. The mining node may insert 0 value entry or entries to obfuscate the number of poisoned transactions in the block by padding the data.

[0110] In some embodiments, the replacement data is sent with the POW solution and decryption key instead of being included in the encrypted list.

[0111] One illustrative example of a message structure for sending a POW solution and decryption key is:

```
sbt_finalize
    template_id_len: varint
    template_id: byte[],
    sequence_no: varint,
    aes_128_encryption_key: byte[16],

    block_header: byte[80],
    coinbase_tx_len: varint,
    coinbase_tx: byte[],

    aol_entry_count: varint,
    aol: aol_item[]

    poison_tx_count: varint,
    poison_tx_replacements: aol_item[]
```

[0112] In the above example, the template_id_len, template_id, and sequence_no fields identify the template to which the message relates and ensure proper sequencing among messages

relating to the template. The aes_128_encryption_key field stores the decryption key for enabling decryption of the encrypted list. In some cases, the decryption key may also be used to decrypt encrypted AOL data.

[0113] The block_header and coinbase_tx fields store the data needed to complete the candidate block. The aol field provides any additional transactions to be added to the template; this field may be empty.

[0114] The poison_tx_replacements field contains replacement data, *i.e.* the transaction data for replacing the invalid transactions revealed in the encrypted list.

[0115] It will be appreciated that the foregoing message structure is one example. Other message structures may include additional fields, alternate fields, or fewer fields, depending on the implementation.

[0116] It will also be appreciated that it may be that some or all of the above-described operations of the various above-described example methods may be performed in orders other than those illustrated and/or may be performed concurrently without varying the overall operation of those methods.

[0117] Reference is now made to Figure 6, which shows, in block diagram form, a simplified mining node 600, in accordance with an example of the present application. The mining node 600 includes a processor 602, which may include one or more microprocessors, application specific integrated circuits (ASICs), microcontrollers, or similar computer processing devices. The mining node 600 may further include memory 604, which may include persistent and non-persistent memory, to store values, variables, and in some instances processor-executable program instructions, and a network interface 606.

[0118] The mining node 600 may include a processor-executable blockchain application 608 containing processor-executable instructions that, when executed, cause the processor 602 to carry out one or more of the functions or operations described herein.

[0119] The various embodiments presented above are merely examples and are in no way meant to limit the scope of this application. Variations of the innovations described herein will be apparent to persons of ordinary skill in the art, such variations being within the intended scope of the present application. In particular, features from one or more of the above-described example embodiments may be selected to create alternative example embodiments including a sub-combination of features which may not be explicitly described above. In addition, features from one or more of the above-described example embodiments may be selected and combined to create alternative example embodiments including a combination of features which may not be explicitly described above. Features suitable for such combinations and sub-combinations would be readily apparent to persons skilled in the art upon review of the present application as a whole. The subject matter described herein and in the recited claims intends to cover and embrace all suitable changes in technology.