Critical Scrutiny of Sybil Attack Resistance Protocols based on Blockchain

Om Brahmbhatt[1], Shrey Arora[1], Trupal Chaudhary[2], Shashank Mistry[1], Qusai Onali[1], Dharven Doshi[1]*

[1]B. Eng. Student, Department of Information Technology, GCET, Anand, India
[2]B. Tech. Student, U & P.U. Department of Computer Engineering, CSPIT, Anand, India

---***---

Abstract - *Resistance to Sybil attacks is becoming increasingly relevant as decentralized systems such as cryptocurrency are rising in popularity. As far as we know, avoiding Sybil attacks in a decentralized manner is not feasible due to the fact that IP addresses do not correspond to individuals. However, there are two basic techniques to reducing the risk of Sybil attacks: either making it difficult for a single person to control a large number of peers or detecting abnormal Sybil attack activity patterns in the decentralized system and ejecting malevolent nodes. This paper focuses on the first approach and rely on the complexity of some computations. Indeed, if joining a peer-to-peer network requires a lot of processing, such as solving a crypto puzzle, joining a large number of peers becomes prohibitively expensive that an attacker will relinquish to even try to endeavor the attack.*

Key Words: Blockchain, Sybil Attack, GO, Proof of Work, Cryptopuzzle

1. Goals and Functionalities

The purpose of the first section of the paper is for joining peers to submit a PoW (proof-of-work) in order to be accepted. We decided to design a protocol that would allow us to generate a crypto puzzle and ask the peer who wanted to join for the solution. The solution should be a time-limited valid token that validates network access eligibility.

To put it another way, each joined peer should keep records of the IDs of the accepted nodes. A new peer N, who does not have an ID, sends a join request to an existing peer P. P delivers a crypto puzzle to N along with an available ID and timestamp. This crypto puzzle should not rely exclusively on P, because solving a crypto puzzle produced by P is straightforward if P and N are both malevolent and cooperate. Then N solves the crypto puzzle, with the solution serving as its proof-of-work and sends it back to P for verification. After then, all peers should be aware that N has joined the network. The timestamp will be included in the crypto puzzle solution, ensuring that all peers have the same ID with the same creation timestamp. These IDs will become obsolete once a certain amount of time has passed, and the peers will have to show a new PoW.

The next goal is to ensure that communication inside the system is predicated on the possession and validity of such a token after we have that ID based on a proof-of-work. Each token should be associated with a single user; no other user should be able to decipher another's token. The following is the second section of the paper.

To put it another way, every time a peer A wishes to connect with a peer B (either to transmit a rumor, a private message, or whatever), B should make sure that A is one of the accepted peers who has shown proof-of-work that has not yet expired. A hostile peer M should not be able to communicate as A, hence B should be able to validate A's signature in some way. Another requirement is that a malicious peer M who has demonstrated PoW should not be able to interact from many locations at the same time using its ID. As a result, peers should be aware of which peers are now active in order to decline contact from a peer who is currently active in another area.

2. Related Work

As previously mentioned, there are a variety of ways to detect or prevent Sybil attacks, such as trusted certifications or social graph patterns, but here we'll focus on the many methods for imposing a high cost to join the system, hence reducing the likelihood of Sybil attacks. There are a variety of ways to establish incentives for adversaries to refrain from executing Sybil attacks. For instance, the Dash cryptocurrency requires to pay $ 13,000 to attain a master node. Another option is to involve CAPTCHAs, which are apparently difficult for a machine but not for a human to solve but completing a large number of them would take a long time even for humans, especially if the system requires recurrent solutions.

This paper [2] describes a method for a joining peer to solve crypto challenges without relying on a single peer or all peers to do so (the first case is insecure if that peer is malevolent, and the second case is practically infeasible because the probability that all peers are active is extremely low). A tree hierarchical structure is used to organize the nodes. To connect, a peer must locate a leaf node and request a crypto challenge from him. When this is completed, the joining peer receives a token that allows him to request another crypto-challenge from the parent node. This process continues until the peer solves the root node's challenge, at which time the peer is permitted into the network. This has the advantage of distributing burden and trust among diverse peers, but it necessitates a hierarchy and, most critically, a single trustworthy root node. Because we're trying to develop entirely decentralized systems, this isn't a desirable trait.

Another strategy is described in this publication [1] The topology of a chord is used to define the neighbors of peers. The concept is to broadcast challenges to all other peers, and these challenges must be created by all peers. They generate puzzles by concatenating the ids of peers, and the non-invertibility property of hash functions ensures that a challenge cannot be computed without the challenges from other peers. Peers ping each other on a regular basis to determine which peers are online and must participate in the puzzle computation process. Each peer can check to see if a new challenge has its own challenge. This time, the solution is a completely decentralized system with no single point of failure. It ensures that all active nodes compute a challenge. The nodes that are down at that time, however, are not part of the computation. Another drawback is that the communication protocol, with all its pings and broadcasts, is highly bandwidth intensive.

Bitcoin is probably the most closely linked work because it makes use of the blockchain technology that we will employ in this paper and detail in the next part.

3. Background

The blockchain technology is used in the initial portion of the paper's design and implementation. Blockchain is a continuously expanding distributed list of records organized into 'blocks' and linked by hashing. In essence, each block contains some data relevant to the blockchain application, as well as two additional fields: a nonce and a hash. Every peer in a peer-to-peer system has a local copy of the entire blockchain. New data can only be added in a block that references the prior block before being added to the blockchain. The hash field within a block is the hash of the preceding block in the chain, allowing you to verify that one block is the successor of another. In order for the blockchain to be genuine, each block's hash must match some pattern (often a number of leading zero bits in the hash). This is where the nonce field comes into play: to validate a block containing data and the hash of the previous block, the hash of the block must fit the desired pattern by experimenting with different nonce values. Due to the non-invertibility of hash functions, the only way to find a correct hash is to use brute force with various nonce values. Any peer wishing to add a block to the chain will receive a proof-of-work.

For digital signatures, the second portion of the paper will use asymmetric cryptography and hash functions. This is used for communication authentication. When A delivers a message to B, B wishes to double-check that the sender is A. A could do this by 'signing' the communication. A has a public-private key pair, which it uses to hash the message it wishes to transmit and then 'decrypt' it with its private key. This signature is then sent along with the message. B encrypts the signature with A's public key at arrival. The signature is validated if the result is equivalent to the message's hash. Different asymmetric cryptosystems, such as RSA or El-Gamal, can be used to do this. In the case of RSA, the message m is hashed to H(m) and 'decrypted' to sig = (H(m) d) mod N with the secret exponent d and public parameter N. By testing if (sige) mod N = H, the receiver checks the signature with the public exponent e. (m).

4. Design and Architecture

4.1 Blockchain-based joining

A local copy of the blockchain will be available to all peers. This storing of nodes is indeed unique and not changed as required, based on the hash-based linking of blocks. Furthermore, because each block is dependent on a peer and the previous block, we assure that the crypto puzzle cannot be forged by a single peer; instead, it will be determined by the last block and transitivity across all blocks. Because each peer has a complete copy of the blockchain, only one peer needs to be active at the time of a joining request.

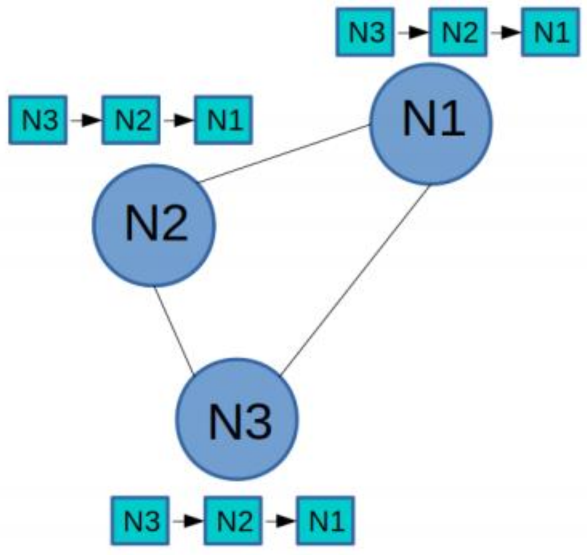

Figure 1. Blockchain replicated in the network

However, this architecture is insufficient because a malevolent peer can listen in on conversations and know the IDs of peers who have previously joined. When a network peer A leaves, a malevolent peer M can use A's ID to interact. M can utilize A's ID even while A is active by dropping its packets if he can undertake a man-in-the-middle attack. To avoid this, we'll need authentication. As a result, we'll need to include a new field in each block: the peer's public key. This will enable each peer to confirm that a peer claiming an ID has the associated private key and is thus the correct peer. This can be done with digital signatures using either a challenge-response system or HMACs, as we'll see in the next section.

Here's how the joining protocol works:

1. A sends B, an already-joined peer, a rumor or private communication m. Every gossip packet now includes a new field: the node's ID.

2. When B notices that A's ID isn't in the blockchain, it sends A an ID, a timestamp, and the last block's hash

3. A creates a pair of public and private keys.

4. A creates a new block with the following fields: ID, timestamp, pub key, nonce, and previous hash.

5. B receives the block from A.

6. B validates it, and if it is legitimate, B adds the block to its local blockchain and uses gossiping to broadcast the new block.

7. Now that A is permitted to join the network, B delivers the entire blockchain to A and treats future gossip packets normally.

8. A verifies the blockchain's integrity, and if it is correct, A considers itself to be a member.

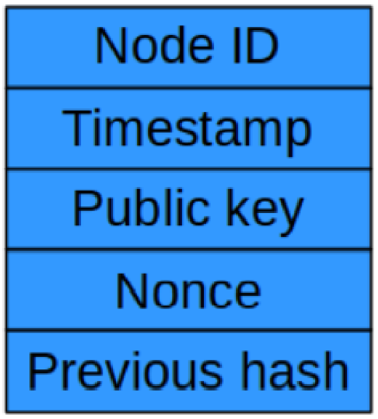

Figure 2. Block Fields

B will see that A is in the blockchain at some point later in step 2, but that the timestamp has expired. The procedure is the same in this situation, and we follow the same stages, which means A will have to mine a new block. This makes it more difficult for an attacker to maintain control of a large number of peers: to maintain control of N peers, the adversary must mine N blocks every t time units, where t is the expiration delay. Consider the scenario of establishing a network: suppose A and Bare not connected to any network and want to interact. Because there is no blockchain yet, A can issue a join request to B as in step 1), but B will not advance to step 2) because there is no blockchain yet. B can then proceed to step2after creating its own block, the beginning of the blockchain (the genesis block, with any value as the previous hash). We've added a new flag genesis to the gossiper command, which can be set to true if the node wants to start the blockchain. Then when it has mined the first block will it be considered joined, and only then will other nodes be permitted to join. To keep the architecture simple and avoid some issues, the genesis node is not treated as a special node later on; instead, it behaves like the others, which means its genesis block will expire at some point. [5] So, we assume that there is always at least one joined peer in the network, because otherwise the following could happen: node A mines the genesis block, no one tries to join, and the block expires at some point, then B tries to join to A, but A is no longer joined, and no one can join. This is a design limitation, but the assumption appears plausible, and if the expiration period is long enough, this is unlikely to occur. It should be noted that this method does not allow for the merging of many peer-to-peer networks. [6] A single peer can join any network or construct one of its own and wait for additional peers to join, but a group of peers cannot merge their blockchains with those of another peer-to peer network [3]. Because nodes would try to connect at the same time to different points in the network, we predict blockchains to diverge a lot in the network. Collisions, on the other hand, are unnecessary. If two blocks are mined at the same time, one of them will bedropped at each node. There will be two copies of the blockchain, but this is unimportant because nodes only need to be joined by their neighbors. We merely had to change the node's ID field in every forwarded gossip packet so that the node's ID field in every incoming packet was that of a neighbor. We also don't have to preserve all the blocks in the blockchain; we can delete the ones that have expired. This has no bearing on the blockchain's integrity because if one block expires, all prior blocks will likewise expire. As a result, whenever a peer joins, it receives A blockchain that begins with the first legitimates block and does not include any subsequent blocks. The benefit is that we can always have a blockchain with only n blocks for a network of n peers.

4.2 P2P Authentication

Once a peer has completed a PoW, it is registered in the blockchain, and we must verify that it can communicate using that registration at any point before the block's timestamp expires. In other words, if a node A leaves the network and returns through anode B, it will submit its ID, which B will check is available in the blockchain. [4] Now, we want to be sure that a malicious peer M can't use A's ID to connect as A, so A uses its private key to sign every packet it transmits to B. With A's public key, which is stored in the block with A's ID, B can check the authenticity and integrity of each packet.

This ensures that A can connect to the network at any time and from any location, and that any joined peer can verify that it is truly A. However, this is insufficient because a malicious peer M might perform a PoW while connecting through B and communicating regularly with other peers C, D, and so on. We'll assume that the attacker can speak with different peers using different IP addresses. Each peer observes M acting normally, but they will be unaware that M has joined the network several times unless they exchange their data.

Each peer observes M acting normally, but they will be unaware that M has joined the network several times unless they exchange their data. Currently, each peer is aware of its immediate neighbors, but it must now be aware of all active peers in the network. To-do this, each peer should keep a local list of active peers and send a specific packet to each of its neighbors whenever a new peer joins the network. When a peer receives such a packet from a neighbor, the new peer is added to the peer's own list [7]. We can now share a list of all current nodes in the network, but we need to be able to remove a peer from the list if it fails or exits the network. We'll need nodes to periodically Ping each other to see whether they're still alive for this to work. Fortunately, the anti-entropy system, which transmits status messages on a regular basis, already does this. So, if a node A has not received any status messages from a neighbor B after a period of time T, A will deem B inactive and delete it from its list of active nodes. A will additionally broadcast the fact that B is no longer operational in a special packet. The other nodes will also remove B from their list after they receive this packet.

If the malicious peer M talks with B, B will not only verify the blockchain ID and the digital signature's validity, but it will also verify that M is not already on the list of active peers. If B refuses to allow M, add it to the active peers list and disseminate the change. Now, if M tries to talk with C, C will refuse because M is already on the list of active peers.

To summarize, whenever node A sends packet m to node B, node B must complete the following checks:

1. Is A's ID a portion of the blockchain that is valid? If not, continue with protocol 5.1; if yes, move on to check 2.

2. Is m's signature valid when utilizing A's public key from the block? If not, reject m; if yes, move on to step 3.

3. Is A already part of the list of active peers? If yes reject m, if not add A to the list and propagate the information that A has joined to the neighbors, then treat m normally.

B simply needs to check the signature on the subsequent packets.

4.3 Implementation Details

We'll give a quick overview of the Go code we implemented in this section to make it easier to navigate. In essence, every GossipPacket now has a new field called NodeID. It's only going to be utilised for this paper. Every peer will now call a function handleGossipPacket on every GossipPacket reception, which will return a boolean indicating whether the peer should accept or drop the packet. This function will determine whether the NodeID is valid on the local blockchain and then follow the protocols described in sections 5.1 and 5.2. All message structures, handling, and other details for the protocol specified in 5.1 are contained in the file puzzleshandler.go. The file blockchain. go implements the blockchain and offers functions for all relevant blockchain operations. The file digital signatures.go is used to construct and manage the digital signature security protocol. The channels and code for the technique to define inactive nodes can be found in the file node communication.go.

5. Evaluations

To compare the joining times of a regular peer and a set of Sybil peers, we will establish alternative p2p topologies and measure the joining time of a normal peer and a set of Sybil peers. The usual joining time should be affordable, whereas the Sybil joining time should be prohibitively costly. We'll experiment with different parameter values and compare the results. Following the creation of the genesis block, we measured the joining time for various levels of difficulty:

• Difficulty = 16

1. Node 1: 147.020307ms
2. Node 2: 164.127784ms
3. Node 3: 186.102790ms
4. Node 4: 214.628043ms
5. Node 5: 204.144341ms
6. Node 6: 194.548098ms

• Difficulty = 18

1. Node 1: 1.3468796s
2. Node 2: 768.457822ms
3. Node 3: 1.053998185s
4. Node 4: 1.581628689s
5. Node 5: 1.372276421s
6. Node 6: 1.472019263s

• Difficulty = 20

1. Node 1: 2.873795175s
2. Node 2: 3.093252135s
3. Node 3: 2.901926432s
4. Node 4: 2.663215389s
5. Node 5: 2.775930321s
6. Node 6: 2.823405346s

• Difficulty = 22

1. Node 1: 6.366208633s
2. Node 2: 7.196392081s
3. Node 3: 8.758302038s
4. Node 4: 12.692505636s
5. Node 5: 8.263607094s
6. Node 6: 5.443429162s

• Level of difficulty = 24 (In this scenario, we constructed a rogue node m1 that attempted to add other malicious nodes m2, m3, m4, m5, m6, m7 to the blockchain, and it took more than 10 minutes to do so, demonstrating the legitimacy of our protocol): Diagram 3

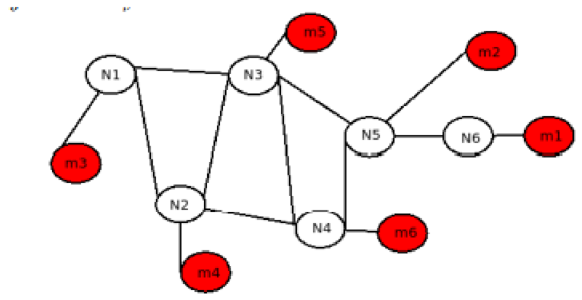

Figure 3: Topology of our system with malicious nodes attempting to join it.

1. Node 1: 45.8752036122s
2. Node 2: 47.3963730081s
3. Node 3: 47.7365930922s
4. Node 4: 49.2134378421s
5. Node 5: 53.3200982513s
6. Node 6: 58.0169600473s

6. Conclusions

In this paper we have discussed that, avoiding Sybil attacks in a decentralized manner is not feasible due to the fact that IP addresses do not correspond to individuals. There are two basic techniques to reducing the risk of such attacks. One involves making it difficult for one person to control a large number of peers. The other involves detecting abnormal activity patterns and ejecting malevolent nodes. Next goal is to ensure that communication inside the system is predicated on the possession and validity of such a token.

As a result, each token should be associated with a single user; no other user can decipher another's token. Another requirement is that a malicious peer M who has demonstrated PoW should not be able to interact from many locations at the same time.

References

[1] Nikita Borisov. “Computational Puzzles as Sybil Defenses”. In: (). doi: https://nymity.ch/sybilhunting/pdf/Borisov2006a.pdf.

[2] Patrick McDaniel Hosam Rowaihy William Enck and Thomas La Porta. “Limiting Sybil Attacks in Structured Peer-to-Peer Networks”. In: (). doi: http://nsrc.cse.psu.edu/tech_report/NAS-TR-0017-2005.pdf

[3] Srikanta Pradhan, Somanath Tripathy. "CAP", Proceedings of the 10th International Conference on Security of Information and Networks - SIN '17, 2017

[4] G. Bracha, “An o (log n) expected rounds randomized byzantine generals’ protocol,” Journal of the ACM (JACM), vol. 34, no. 4, 1987.

[5] A. Hafid, A. S. Hafid, and M. Samih, “New mathematical model to analyze security of sharding-based blockchain protocols,” IEEE Access, vol. 7, pp. 185 447–185 457, 2019.

[6] P. Otte, M. de Vos, and J. Pouwelse, “Trustchain: A sybil-resistant scalable blockchain,” Future Generation Computer Systems, 2017.

[7] J. R. Douceur, “The sybil attack,” in International workshop on peerto-peer systems. Springer, 2002, pp. 251–260.