Understanding and Detecting Majority Attacks

Thomas Cilloni*†, Xiyu Cai*‡, Charles Fleming*§, Jun Li¶
*Department of Computer Science and Software Engineering, Xi'an Jiaotong-Liverpool University, China
¶Department of Computer and Information Science, University of Oregon, United States
Email: †T.Cilloni17@student.xjtlu.edu.cn, ‡Xiyu.Cai16@student.xjtlu.edu.cn, §Charles.Fleming@xjtlu.edu.cn, ¶jun.li@uo.edu

***Abstract*—Blockchain technologies are becoming increasingly popular for financial and business solutions thanks to their many features and adaptability. However, their security is not assured. Of particular concern are majority attacks, which exploit the consensus algorithm used to resolve competing chains to modify past and present transactions on the blockchain. Despite numerous in-the-wild majority attacks, detecting in-progress attacks remains difficult, with most attacks remaining undetected for hours or days. to understand better these attacks and why they are so difficult to detect, in this paper we present a series of increasingly sophisticated attacks performed in an experimental environment and analyze the resulting network traffic. We then propose a possible detection mechanism based on the shared characteristics of these attacks. The implementation of such an algorithm is finally tested in a variety of experimental environments.**

I. Introduction

Blockchains are distributed ledgers where transactions are added in groups called "blocks." Peers use a consensus algorithm to decide which blocks of transactions are added to the blockchain. In fully decentralized blockchains, such as those used for cryptocurrencies, the most common type of consensus algorithm is *proof of work* [1], where peers in the network search for the solution to a cryptographic puzzle, with the winning node adding its block to the chain. Because there generally are many peers in the network, and because propagation of a new block is not instantaneous, in some cases, multiple solutions are found, and multiple competing chains can emerge. To resolve these cases, the consensus algorithm includes the rule that peers will always consider the longest blockchain version encountered as the "definitive" version. To resolve these cases, the consensus algorithm will pick the one chain for which the most work was done. Because each block requires a considerable amount of computing work to be mined, the chosen chain is, in most cases, the longest: this is commonly known as the *Longest Chain Rule* [2].

While the longest chain rule does resolve the issue of competing chains, it also makes these blockchains vulnerable to the **majority attack**, also known as a **51% Attack**. This attack consists of maliciously generating a chain long enough to override chains generated by honest nodes in order to control which blocks and transactions are added to the blockchain. This situation happens when an attacker controls a majority of the network's hashing power and is therefore capable of generating blocks at a faster pace than the rest of the network combined.

Once an attacker has control over which blocks are added to the blockchain, he can change the priority of transactions inserted in blocks and even override older blocks, excluding some transactions. Changing the order of transactions in newer blocks can lead to delayed transaction confirmations, but overriding older blocks allows the attacker to perform double-spending attacks [3]. double-spending attacks consist of spending currency in a transaction to acquire some goods, then rolling back the transaction to regain the same currency. double-spending attacks are performed by first renting or otherwise acquiring hash power (computing power to generate hashes [4]) greater than the current existing network [5]. The attacker then broadcasts a transaction spending his currency on the honest network while simultaneously mining a competing chain that does not include this transaction but instead includes a transaction transferring his coins to another wallet. The attacker then waits for the merchant to accept the transaction as irreversible and to transfer the purchased goods to the attacker. Once the exchange has happened, the attacker releases the competing chain into the honest network, where it quickly replaces the honest chain. The original, 'honest' transaction will then return to the pool of transactions waiting to be inserted in a block. However, this transaction will never be included in a block because the necessary coins do not exist anymore in the attacker's wallet. In conclusion, the customer has moved his coins to another address and has also received the vendor's goods at no cost.

Majority attacks are not theoretical. While one of the basic premises of blockchains is that it is not possible for a single entity to control a majority of the network's hash power, the plethora of small and medium blockchain networks coupled with the value of the cryptocurrency stored on these networks means that not only is it possible, but economically profitable to perform these attacks. Successful majority attacks have taken place against Bitcoin Gold [6], Verge [7], Ethereum Classic [8], and many other cryptocurrencies. Some of these attacks leveraged rented hash power [6] [8], while others exploited flaws in the consensus algorithm to quickly generate competing chains [7].

A common theme in these attacks is the extent of time before the attacks are detected, often hours or even days. Because the longest chain rule exploited by the attacker is normal behavior and because the network is decentralized, with no mechanism to gain a global view of what is happening on the blockchain as a whole, majority attacks can be difficult to detect. In this paper, we perform a series of increasingly sophisticated majority attacks on a network of nodes running almost identical copies of the latest Bitcoin source code,

978-1-7281-6978-1/20/$31.00 ©2020 IEEE
DOI 10.1109/DAPPS49028.2020.00002

all while capturing the resulting network traffic. We then show that while it is possible that sophisticated attacks with sufficient control over the majority network could indeed reduce the footprint of the attack, it is impossible to hide one completely. Based on the common traits among majority attacks that we have seen, we propose a mechanism to detect and warn of these attacks. This mechanism is finally implemented and tested in both a controlled, simulated environment and the actual Bitcoin network.

II. Related works

All blockchains, especially those related to finance, are vulnerable to a number of attacks that may undermine their usefulness. The most prominent is the *double-spending attack*, which consists of spending the same coins twice in two separate transactions. As coins cannot be duplicated, one of the two transactions will eventually fail, possibly tricking a vendor into thinking he has received payment, while in fact, the transaction gets dropped at a later point in time. double-spending attacks usually utilize other attacks to ensure the attacker's control of the otherwise random selection of which transaction will fail. Examples include 51%, Sybil [9], Routing [10], Eclipse [11], and DoS attacks [12].

Dishonest mining is one of the easiest attacks to perform [13], but there are a large number of other attacks that are also feasible. A slightly more dated research [14] provides deep analysis of how a number of different attacks are performed. The analysis includes *Confidential Cryptographic Optimization attacks, Selfish Mining attacks, Block Discarding*, and *Block Withholding attacks*. While the nature and functioning of each one of these attacks are well explained, there is no concrete data or evidence of what happens to the network traffic when one of these attacks is performed.

In the past few years, researchers have proposed a few solutions to some of the most common attacks in Bitcoin, one of which is related to majority attacks. Dey [15] explains how potential majority attacks could be detected by a machine learning algorithm using estimation formulas adapted from algorithmic game theory. While the idea being presented appears valid, to date, it has not been implemented, and thus, its real-world validity cannot be evaluated.

Aside from this well-known form of attack, other less frequent types of attacks to Blockchains have also been studied. A recent article [16] outlines two exploitation scenarios for the message digest collision resistance property often found in blockchains such as Ethereum. Similar to double-spending attacks, an attacker can create two contracts to be deployed to a single address. One of them is shown to a potential victim who is persuaded to spend some amount of money. Once the payment is complete, the attacker deploys the second contract, which allows them to spend the money stored in the contract without completing its terms. The team of developers behind Ethereum is currently testing a solution to this issue.

While all these attacks require some level of technical tweaking, either to clients themselves or to custom programs to generate pools, there exist forms of attacks based solely on data analysis. Recent research [17] provides a comprehensive case-study of what are described as front-running attacks. As the paper explains, front running means drawing benefit from the early access to relevant market data on an upcoming trade or transaction. Even though this is arguably a dishonest practice rather than a proper attack [18], it still poses a threat to the reliability and trustworthiness of blockchains [19]. The research also proposes three potential solutions to the issue.

There are a few active websites [20] [21] [22] that provide real-time and historical information on network activity, mining, and other variables in the Bitcoin blockchain. Some researchers have analyzed transactions and blocks propagation in the network [23], while others have focused on looking at the network data from the point of view of Bitcoin addresses and IP addresses [24]. There have also been studies on models to identify Bitcoin users and match them with their Bitcoin addresses [25]. While this data can be used for market analysis and studies on privacy, it has not been proved useful for the analysis and detection of majority attacks.

III. Goals of this research

Majority attacks are impossible to prevent completely. In the very worst case, no consensus algorithm could be proof against an attacker that controls the entire network. While it is clear how majority attacks work and what they do, it is yet uncertain how they precisely affect other clients in the network. On top of this, because of the overriding mechanism exploited by double-spending attacks, it is difficult to discern the targeted transactions and the exact time an attack takes place. Whenever an attack is successful, the honest chain is overtaken by the attacker's chain, and all useful data is lost. In order to shape an effective countermeasure, this research is divided into two main parts: a study of the network traffic and the design of a detection algorithm.

First, we have decided to take a step back and analyze in-depth a series of majority attacks to understand better what happens when one takes place. To gather relevant data, we have developed a cryptocurrency blockchain based on the original Bitcoin. Once deployed and grown sufficiently, we performed a series of 51% attacks in various ways on different copies of this blockchain and different network environments, trying to override the transaction history by double-spending. While the attack is happening, all nodes in the network are monitored, honest and malicious alike, saving logs of what happens both in the code and in the network. Finally, we analyzed these to find clues that can hint at an attack while it is taking place or shortly after when the blockchain is still in time to take action to counter the attack.

The second part of our research is therefore focused on designing a model to detect attacks on the fly. In order not to tamper the alternative Bitcoin clients, we worked on a program that runs independently and does not affect or slow down the client. This program is then deployed in a variety of environments, including the test scenarios we describe later and the original Bitcoin, to show that it detects attacks and does not give false positives.

IV. Experimental environment

A. Test code: Cillocoin

To provide the most realistic analysis of the flow of data during an attack and to test our algorithm properly, we have chosen the original Bitcoin as our testing environment. However, performing a majority attack on Bitcoin would be computationally infeasible and far from ethical, even for research purposes.

We consequently decided to fork Bitcoin and create what is often referred to as an *Altcoin* [26], an alternative Bitcoin-based cryptocurrency that we call Cillocoin. In the process of editing the source code of Bitcoin, we referred to the official documentation of Bitcoin [27], to a number of in-depth analyses of the source code [28] [29] [30] and to books and tutorials on the functioning of the client and possible modifications to it, available in specialized forums or in print [31] [32]. Our main goal when editing the source code of Bitcoin is to stop it from connecting to the actual Bitcoin network, letting us run a private network of many nodes that we can use to perform attacks ethically. The secondary reason to edit the original Bitcoin is to create a tampered version of the code that an attacker would use to perform the most stealth attack we could observe. The latter is explained in the third test case scenario.

A regular Bitcoin client bootstraps its peer discovery using *seed nodes* and *DNS seeds*, where seed nodes are a set of known Bitcoin nodes from which the current node can query for more peers, while DNS seeds are special DNS entries that return a connectable peer in the Bitcoin network on each query. In order to stop instances of Cillocoin clients from communicating with nodes running the original Bitcoin, we removed seed nodes and DNS seeds from the source code so that Cillocoin clients had no chance of finding Bitcoin nodes. We then added a single seed node, run by ourselves, that non-seed nodes could connect to in order to find one another. To further discourage connections, the ports used by the clients were also changed. To protect from the now very unlikely connection with a Bitcoin client, the initial message signature was also changed: this way, Bitcoin nodes will refuse to connect to Cillocoin nodes and vice versa. In order to mine blocks faster, the difficulty target was also adjusted so that a new block would be mined in no more than a minute when running on an average laptop CPU of the latest generation.

One final change was made to the checkpoints. If a node is started and has a list of checkpoints, it will start downloading blocks from its peers until it has downloaded all blocks in the checkpoints list. However, since nodes are not connecting to the main Bitcoin network, checkpoint blocks can never be downloaded, and the list has therefore been emptied. It can be argued that removing checkpoints increases the chances of majority attacks succeeding, opening to the possibility of having the whole blockchain replaced by a malicious one. While this is true, we are mainly interested in majority attacks that allow double-spending, and these always replace the tip of the chain only. Since a transaction is generally considered confirmed when it is in a block at a depth of at least six [33] (meaning five newer blocks follow it), majority attacks need to target the last six or more blocks only. Because the cost of renting hash power increases with the time it takes for the attack to be performed, attackers try to keep the number of blocks to override to the minimum. Hence, majority attacks are not so extended to go as deep as to conflict with checkpoint blocks, and therefore their removal is not relevant.

The last step in designing the Cillocoin was generating a new Genesis block with a more recent timestamp so that nodes could start mining from it. The genesis block has been hard-coded into the source code, and its hash value added to the checkpoints list. Mining the block separately was necessary because versions of Bitcoin from 0.13 do not have an integrated miner. We pre-mined a blockchain of Cillocoin to an arbitrary height as the basis of our experiments. All Cillocoin daemons in our testing environments begin with the same blockchain pre-mined.

Despite these changes, none of the underlying semantics of the Bitcoin network or any of the Bitcoin protocols (other than the format of the initial handshake, as discussed) was changed. As a result, Cillocoin clients work exactly as if running the original Bitcoin.

B. Test network

One important focus of this study is the flow of data over the internet while the blockchain is in different attack scenarios. In order to better simulate the real network environment, we cannot simply run nodes on one system within different processes – a full network simulation is required. We used *MiniNet* [34] as the network virtualization facility in this study. MiniNet is implemented based on the Linux kernel-driven network name space feature and provides a full-range of Python-based APIs to directly manipulate the simulated network without the need for virtual machines, which simplifies the process of our experiments.

During our study, we made use of these APIs to define our custom network topology for the respective attack design. The honest and malicious networks are formed by connecting those respective hosts to at least two different central network controllers, for example, switches, as they are present in real network interconnections. The isolation between the honest and malicious networks and its removal can then be trivially controlled by inserting and removing virtual connections between those central network controllers. In this way, the experiments can be carried out automatically by scripts, with test parameters provided by the researchers.

To remove uncertainties introduced by human error, we used a Python script to monitor and carry out the experiments. The MiniNet virtual hosts were set up for all the network partitions required by the respective attack design, then the controllers were added accordingly. Due to a limitation of the Bitcoin (and Cillocoin) daemon, we were only able to use IP addresses *not* under the same /16 subnet to enable connection between two Cillocoin nodes, and they also could not be within the reserved private IP address space. This could have been

resolved by modifying the source code of the daemon, but it was more convenient to resolve it in the network setup – by assigning addresses from an unused public IP space *26.0.0.0/8* – thus also avoiding unintended side effects in the network behavior of Bitcoin. Doing so will not disturb the internet because no route to the public internet was available inside the virtualized network. After the virtual network was set up, another Shell script was started on each virtual node to spawn their respective Cillocoin daemon, miners, and the network packet capture program (*tcpdump*) and to create and broadcast the double-spend transactions. The Python script was then responsible for merging / isolating the networks by the means described earlier. All command-line outputs and captured packets were redirected and stored within a data directory for collection and analysis by the researchers.

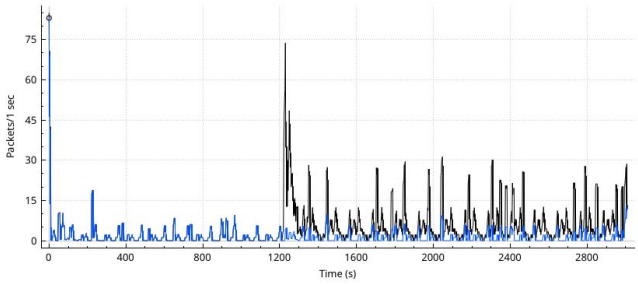
The size of the network used in our tests varies from ten to several hundred nodes, depending on the test. In the experiments for every type of attack, for consistency and convenience of comparison across different types of attacks, we always start with a small-scaled configuration of 10 nodes, after which we add modifications to explore the specific kind of attack according to what we find interesting in the initial small-scaled result. While the sizes of all these networks are much smaller than the real Bitcoin network, we feel it is reasonable for estimating the network behavior, as our results will indicate. We will examine the effects of scaling in our experiments and show that while the volume of traffic increases, the fundamental patterns in traffic changes caused by the attacks do not.

Note that in all of our experiments the simulated Bitcoin network is only running after the experiment begins, which means that all of the nodes will only start to connect to each other when the experiment is started, producing an initial burst of packets across the network, and these nodes are not started strictly simultaneously, especially for the larger-scaled ones, where the CPU cannot possibly handle all of their starting sequence and initialization code at exactly one particular time, causing the initial spike to spread out. In particular, any network traffic within the first several minutes (3-5 minutes) should be considered belonging to this initial burst (the attacks are not started until 20 minutes after the beginning). Such initial burst is visible in all of our plots, and we included the burst for completeness of the plots. This should not be confused with the data produced later on by the actual attack.

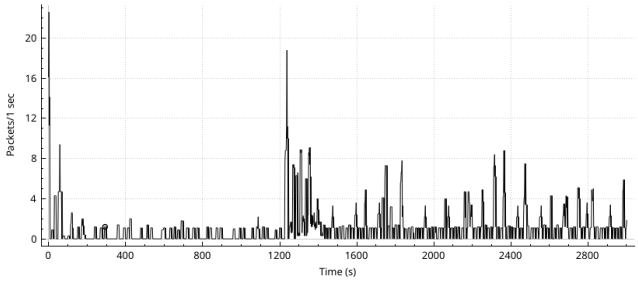
V. Attack Simulation and Analysis

A. Scenario I – Network split attack

We first tested a majority attack based on network partitioning that makes no out-of-spec requirements on the Bitcoin client software running on each node, assuming that the attacker could only control the network infrastructure of a subset of all nodes. No modifications other than those to ensure proper functionality in our testing environment (mentioned in previous sections) were made to the software used in this simulated attack. Consequently, the same type of attack could be conducted in the real Bitcoin network by any adversary



(a) Aggregated (overall)



(b) Single node perspective

Fig. 1. Packet frequency from the naïve attack

with control over part of the network infrastructure while the nodes are running the original Bitcoin client without altered source code. In addition, the unmodified client software reduces the number of compromised machines and the technical knowledge required to perform such an attack, thus possibly decreasing the difficulty of this attack.

1) Attack design: For this attack, we assumed a simple network topology, where there are two groups of nodes connected to two switches, with each switch controlling one group of nodes. Nodes in the same group can always reach each other, while the hypothetical adversary is capable of connecting or disconnecting the two switches, thus controlling the reachability between nodes in different groups. The hypothetical attacker first isolates the two groups, broadcasts an innocent-looking transaction to one of the groups (the *honest* network), then immediately broadcasts another transaction from the same Bitcoin address to the other group (the *attacker* network). The latter transaction is malicious in the sense that it transfers the same currency from the same address to an address different from the one seen in the honest network. The attacker then waits for some time until the attack chain grows beyond the honest chain in length (probably by an uneven division of the networks or by providing extra hashing power to the attacker network), and then connects the two subnets. After the two connect, every node in the attacker network will connect to the seed nodes in the honest network, thus joining the partitioned Bitcoin network.

2) Results and analysis: While simulating this attack, we chose a 5-5 node division (10 nodes in total) between the networks, giving extra hashing power to the attacker network.

All packets traveling through the honest network were captured and saved for further analysis. If we assume a defender with the ability to monitor the honest network (e.g. one of the major ISPs behind the honest network or a collaboration between honest nodes), a graph of packet frequency can be plotted as in Figure 1(a).

In Figure 1(a), the black line represents the number of packets at a given time with 10-seconds simple moving average (SMA), and the joining event happens at approximately $t = 1200s$. The packets between only the honest nodes are plotted in blue for comparison.

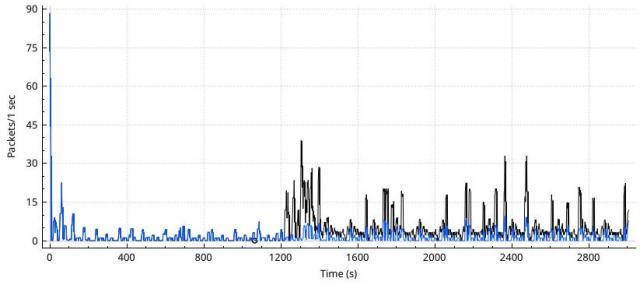
Figure 1(a) clearly shows that a burst in global packet traffic can be observed at the same time the attacker connects the malicious network with the honest one. Note that in a real 51% attack, there will probably be far more nodes in the attacker network than our current simulation; thus the burst in this figure should be an underestimation. This increase in packet traffic is caused by the intense exchange of block information resulting from the attack chain replacing the honest chain, which is the purpose of the attacker. We have also made an attempt to observe only from a single node perspective (Figure 1(b)) and noticed the same pattern described above. If such characteristics can be confirmed, the detection of such attacks is trivial, given that the defenders are capable of observing their own networks or possibly even only their own node.

Since all the attacker's nodes are connected to the seed nodes at once, this sudden burst of packets is also a result of the multitude of handshakes happening between the newly connected nodes. In a real-world case, it is difficult that such a large network partition would have remained left out of the rest of the network for so long without an attacker being behind the scenes. Therefore, while the burst could only be the result of a network merge, the possibility is negligible. An attacker would probably design a stealthier attack to work around this spike and not get caught so easily.

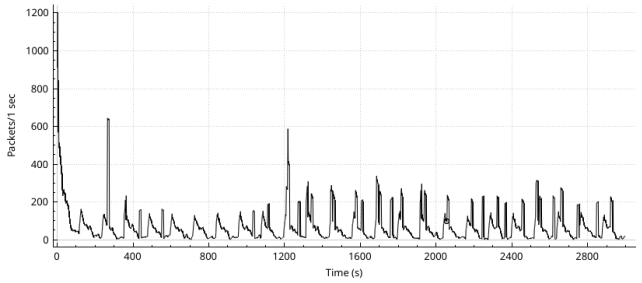
B. Scenario II – Single node network split attack

Since our simulated attack in Scenario I could introduce unwanted noise caused by isolated nodes connecting to the seed nodes at the joining event, a modification to the previous naïve attack was proposed to reduce the noise: the adversary can choose to only allow one node in the *attacker network* to connect to the seed node in the *honest network*. There will no longer be any burst of connections to the seed nodes (and to all other nodes pointed by them), and the networks will instead join slowly, thanks to the peer discovery feature of all the Bitcoin nodes, both in honest and attacker networks. The goal is to make the merging process more gradual and to reduce the spike observed in Scenario I. The downside of this approach is that it would slow down the propagation of the malicious chain in the honest network, increasing the resource cost for the attacker.

1) Attack design: Similarly to the previous scenario, we still divide the network into the *honest* and *attacker* networks. The hypothetical attacker would still isolate the two groups of nodes, broadcast the double-spend transactions and wait for the attacker chain to grow. When joining the networks, the attacker would also allow packets to pass between the networks, but this time let only one node in the attacker network connect to the seed node of the honest network. This node will then send the addresses of some honest nodes to the malicious subnet, leading to a slower discovery and connection of nodes, allowing the two networks to merge in a more gradual manner.



(a) Aggregated (10 nodes)



(b) Aggregated (100 nodes)

Fig. 2. Packet frequency from the improved attack

2) Results and analysis: We first conducted this improved attack with similar network sizes to the previous one – 5 nodes in each network with unbalanced computing power. Data was collected in the same way described in the previous attack, with the same confirmation that this pattern also applies when observing from the perspective of a single node. From Figure 2(a), although less obvious, it is still trivial to notice the spike in packet frequency at the time of the joining event $t = 1200s$. Therefore, it is at least more convincing that the burst of packets is not solely a result of sudden connections to the seed nodes.

Having collected these results, we scaled the attack up to a network of 100 nodes in total, questioning whether these patterns would persist in a more realistic simulation. Figure 2(b), plotted from the new data collected, shows that the height of the spike relative to normal levels was reduced, though still rather noticeable (3 times the normal level), and the raised level of packet frequency was still present after the joining event. This new result indicates that the ability to detect such attacks could be hindered to some extent in a real blockchain network, but the patterns are not hidden and can still be described similarly as in the smaller scale experiments.

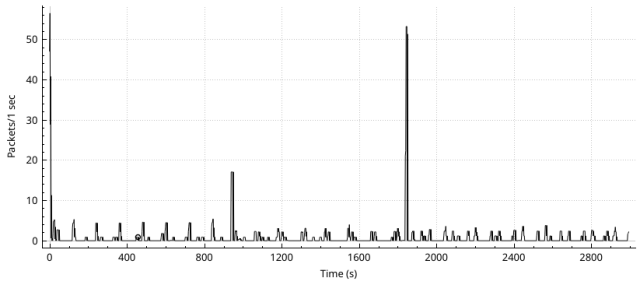
The result of this scaled-up experiment could be explained

by a specific behavior of the Bitcoin Core [27] client that limits the total number of outbound peer connections of a node to 8 by default. Since our network used distinct IP addresses that are not within the same /16 block, even they are still in a relatively small network, each node in either partition of the network would probably approach if not reach its maximum number of outgoing connections in the initial burst of peer discovery when the network was started. Therefore, the number of new connections each node can initiate after the joining event would be relatively low, and in the 100-node case, it is certain that the connections will be completely saturated, just as how real Bitcoin nodes behave. Most of the exchange of blocks caused by the attacker's chain replacing the original one now happens between the node we connected to the seed node and the seed node itself first and then propagates through the existing connections between the already connected nodes in both networks. The spike in the plot, in this case, is mostly caused by such data exchange packets, with little new connections in play. Since the spike is still noticeable when the noise introduced by new connections is excluded, this result supports our assumption that the patterns we extracted should be applicable to real Bitcoin networks.

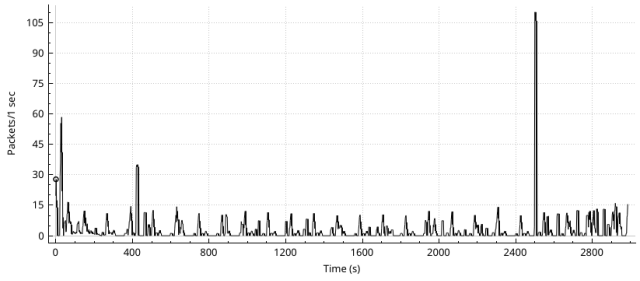
On the other hand, the presence of the sustaining increased level of packet frequency after the spike of the attack is still not addressed in our experiments. From our analysis, the attack should not directly influence the packet frequency once the malicious chain is distributed – the data exchange should stop once all nodes have switched to the attack chain, which should take far less than 30 minutes. We hypothesized that the sustaining effects after the attack could be the result of the increase in the total number of nodes present in the network: every node needs to perform periodic ping-pong and other network activities to indicate their existence. To test this hypothesis, a method of attack without requiring "joining" and without a sudden increase in node numbers in the entire network is required. After all, our attacks do not seem to rely on the number of nodes but on the division of the hashing power and on the propagation of the attack chain. It is not necessary that new nodes need to be added into the network for the attack chain to replace the original chain; as long as there exists a node that injects the longer attack chain into the honest network, the attack chain can propagate through the existing connections anyway.

C. Scenario III – Malicious nodes

Given the premises in the analysis of Scenario II, we altered the Cillocoin client to design a programmed attack to be performed by a number of clients running this tampered code. The source code was altered to temporarily ignore and to stop forwarding blocks and transactions as soon as the client daemon starts. A node running such code, once spawned in the network, would be almost indistinguishable from a normal one: it will continue to ping-pong with its neighbors, keep connections alive, and occasionally request and send data. The only apparent difference from an honest node will be in that



(a) Aggregated (without latency)



(b) With latency (single node perspective)

Fig. 3. Packet frequency from the attack with malicious nodes

whenever a new block is relayed to it, such a block will not be saved, and therefo,re the node will keep asking for the same blocks whenever nodes exchange messages that include information relative to blocks. The code was finally altered to allow the node to resume normal operations with a specific RPC command.

A pool of "silent" nodes (so-called because they do not forward new transactions and blocks) can mine new malicious blocks while still being connected to the rest of the network. At some point, when the malicious nodes release their chain to the public, a majority attack will occur. The following steps help conduct an attack that is difficult to detect:

1) Create a personal mining pool with enough power to mine blocks faster than the honest network;
2) This mining pool, assuming it is a subnetwork of nodes, must be either disconnected from the rest of the blockchain or be running the tampered software on every node;
3) In case the network is simply disconnected from the main network, an additional node running the malicious code mentioned above must be placed in between the attacker's mining pool and the rest of the blockchain. This node will function as a bridge between the two networks;
4) To merge blocks from the two networks, one or more nodes in the attacker's subnet must resume normal operations, forwarding newly mined blocks and the ones previously mined.

This protocol of attack ensures a reduced volume of packets being transferred between the honest nodes and the attacker's.

Because nodes are already connected to one another, there are no messages to do peer discovery, handshaking, and dataset comparisons.

1) Attack design: For this attack, we assumed that the attacker has no control over the network infrastructure but has control over a group of malicious "silent" nodes. The normal nodes run as usual, but the malicious nodes all only connect to one specific node (the *bridge* node) among them, which runs the altered software mentioned earlier. All malicious nodes, except the bridge node, had their peer discovery feature and incoming connections blocked so that they would not connect to any other nodes; the bridge node is connected to the normal network.

The first transaction is broadcast to the honest nodes, while the malicious transaction is sent to the nodes in the attacker's network. Since the only channel between them is the bridge node, which does not forward any transactions and blocks by default, they cannot be aware of each other, and no data from the malicious blockchain is visible to the rest of the network, at least until the RPC command to restore normal operation is issued. The attacker then mines new blocks on all the malicious nodes (with the exception of the bridge node) and waits until it grows longer than the honest chain. At this point finally restores the normal operation of the bridge node so that it will begin to forward the previously mined blocks and transactions from the malicious network.

2) Results and analysis: At first, we used five clients in the honest network and five clients as the malicious "silent" nodes for consistency with the previous experiments. As explained previously, the number of nodes should not change the fundamental patterns of attacks. From Figure 3(a), it is noticeable that the previous spikes at $t = 1200s$ have vanished, even if the RPC command (and therefore the attack) was still issued at the same time as the previous experiments. There are two reasons why the spike was not present at that point:

1) No new node joins the network suddenly; therefore there are no version messages, handshakes, and dataset comparisons.
2) Because of (1) and because nodes do not immediately compare their datasets, they are not aware of having different chains and so they do not start exchanging blocks until later in time.

The bridge node resumes normal operation once the RPC command is issued, which means it is able to forward *new* blocks to the normal network after this point in time but will not revisit old blocks automatically. Only when new blocks are then found by the malicious nodes, and when at least one of the normal nodes polls for new blocks, can they possibly be forwarded to the normal nodes. The normal nodes then start pulling all the blocks of the malicious blockchain, up to the point where the double-spend transactions were broadcast to the respective blockchains. This results in a delayed spike in packet frequency, which corresponds to what is shown in Figure 3(a).

This attack failed to hide the characteristic spike in packet frequency of such attacks but avoided the persisting increased average level of packet frequency after the majority attack, as we have hypothesized in the analysis of the previous attack. The packet frequency quickly dropped back to normal levels, as shown in Figure 3(a). Though it is possible that the quicker decrease back to normal levels can make it easier to miss by the defender, without the increased average level and noise after the attack, the spike gets even more noticeable than the previous ones. However, such a pattern could be fragile if the latency and bandwidth limits of a real network are taken into account. This could make the detection of the attack more difficult than any of the two attacks described previously, with only a short spike indicating the attack, which could be even further hidden by the latency and bandwidth limit in a real network.

To further explore how this attack would behave in a more realistic network with latency, we added 50ms of latency within both the honest and attack network, while an extra 50ms was added between the two networks. The network was also scaled up to 100 nodes with 50 nodes on each side to simulate an actual Bitcoin network better. The data is then used to produce Figure 3(b), which shows only data from an arbitrary single node in the honest network. It can be noticed that the new figure shows similar patterns as in Figure 3(a), with the exception of a slower propagation rate, more noise, and higher base packet frequency caused by more nodes being present in the network. Therefore, it can be concluded that the addition of latency did not disturb the characteristic spike resulting from the attack.

VI. Detection Algorithm

A. Real-time traffic analysis of a multitude of nodes

The experiments we performed based on the three increasingly sophisticated attacks have all shown that majority attacks do leave traces. More importantly, these traces are left immediately (in the two earlier scenarios) or a few moments (in the third scenario) after the attack is started. Based on this observation, we have designed a detection system that signals when a majority attack is potentially taking place by looking at the network traffic at the node level.

This system runs in parallel with the Cillocoin / Bitcoin client to constantly monitor network traffic volume. When abnormal traffic is detected, the node may suspect that a majority attack is underway. However, while majority attacks are proven to cause spikes in packet traffic, they are not the only cause of network spikes. A bottle-necked connection that suddenly becomes faster can cause a similar spike, as connecting to a much faster peer can. There are countless other reasons why internet traffic can spike at a particular node, so we introduce a second mechanism to verify the presence of a majority attack by confirming with a set of trusted nodes.

In our system, we utilize a set of trusted nodes that collaborate when suspected attacks are detected to confirm these attacks. Trusted nodes are nodes configured by the node owner and could be nodes run by major companies, financial institutions, or known people. To authenticate these nodes, each node has a list of public keys for trusted nodes that it uses

to verify signatures on incoming messages. When a potential attack is detected by a node, the node will confer with other nodes it trusts to determine if they have seen similar suspicious behavior.

A potential attack query is issued by a node when any of these two conditions is met:

1) The node detects a spike in its traffic of data that exceeds that node's attack threshold. (This threshold is explained in the next subsection.)
2) At least one trusted node signals a potential attack. This choice is explained in a further section.

Confirmed attacks are issued from a node when that node is almost certain that a majority attack is taking place. A confirmed attack is issued by a node when any of these two conditions are met:

1) More than half of the trusted nodes have signaled a potential attack within a short time frame.
2) At least one trusted node has signaled a confirmed attack.

B. Algorithm design

We have implemented the model described above using a Python script to be run concurrently with our Cillocoin software and on the same machine. By running a separate script, we make sure not to alter the working of the client, thus ensuring that it behaves as its original counterpart. The script must be configured to provide the path to the data directory of the running client, the interface to monitor traffic on, and the list of trusted nodes. The model is implemented with an algorithm that goes through three phases: deployment, initialization, and detection.

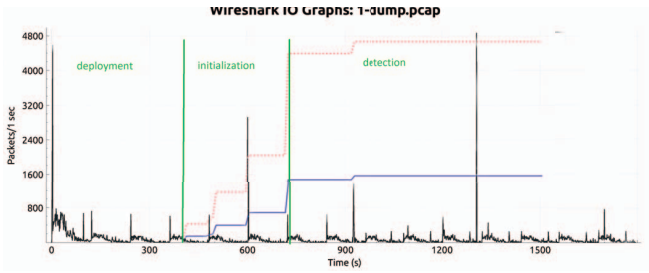
The deployment phase is roughly a long wait before starting to run. As shown in all experiments, newly started nodes go through an initialization phase in which they query the seed node, get a number of addresses, and start connecting to these, then discover more peers and repeat until the connection is saturated. This causes an initial burst in packet traffic that can greatly affect the accuracy of the algorithm in the detection phase. If this initial traffic was to be considered, the average spike height would be dramatically higher than it should be, setting the threshold to fire a potential attack so high that it would never fire. To avoid such inaccuracy, the algorithm sleeps for 5 minutes once started, which proved sufficient in all our tests.

The goal of the initialization phase is to set the threshold to fire signals. This threshold is directly proportional to the average height of the three highest spikes of traffic encountered. Traffic is monitored by calculating the number of packets that go though the given interface every second. The algorithm will count the packets sent and received in the last second and, if this value is higher than the lowest of the three highest spikes, that lowest will be discarded, and the new highest spike replaces it. We decided to allow the initialization phase to run for 5 minutes once the deployment phase is over. This proved to be enough to register traffic spikes resulting from newly mined blocks being distributed in the chain and ping-pongs happening concurrently. Once this phase is over, the three highest spikes so far will be indicative of how intense the traffic can become in normal behavior.

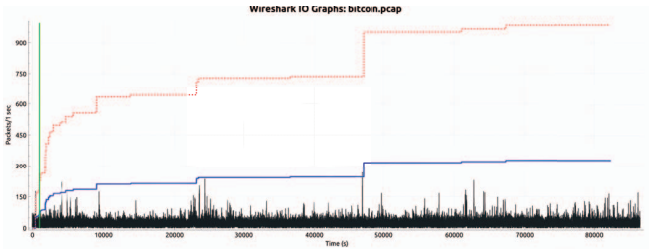
Once the initialization is over, the detection phase starts and goes on until the client stops running. In this phase, we continue to consider the packet traffic from second to second, which will be referred to as *current traffic* from now on. At each cycle, which lasts about a second, only one of the following three things can happen and no matter which one happens, the loop will continue with its next iteration once relevant actions are taken:

1) A confirmed attack is detected. This can be caused by a trusted node that has fired a confirmed attack or by at least half trusted nodes detecting potential attacks.
2) If the current traffic is greater than the threshold, a potential attack is suspected, and queries are sent to other trusted nodes.
3) If no attacks are detected or confirmed, we consider the current traffic level, and if it exceeds one of the previous three highest spikes, we update the threshold. This allows us to account for gradual network changes over time without triggering false alarms.

C. Deployment on Cillocoin



(a) Deployment on Cillocoin



(b) Deployment on Bitcoin

Fig. 4. Packet frequency and threshold value. Green lines indicate the boundaries of each phase, the blue line is the average of the last three spikes and the red dotted line is the corresponding threshold

To test the accuracy of the algorithm, we tested it in our experimental environment of client nodes running Cillocoin. The Python script runs alongside the Cillocoin client on each machine in the network.

The script was first deployed on nodes arranged according to scenarios I and II. Each node was given a set of five

trusted nodes to monitor. In both cases, the script detected the abnormal spike in network traffic immediately after the attack was launched. After querying the trusted nodes and seeing that they also detected or confirmed attacks, all nodes reported a confirmed attack. In particular, out of the 50 nodes running in the honest network: 12 detected first a potential attack because of the spike in traffic, queried their trusted nodes and confirmed the attack; the remaining 38 nodes, instead, directly fired a confirmed attack after detecting that at least one of their trusted nodes had confirmed the attack. All honest nodes detected the attack within 2 seconds after the attack was launched.

The third and most important experiment was running the algorithm on client nodes arranged according to scenario III. Each node was again given a set of trusted nodes. For each node, the algorithm first enters the deployment phase of five minutes, then the initialization phase of five more minutes. During the initialization phase, all nodes successfully start increasing their threshold values until they all have a sufficiently accurate one. The updating of the threshold value does not stop after the initialization phase terminated, thus keeping the value reliable.

Figure 4(a) represents the traffic of data of the first node in the honest network. It shows how the threshold value is updated as the algorithm runs. In the first 5 minutes, the threshold does not exist yet, then in the initialization phase, it keeps growing until it becomes approximately in line with the spikes. Once the initialization phase is over, it does not change significantly: after five minutes of updates, the threshold has converged to a consistent value. The attack happens at time *t=1200*; however, its footprint (the spike that it causes) does not show until about 100 seconds later. It is right at time *t=1307* that the traffic spikes to a point higher than the threshold, and the node fires a potential attack.

Once a node notifies trusted neighbors of a potential attack, many other nodes do too because of the same spike they detect, quickly confirming the attack. Table I shows how many nodes have fired a potential attack, how many have confirmed it by looking at potentially attacked trusted nodes (case I), and how many have done so because at least a trusted node confirmed it (case II). It also provides the statistics of the average time at which nodes detected or confirmed the attack. It must be noted that since a node will fire a confirmed attack upon seeing a trusted node firing it, not all nodes have fired potential attacks. However, all nodes have confirmed it in some way or another (the sum of the two typed of confirmations sum to 50, the number of nodes in the honest network).

	Potential attack	**Confirmed attack (case I)**	**Confirmed attack (case II)**
Number of nodes	27	16	34
Avg. signal time (in ms)	1307	1307	1308

TABLE I
OVERVIEW OF THE ALGORITHM RESULTS

D. Deployment on Bitcoin

The final experiment that we deemed necessary was testing the detection algorithm on a real Bitcoin client connected to the Bitcoin network and actively exchanging data with other nodes. To do so, we set up a full node following the official documentation [35], waited for it to complete the Initial Block Download phase [36], then started our detection code. The program was slightly edited to adapt to the slower pace of the original Bitcoin (we artificially decreased the block discovery time in the Cillocoin clients) by increasing the length of the initialization phase from 5 minutes to 20, or twice the average block discovery time of 10 minutes, and eliminating the initial sleeping phase. This is unnecessary because the Bitcoin client had been already running for a reasonable amount of time. Figure 4(b) shows that the algorithm updated its threshold value to match the height of the frequent spikes, and no spike ever overcomes the threshold. While observing a majority attack in the wild with our system is impossible without being incredibly lucky, and Bitcoin is too large for a majority attack to be achieved easily, this test demonstrates that our threshold calculation is compatible with the live Bitcoin blockchain and will not fire spurious false alarms.

VII. DISCUSSION AND MITIGATION

The three experiments were run in a controlled environment design to mimic a real-world scenario as closely as possible. It was impossible to simulate it perfectly though: networks in the real world can go down unexpectedly, have sudden long delays, and lose substantial portions of the transmitted packets, variables that we did not account for. It can be argued, however, that such abnormal cases impact only a minority of the nodes in the real Bitcoin network and can, therefore, be ignored. The other factor that can affect the accuracy of the measurements is noise, again not present in our controlled environment but potentially relevant in a node deployed on the internet. Since our nodes were connected to one another, they had no actual access to the internet, and, therefore, the noise was kept to the minimum.

The choice of firing potential attacks upon seeing a spike more than three times higher than the greatest three spikes so far was experimentally derived. We noticed that some random spikes were occasionally occurring, and these were always no higher than twice the average of the previous. However, spikes caused by majority attacks were always at least 3.5 to 4 times greater. Thus we decided to pick 3 as the multiplier of the average to define the threshold, and the choice proved effective. It is possible that it may need to be adjusted differently on each node the algorithm is run on, but for our controlled tests, this was accurate enough.

This detection scheme works well on the basis that each node picks reliable, trusted nodes. Choosing potentially malicious nodes as trusted nodes can lead to either over or under detection, both cases rendering the algorithm useless. One way to prevent this is for node owners to choose trusted nodes based on economic reasons, e.g. nodes that stand to lose money if a majority attack succeeds. It is not necessary that all trusted

nodes are reliable, though. An attacker could try to undermine the efficiency of the software by taking control of some trusted nodes and blocking them from firing attacks. For an attack to pass undetected without a single node firing it though, the attacker would need to control the majority of the trusted nodes. As highly trusted entities will likely choose other highly trusted entities as trusted nodes, for example, crypto exchanges could select other crypto exchanges, the effort needed to subvert significant numbers of these trusted nodes would potentially be greater the reward from a successful majority attack.

Another possible weakness is during prolonged monitoring of a node. Our test cases lasted from half an hour to a maximum of a day, which is a small portion of the up-time of full nodes, which often run 24/7 all year long. Because network traffic can increase and decrease, we may end up having exceptionally high registered spikes and a resulting threshold that is too high to detect attacks. Because our tests never required these values to be somehow managed in the long term and testing any solution would have required an exceptionally long time, we decided to leave them as-is. A future implementation must take this into account, restarting the algorithm every once in a while or using the three highest spikes during a sliding window of time.

VIII. Conclusion and future work

This study proved that it is practically possible to detect majority attacks in a controlled environment. Although the algorithm performed well in our simulated scenarios, in a real-world deployment, it would need further adjustments to take into account the unpredictable nature of a real network. The results nonetheless are promising and open the way to future works on algorithm designs more tailored to implementation on real crypto currency networks.

The goal of this research was to study the effects of majority attacks on a blockchain and develop a detection algorithm. What to do once the attack is discovered falls outside the scope of this paper. We can, however, provide our educated opinion on what paths can be taken in developing a defense system against majority attacks. Given the short time elapsed from the launch of the attack to its detection, the number of transactions affected by the attack is quite limited. It could therefore be feasible to analyze all affected transactions and identify those that conflict with one another. What action to take once the conflicting transactions are identified is not ours to discuss. We hope to see future research take this basic detection scheme and evolve it into a fully-fledged system to detect and take action against majority attacks.

References

[1] "Proof of work," Nov 2018, https://en.bitcoin.it/wiki/Proof_of_work.

[2] S. Dexter, "Longest chain – how are blockchain forks resolved?" June 2018, https://www.mangoresearch.co/blockchain-forks-explained/.

[3] G. O. Karame, E. Androulaki, and S. Capkun, "Double-spending fast payments in bitcoin," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 906–917, http://doi.acm.org/10.1145/2382196.2382292.

[4] J. Waldo, "A hitchhiker's guide to the blockchain universe," *Queue*, vol. 16, no. 6, pp. 10:21–10:35, Dec. 2018. [Online]. Available: http://doi.acm.org/10.1145/3305263.3305265

[5] M. Rosenfeld, "Analysis of hashrate-based double spending," *arXiv preprint arXiv:1402.2009*, 2014.

[6] J. J. Roberts, "Bitcoin spinoff hacked in rare '51% attack'," May 2018, http://fortune.com/2018/05/29/bitcoin-gold-hack/.

[7] T. Delahunty, "Verge cryptocurrency hit by 51% attack, loses 250,000 tokens," Apr 2018, https://www.newsbtc.com/2018/04/05/cryptocurrency-verge-has-been-hit-with-a-51-attack-loses-250000-tokens/.

[8] P. H. Madore, "Ethereum classic might have been hit by a 51% attack," Jan 2019, https://www.ccn.com/ethereum-classic-might-have-been-hit-by-a-51-attack.

[9] B. N. Levine, C. Shields, and N. B. Margolin, "A survey of solutions to the sybil attack," *University of Massachusetts Amherst, Amherst, MA*, vol. 7, p. 224, 2006.

[10] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 375–392.

[11] Y. Marcus, E. Heilman, and S. Goldberg, "Low-resource eclipse attacks on ethereum's peer-to-peer network." *IACR Cryptology ePrint Archive*, vol. 2018, p. 236, 2018.

[12] M. Vasek, M. Thornton, and T. Moore, "Empirical analysis of denial-of-service attacks in the bitcoin ecosystem," in *International conference on financial cryptography and data security*. Springer, 2014, pp. 57–71.

[13] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," *Commun. ACM*, vol. 61, no. 7, pp. 95–102, Jun. 2018, http://doi.acm.org/10.1145/3212998.

[14] N. T. Courtois and L. Bahack, "On subversive miner strategies and block withholding attack in bitcoin digital currency," 2014.

[15] S. Dey, "Securing majority-attack in blockchain using machine learning and algorithmic game theory: A proof of work," 2018.

[16] P. Robinson, "Design patterns which facilitate message digest collision attacks on blockchains," *CoRR*, vol. abs/1806.05822, 2018, http://arxiv.org/abs/1806.05822.

[17] S. Eskandari, S. Moosavi, and J. Clark, "Sok: Transparent dishonesty: front-running attacks on blockchain," *arXiv preprint arXiv:1902.05164*, 2019.

[18] W. Kenton, "Front-running," Mar 2018, https://www.investopedia.com/terms/f/frontrunning.asp.

[19] D. Azaraf, "Front running and its effect on decentralized exchanges," Jul 2018, https://medium.com/totle/front-running-and-its-effect-on-decentralized-exchanges-e463ca4474db.

[20] "Bitcoin charts & graphs - blockchain," https://www.blockchain.com/en/charts.

[21] O. Ofcorti, "Neighbourhood pool watch," Nov 2016, https://organofcorti.blogspot.com/.

[22] K. Cieśla, "Bitcoin network hashrate," https://data.bitcoinity.org/bitcoin/hashrate/6m?c=m&g=15&t=a.

[23] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE P2P 2013 Proceedings*. IEEE, 2013, pp. 1–10.

[24] J. Zhu, P. Liu, and L. He, "Mining information on bitcoin network data," in *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, June 2017, pp. 999–1003.

[25] P. L. Juhász, J. Stéger, D. Kondor, and G. Vattay, "A bayesian approach to identify bitcoin users," *PloS one*, vol. 13, no. 12, p. e0207000, 2018.

[26] J. Wilmoth, "What is an altcoin?" 2014, https://www.ccn.com/altcoin/.

[27] Bitcoin, "Bitcoin core v0.16.0," Feb 2018, https://github.com/bitcoin/bitcoin/releases/tag/v0.16.0.

[28] P. Huang, *A dissection of bitcoin*. Lulu Press, Inc, 2016.

[29] K. Okupski, "Bitcoin developer reference," *Eindhoven*, 2014.

[30] F. Board, M. Chen, and A. Badev, *Bitcoin: Technical Background and Data Analysis*. Lulu.com, 2015, https://books.google.com.hk/books?id=-vo1jwEACAAJ.

[31] "How to compile bitcoin source code in ubuntu 16.04 lts," Aug 2018, https://www.toshblocks.com/bitcoin/compile-bitcoin-source-code-ubuntu-16-04-lts/.

[32] fisheater, "Complete guide on how to create a new alt coin," June 2013, https://bitcointalk.org/index.php?topic=225690.0.

[33] "Confirmation," March 2018, https://en.bitcoin.it/wiki/Confirmation.

[34] MiniNet, "Mininet project," http://mininet.org/.

[35] "Running a full node," May 2019, https://bitcoin.org/en/full-node.

[36] “Initial block download,” 2019, https://bitcoin.org/en/p2p-network-guide#initial-block-download.