TendrilStaller: Block Delay Attack in Bitcoin

Matthew Walck, Ke Wang, Hyong S.Kim
Carnegie Mellon University
mwalck@andrew.cmu.edu, kewang1@andrew.cmu.edu, kim@ece.cmu.edu

Abstract*—We present TendrilStaller, an eclipse attack targeting at Bitcoin's peer-to-peer network. TendrilStaller enables an adversary to delay block propagation to a victim for *10* minutes. The adversary thus impedes the victim from getting the latest blockchain state. It only takes as few as one Bitcoin full node and two light weight nodes to perform the attack. The light weight nodes perform a subset of the functions of a full Bitcoin node. The attack exploits a recent block propagation protocol introduced in April 2016. The protocol prescribes a Bitcoin node to select *3* neighbors that can send new blocks unsolicited. These neighbors are selected based on their recent performance in providing blocks quickly. The adversary induces the victim to select *3* attack nodes by having attack nodes send valid blocks to the victim more quickly than other neighbors. For this purpose, the adversary deploys a handful of light weight nodes so that the adversary itself receives new blocks faster. The adversary then performs the attack to delay blocks propagated to the victim. We implement the attack on top of current default Bitcoin protocol We deploy the attack nodes in multiple locations around the globe and randomly select victim nodes. Depending on the round-trip time between the adversary and the victim, *50%-85%* of the blocks could be delayed to the victim. We further show that the adoption of light weight nodes greatly increases the attack probability by *15%* in average. Finally, we propose several countermeasures to mitigate this eclipse attack.

Keywords—Bitcoin, eclipse attack, block propagation, peer-to-peer network, security analysis, countermeasure

I. Introduction

Bitcoin is the most widely adopted cryptocurrency with more than *17* million Bitcoins in circulation and *76* billion USD market capitalization as of March 2019. Bitcoin's underlying technology, the blockchain, is considered for applications in smart contract [1], health care [2], supply chain [3], review system [4] and data storage system [5].

Earlier works have analyzed the security and reliability of Bitcoin. Nakamoto [14] shows that Bitcoin is fundamentally broken if a single party possesses more than *51%* of the total mining power. Selfish mining [6] enables attackers to command only *33%* of the total mining power to gain significant advantage. Double-spending attack [7, 8], block withholding attack [9, 10] and denial of service attack [11] all exploit Bitcoin's weaknesses for profit. Most of these studies assume blocks in the Bitcoin network are almost immediately available to the majority of the miners.

Network-plane attacks [12,19,21] target Bitcoin's peer-to-peer network. Victims of network-plane attacks may not receive the latest blockchain state in a timely manner, inducing them to waste their mining power on stale blocks. Consequently, the adversary in [6-11] could gain even more of an advantage [12,22]. Heilman et al propose an eclipse attack on Bitcoin's peer to peer network [19]. The attack allows the adversary to isolate the victim node from the rest of the network. An adversary needs to control a large number of IP addresses and operate a large number of attack nodes. The attack nodes continuously initiate connections to the victim, and send crafted *addr* messages to the victim. The victim's table of future neighbors would be populated with IP addresses controlled by the adversary. After the victim's restart, all of its neighbors will be controlled by the adversary, with high probability. The adversary then tampers with the messages sent to the victim to manipulate its view of the blockchain.

Apostolaki et al propose a BGP hijacking attack [21] that similarly causes a partition of the Bitcoin network leading to the isolation of a certain victim. This attack leverages BGP hijacking to intercept the connections of the victim nodes. An Autonomous-System (AS) level adversary sends rogue route advertisement. Traffic destined for, or originated from, the victims is sent to the wrong locations. The adversary isolates the victim nodes by dropping the traffic. The adversary could also delay information sent to the victim by manipulating a small number of Bitcoin messages.

The above two attacks are suitable for AS-level adversaries with sufficient IP resources or network bandwidth. The cost of a successful attack could be expensive. The attack in [19] requires the adversary to operate large number of Bitcoin nodes with distinct public IP address. The attack in [21] requires the adversary to receive and then relay all of the internet traffic (not necessarily Bitcoin related) of the entire hijacked AS (not only the victim node). Unlike [19, 21], our attack only requires as few as *3* bitcoin nodes to perform the attack. These attack nodes do not need to have distinct public IP addresses. The attack could be performed from nodes behind a firewall.

Gervais et al [12] propose a block delay attack. An adversary can delay the propagation of blocks to specific nodes for a considerable amount of time without causing a network partition. The attack exploits a historical block propagation protocol. Bitcoin nodes broadcast the advertisements of new blocks. When a node receives multiple advertisements of the same block, it only requests the block from the first neighbor

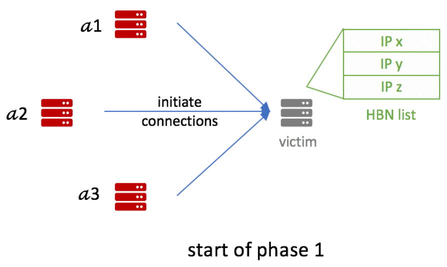


Figure 2.a

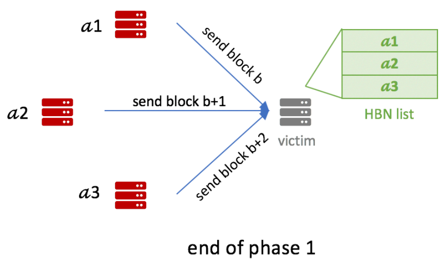


Figure 2.b

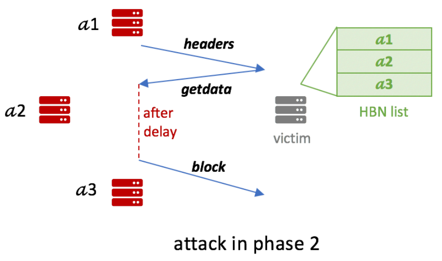


Figure 2.c

Figure 2 Two phases of the attack

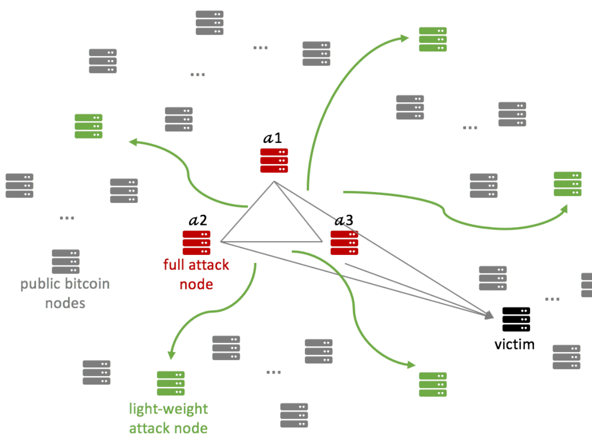


Figure 1 TendrilStaller setup

that sends the advertisement. Attack nodes in [12] thus try to be the earliest one to send the advertisement of new blocks to the victim. Once the victim requests the block from an attack node, the victim temporarily abstains from requesting the same block from other neighbors. The victim will not get the new block until the attack node actually sends the block. The timeout for a *getdata* request was *20* minutes. The adversary thus could delay the delivery of new blocks to the victim by *20* minutes. This attack is effective for victim nodes running Bitcoin software with versions up to *v0.10*. Bitcoin has evolved over the years. A new block propagation protocol has been adopted which makes the attack in [12] obsolete.

The new block propagation protocol allows a node $\mathcal{V}$ to select up to *3 High-Bandwidth (HB)* mode neighbors [13]. The list of HB neighbors are referred to as the *HB-Neighbor (HBN)* list. HB neighbors do not send advertisements of new blocks; instead , they send a new, unsolicited block to $\mathcal{V}$ as soon as they receive it. This change mitigates the original attack. Even if the attack node $\mathcal{A}$ manages to be the first to send the advertisement to $\mathcal{V}$, and then delays the response to $\mathcal{V}$'s *getdata* request, one of the $\mathcal{V}$'s non-attack HB neighbors could later send the block to $\mathcal{V}$, unsolicited. $\mathcal{V}$ still receives the new block without significant delay despite $\mathcal{A}$'s delay.

Figure 1 shows the setup of TendrilStaller. We define *attack nodes* as nodes controlled by the adversary, and *non-attack nodes* as all other nodes in the public Bitcoin network. The adversary deploys two type of attack nodes, i.e., *full attack nodes* and *light weight attack nodes*. The full attack nodes establish connections to the victim and perform the actual attack. The light weight attack nodes operate in either of the two modes. In Mode 1, light weight attack nodes do not connect to the victim. They serve as a proxy for the full attack nodes to receive new blocks faster. In Mode 2, light weight attack nodes also participate in the attack. Figure 1 and Figure 2 describe the attack with light weight attack nodes operating in Mode 1. There are *3* full attack nodes, referred to as $a1$, $a2$ and $a3$. The attack consists of two phases. Each full attack node initiates a connection to the victim node $\mathcal{V}$. In Phase 1, the adversary induces the victim to select all *3* full attack nodes as its HB neighbors. Node $\mathcal{V}$ selects its HB neighbors according to the *most-recent-sent-block* rule. If a neighbor sends a new valid block to $\mathcal{V}$, $\mathcal{V}$ adds that neighbor to the HBN list. If there are already *3* existing HBN neighbors, the oldest one is removed from the HB list. $a1$, $a2$ and $a3$ keep sending new, valid blocks to the victim, until victim's HBN list consists of *3* full attack nodes, as shown in Figure 2.a and Figure 2.b. In Phase 2, the full attack nodes behave similarly to the attack in [12]. Since no non-attack neighbor sends unsolicited blocks to $\mathcal{V}$ in Phase 2, full attack nodes delay the block propagation to the victim.

We make the following contributions in the paper.

- We present TendrilStaller, a block delay attack that could delay the block propagation to a victim for considerable time. We implement the attack code as published in Github [17].
- We validate the attack algorithm in the public Bitcoin network. We demonstrate that as many as *50-85%* of blocks could be delayed to the victim during the attack. We further show that the adoption of light-weight nodes increases the attack probability by *15%* on average.
- We propose several countermeasures to mitigate TendrilStaller. We show that these countermeasures could make the block delay attack infeasible.

The rest of the paper is organized as follows. In Section II, we briefly present the background of Bitcoin and its blockchain. We also describe the old block propagation protocol and the existing block delay attack in [12]. In Section III, we describe the new protocol adopted by Bitcoin and its impact on the attack in [12]. We formally present our attack algorithm and describe various optimizations for the attack in Section IV. We validate our findings through experiments in Section V. We discuss the impact of the attack and propose a few countermeasures in Section VI. Finally, conclusion is drawn in Section VII.

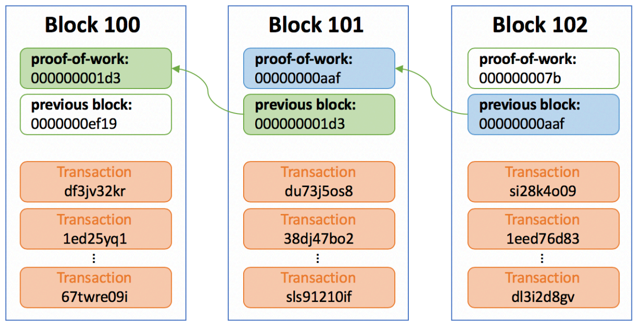


Figure 3 Blockchain

II. Historical Mechanism of Block Propagation and Block Delay Attack

A. *Bitcoin*

Bitcoin implements a blockchain protocol to record and serialize transactions among users. Figure 3 shows how each block is related to the previous one, with each block containing the hash of the prior one. Computing nodes, or miners, compete with each other to be the first to build a new block by finding a satisfactory nonce. Miners are interconnected by a peer-to-peer network, through which the newly built blocks are broadcast. Miners then verify blocks independently and start working on the next block. We use miners and nodes interchangeably in the rest of the paper. Each miner keeps a local copy of the blockchain, referred to as the view perceived by the miner. Miners may have different views of the blockchain due to the block propagation delay. Different views of the blockchains contain different sets of transactions in different orders. If a miner does not receive the latest block in a timely manner, its view will differ from those of the majority of miners. This miner would waste its mining power, as even if it succeeds in building a new block, with high probability, that block will not be accepted by other miners and no reward will be given for the block.

B. *Bitcoin peer-to-peer network*

Bitcoin nodes connect to each other over TCP/IP. Each node maintains a *tried* and *new* table to store public IPs. The *tried* table stores the IP addresses of neighbors to whom the node has recently established a connection. The *new* table stores the addresses of nodes to whom a future connection may be established. The *new* table is initially populated with information from a DNS server when the node first joins the network. Nodes also exchange IP addresses of known nodes via *addr* messages to update their *new* table periodically. The node randomly selects from the two tables when it needs to setup a new connection. In the default configuration [15], a Bitcoin node picks at most *8* nodes from the two tables randomly to establish connections, referred to as *outgoing connections*. A node also accepts connections initiated by other nodes, referred to as incoming connections. One node can have at most *125* connections in total.

C. *Mining pools*

Being the first to mine a block occurs only with a small probability. The payoff for individual miners are bursty. Miners therefore form mining pools. Members of a pool do not establish

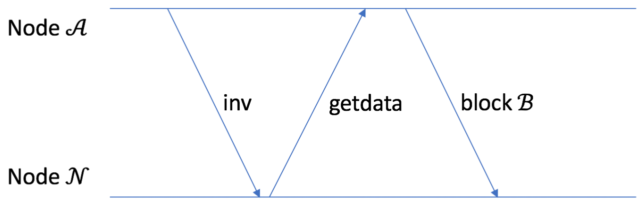


Figure 4. Advertisement based propagation

connections with nodes from outside the pool. Instead, they only receive information from the pool manager. A mining pool deploys a number of pool gateway nodes to connect the mining pool with the public Bitcoin network. Pool gateway nodes are known to have many more connections than a default node [23], as they must ensure low-latency relay of blocks and transactions for the mining pool.

D. *Historical block relay model*

Bitcoin protocol implements various methods to minimize the bandwidth consumption. First, Bitcoin adopts an advertisement based block relay mechanism as shown in Figure 4. When a node $\mathcal{A}$ learns about the new block, either by mining a new block itself or receiving one from its neighbor, it broadcasts an *inv* message to its other neighbors to announce the existence of the new block. For neighbor $\mathcal{N}$, if it does not already have that block, it sends a *getdata* message to $\mathcal{A}$ to request the block. Node $\mathcal{A}$ then responds with the actual block to $\mathcal{N}$. Compared to directly sending a new block upon receiving it, the advertising-first approach reduces redundant block transmissions of blocks the neighbors already have. A block can be as large as *1MB*, whereas an *inv* message only contains the hash of the block of *40* bytes. Second, a node only responds to the first advertisement it receives. It temporarily stops requesting the same block from other neighbors until the previous *getdata* request times out. This saves the outbound bandwidth of the neighbors. Finally, Bitcoin adopts a timeout of *20* minutes for *getdata* request (now reduced to *10* minutes). Bitcoin nodes are provisioned with different network bandwidth [16]. Nodes with limited outbound bandwidth takes longer time to transmit a block. A longer timeout avoids unnecessary block retransmissions in case of slow network or network congestion.

E. *Original block delay attack*

The block delay attack in [12] is carried out as follows. An attacker $\mathcal{A}$ first manages to be the first node to send the advertisement of a block $\mathcal{B}$ (inv message) to the victim $\mathcal{V}$. The attack node implements two methods in order to be the first to send the advertisement.

- First, an attack node establishes more connections than a default Bitcoin node. More connections enable the attack node to learn about a new block faster when it is mined.
- Second, the attack node advertises the new block immediately after receiving it, without verifying its correctness.

Default Bitcoin nodes fully validate the new block before announcing it to the neighbors. The validation prevents malformed block propagation in the network. The validation process involves a series of steps and takes considerable time. It

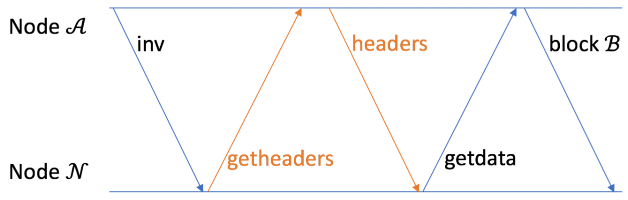


Figure 5.1 modified *inv* message flow

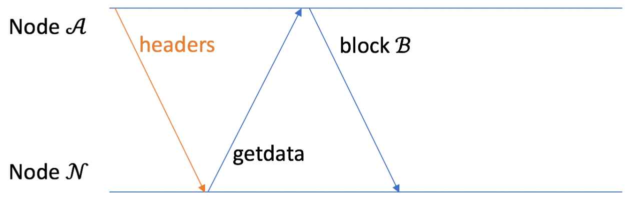


Figure 5.2 direct-headers announcement

checks whether the block contains a correct proof of work, whether the mining difficulty of this block is set correctly and so on. For each transaction in the block, the validation also checks whether the transaction format is correct and whether all the inputs of the transactions are valid, unspent transaction outputs (UTxO) [20]. It is reported in [12] that the validation takes on average *174* milliseconds on an Amazon EC2 dual-core instance. Bypassing the validation process gives the attacker *174* milliseconds advantage on average.

If the attack node succeeds in being the first to send the *inv* message, the victim 𝒱 then sends a *getdata* request to 𝒜, and temporarily holds back from requesting ℬ from other neighbors. Attacker 𝒜 waits sufficiently long after receiving the *getdata* message from the victim. The timeout for a *getdata* request used to be *20* minutes. An attacker can delay the block delivery to the victim for at least *20* minutes. In average, a block is mined every *10* minutes. By the time the victim receives the block, it will be *2* blocks behind the longest chain.

III. New Model of Block Propagation

Bitcoin protocol has evolved over the years. New block propagation protocols have been adopted to make the network more scalable. Four new measures are of interest to us.

A. *New measure 1: direct-headers announcement*

Since version *v0.10.0* in 2015, a node only requests a block if it has received a valid header of that block. The message flow to relay a new block in Figure 3 changes to that in Figure 5.1. Upon receiving an *inv* message, node 𝒩 first sends a *getheaders* message to request the header of that block. After 𝒩 receives and validates the *headers* message, it sends the *getdata* request. A valid *headers* message should contain the block header that has the correct proof-of-work and is built on top of the current blockchain. The previous *1.5* round trip time (RTT) increases to *2.5* RTT. To reduce the propagation delay, direct-headers announcement is proposed and added to Bitcoin protocol since version *v0.12.0*. As shown in Figure 5.2, node 𝒜 directly sends a *headers* message when it learns about a new block. Round trip time is reduced to *1.5* RTT as before.

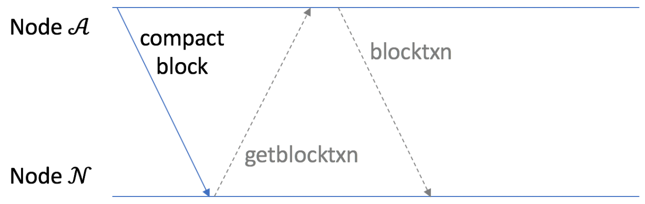


Figure 6.1 HB neighbor ***𝒜***

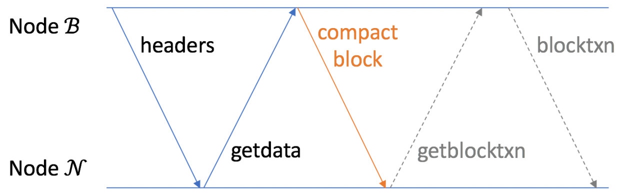


Figure 6.2 Non-HB neighbor ***ℬ***

B. *New measure 2: compact block relay*

Block relay causes significant outbound bandwidth spikes for nodes which receive a block before their neighbors [13]. A node receiving block earlier than its neighbors needs to send the new block multiple times, one to each neighbor. Such bandwidth spikes delay the transmission of blocks. Combining with the fact that a node only requests the same block from one neighbor at a time, the block propagation time would increase.

Compact block relay is proposed in [13] to save bandwidth. A full block as large as *1* MB contains a small header of *80* bytes and a list of transactions. Most of these transactions are already available to other nodes before the block propagation. Instead of sending the full block, a node could send a compact version of the block. A compact block consists of a block header, a list of transaction IDs and a small set of prefilled transactions.

- *Block header* contains *80* bytes, the same as in a regular block.
- *Transaction ID* is the hash of a transaction. When receiving a compact block, the receiver is expected to look up the actual transaction from receiver's own in-memory transaction table.
- *Prefilled transactions* are regular transactions. When constructing a compact block, if the sender believes certain transactions have not been seen by the receiver, the sender will put the actual transactions into the compact block instead of the transaction hash.

The intention of compact block relay is to send minimum data while providing enough information for the receiver to reconstruct the block. If a certain transaction ID is unknown to the receiver and that transaction is not in the set of prefilled transactions, the receiver subsequently requests that specific transaction. In the optimal case it only takes *0.5* RTT to relay a block. In the worst case when there are missing transactions in the compact block, one extra roundtrip is required. It takes *1.5* RTT in total, the same as direct-headers announcement, but with much less bandwidth consumption. Compact blocks are sent either in response to *getdata* requests or unsolicitedly as in HB mode (next section).

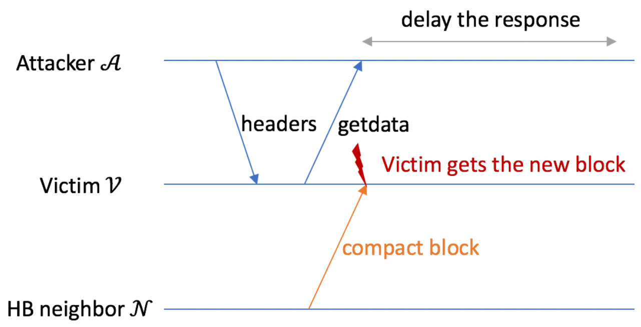


Figure 7. Original attack fails

C. *New measure 3: High bandwidth (HB) mode neighbor*

A node selects *3* neighbors as High Bandwidth (HB) neighbors. We refer to the list of HB neighbors as HBN list. A node $\mathcal{N}$ instructs its HB neighbors to send compact blocks unsolicitedly. We refer to such blocks as unsolicited compact blocks. When a neighbor is added to the HBN list, $\mathcal{N}$ sends a *sendcmpct(true)* message to that neighbor. Similarly, if a neighbor is removed from the HBN list, $\mathcal{N}$ sends a *sendcmpct(false)* message to the evicted neighbor to instruct it to stop sending unsolicited compact blocks. Limiting HBN list to only *3* neighbors reduces the bandwidth consumption. Selecting *m* HB neighbors results in transmitting *m-1* redundant compact blocks, as all the HB neighbors send a new compact block when receiving it.

As shown in Figure 6.1, the HB neighbor $\mathcal{A}$ relays new blocks to $\mathcal{N}$ by sending compact blocks directly. If necessary, node $\mathcal{N}$ further sends a *getblocktxn* message to request missing transactions. $\mathcal{A}$ then responds with *blocktxn* messages. Node $\mathcal{N}$ validates the block and then updates $\mathcal{A}$'s *last-block-sent* time. The last-block-sent timestamp records the last time when $\mathcal{A}$ sends a new valid block to node $\mathcal{N}$. This timestamp is used to determine which of the HB neighbors will be evicted in the case that a new HB neighbor is added to the list.

As shown in Figure 6.2, when a non-HB neighbor $\mathcal{B}$ learns about a new block, the following sequence of events happen.

1. $\mathcal{B}$ sends the *headers* message to advertise the new block.
2. If node $\mathcal{N}$ has not heard about this header before, $\mathcal{N}$ subsequently sends a *getdata* request requesting the compact block.
3. $\mathcal{B}$ responds with the compact block.
4. *getblocktxn/blocktxn* messages are exchanged if node $\mathcal{N}$ misses certain transactions in the compact block.
5. $\mathcal{N}$ updates its HBN list to include node $\mathcal{B}$. A previous HB neighbor is evicted from the list according to its *last-block-sent* timestamp.

In addition, an HB neighbor $\mathcal{A}$ only sends a compact block to its neighbor $\mathcal{N}$ if the following additional requirements are met.

- R_1: Node $\mathcal{A}$ believes $\mathcal{N}$ does not have the current block. For example, $\mathcal{N}$ must not have advertised this block to $\mathcal{A}$. This is to avoid sending unnecessary compact blocks.
- R_2: Node $\mathcal{A}$ believes that the previous block of the current block is known to node $\mathcal{N}$. For example, node $\mathcal{N}$ has announced the previous block to node $\mathcal{A}$, or node $\mathcal{A}$ has also sent the previous block to node $\mathcal{N}$ before. This is to make sure node $\mathcal{N}$ is able to validate and reconstruct the actual block when receiving the compact block.

D. *New measure 4: shorter block download timeout*

Bitcoin protocol has shortened the block download timeout since version *v0.10.5*. The base timeout value is *10* minutes. The timeout increases *5* minutes for each additional *getdata* request in parallel. This is to prevent timing out peers due to the node's own downstream link being saturated.

E. *Impact on original attack*

The block delay attack described in [12] is not possible with the new block propagation protocol. The attack node does not know whether the victim has picked its HB neighbors. In the common case, where there are HB neighbors that send compact blocks unsolicited to the victim $\mathcal{V}$, the attack fails. As in Figure 7, attack node $\mathcal{A}$ manages to be the first to send a *headers* message and then delays the response to victim's *getdata* request. An HB neighbor $\mathcal{N}$ later sends an unsolicited compact block to victim $\mathcal{V}$. The victim still receives the new block without a significant delay. The attack only succeeds in the rare case where no neighbors are able to send compact blocks directly, i.e., none of victim $\mathcal{V}$'s HB neighbors satisfies requirements R1 and R2 at the same time. This only happens when $\mathcal{V}$ has recently joined the network.

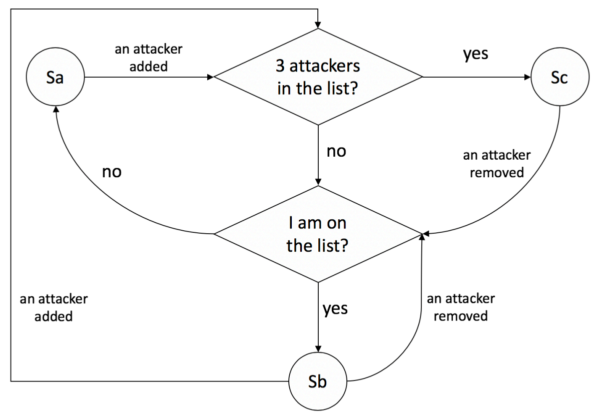
The attack in [12] was also able to delay the block for extended time, exceeding the timeout of the *getdata* request. It is impossible in the new Bitcoin block propagation protocol, neither. Once victim $\mathcal{V}$ sends a *getdata* request to the attack node, while $\mathcal{V}$ does not respond to subsequent *headers* messages, it records who has sent these *headers* messages. If attack node $\mathcal{A}$ delays the block by more than *10* minutes, victim $\mathcal{V}$ disconnects from $\mathcal{A}$. $\mathcal{V}$ then picks a neighbor on an outgoing connection that has sent the *headers* message for the same block. $\mathcal{V}$ picks the neighbor according to the time an outgoing connection is established. The neighbor connected the earliest is picked. Since the attack nodes are on $\mathcal{V}$'s incoming connections and the adversary does not know the order of the established time of $\mathcal{V}$'s outgoing connections, with high probability $\mathcal{V}$ picks a non-attack node and receives the block from that neighbor soon after. It is therefore impossible to delay the block by more than the timeout of *getdata* request.

IV. TENDRILSTALLER ATTACK ALGORITHM

TendrilStaller could operate with or without light weight attack nodes. We thus first describe TendrilStaller with only full attack nodes. We then introduce the light weight attack nodes and discuss the full operations of TendrilStaller.

A. *Attack with full attack nodes only*

TendrilStaller requires as few as *3* attack nodes. The attack consists of two phases. In Phase 1, attack nodes try to be selected as the victim's HB neighbors. Three distinct attack nodes establish connections to the victim. Each attack node keeps sending new compact blocks unsolicited to the victim, even if it is not yet in the victim's HBN list. According to the Bitcoin



S_a: Phase 1, keep sending compact blocks;
S_b: Wait for other attack nodes;
S_c: Proceed to Phase 2;
Figure 8. Single-node attack state

source code, when a node $\mathcal{V}$ receives an unsolicited compact block from a non-HB neighbor, $\mathcal{V}$ still tries to reconstruct the block if the following two conditions are met.

- Condition C_1: $\mathcal{V}$ has not seen the block before.
- Condition C_2: $\mathcal{V}$ has already received all the transactions in this compact block before.

We observe that C_2 holds most of the time for any victim $\mathcal{V}$ that has joined the peer-to-peer network long enough to observe a few newly mined blocks. If the reconstructed block is valid, the victim adds the sending attack node to the HBN list. The victim sends a *sendcmpct(true)* message to the sender. Attack nodes implement three methods to increase their chance to be the first node to send the new compact block to the victim.

- Method O_1: attack nodes establish more outgoing connections than the default *8* connections defined in the protocol. As long as any of its neighbors receives the new block, the attack nodes will also receive the block shortly afterwards. Attack nodes are thus likely to get the newly mined blocks faster with more neighbors.
- Method O_2: attack nodes instruct all of its neighbors to send compact blocks unsolicited. Attack nodes does not limit the size of its own HBN list. The attack nodes are more likely to receive new blocks faster.
- Method O_3: upon receiving a new compact block, the attack nodes send it to the victim without fully validating the block. Similar to the attack in [12], bypassing the validation process allows attack nodes to reduce tens of milliseconds in propagating new blocks to the victim.

Attack nodes receive the *sendcmpct(true/false)* messages from the victim. When an attack node receives a *sendcompact* message, it informs all other attack nodes. Each attack node is thus aware of which attack nodes are currently in the victim's HBN list. An attack node that is not yet in $\mathcal{V}$'s HBN list keeps sending compact blocks to $\mathcal{V}$ until it is added to the list. Once an attack node is added to the list, it does not relay compact blocks to $\mathcal{V}$ thus giving other attack nodes a better chance to be added to the HBN list. When there are three attack nodes in the HBN list, all the attack nodes proceed to Phase 2. Each attack node acts independently according to a state transition graph as shown in Figure 8.

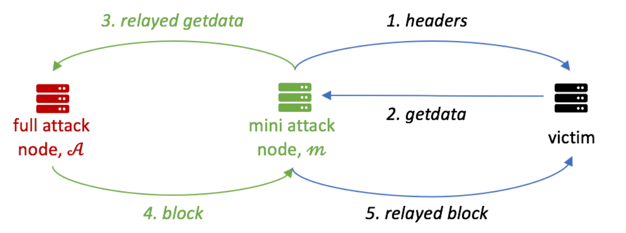


Figure 9. light weight attack node relaying *getdata* request

In Phase 2, the attack nodes perform the actual block delay attack. Attack nodes try to be the first to send the advertisement of new blocks (*headers* messages) to the victim $\mathcal{V}$. Upon receiving a new compact block, the attack node extracts the header and sends it to the victim without the validation process. If $\mathcal{V}$ responds with a *getdata* request, it implies the attack node's advertisement is indeed the first one received and accepted by $\mathcal{V}$. The attack node then delays sending the block by *10* minutes. If no attack node receives the *getdata* request from the victim, it implies that a non-attack node has sent the *headers* message to $\mathcal{V}$ before the attack nodes. The victim subsequently requests the new block from the non-attack node. As a result, that non-attack node will be added to victim's HBN list and one of the attack nodes is removed from the list. Attack nodes monitor the *sendcmpct(false)* messages from the victim. Once at least one attack node is removed from the HBN list, the attack transitions back to Phase 1, as shown in Figure 8. Attack nodes start sending compact blocks again.

B. Light weight attack nodes

Light weight attack nodes consume less CPU, bandwidth and disk space and are cheaper to deploy in the cloud environment. Specifically, a light weight attack node does not store any transaction or block data on the disk. It does not perform the initial block download process as opposed to a full Bitcoin node. The disk size requirement for a light weight attack node can be less than *1G*. In contrast, a full Bitcoin node needs to store at least *230G* of data and the amount of stored data grows with every new block. Additionally, the light weight attack nodes do not perform block validation, significantly reducing the necessary computational power. They do not maintain any blockchain state. A lightweight attack node can operate in either of the two modes. In Mode 1, a light weight attack node implements a minimum subset of functions of a full Bitcoin node. In Mode 1, a light weight attack node performs only one major operation: when it receives a new compact block from its neighbors, it sends the compact block immediately to the full attack nodes. Similar to the full attack nodes, a light weight attack node instruct all of its neighbors to send unsolicited compact blocks. Additionally, light weight attack nodes also perform basic routines such as responding to heartbeat messages from the neighbors to keep the connections alive. They also exchange addresses with neighbors to periodically update the address table.

Light weight attack nodes in Mode 1 act as a proxy for the full attack nodes. Each light weight attack node connects to a

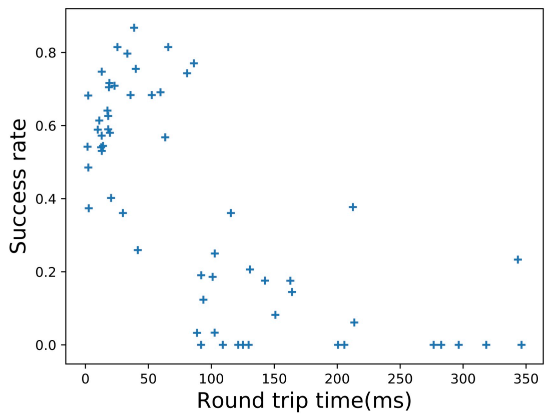


Figure 10. *P_delay* with link latency

random, and at least partly distinct, subset of public Bitcoin nodes. Light weight attack nodes also relay a compact block directly to the full attack node without validating the blocks. Combining the neighbors of all the light weight attack nodes, a large group of Bitcoin nodes are only one hop away from the full attack nodes. Consequently, full attack nodes could receive newly mined blocks earlier on average.

Light weight attack nodes can also operate in Mode 2, where they participate in the attack and connect to the victims. Light weight attack nodes implement the same two-phase attack logic as a full attack node. Additionally, since a light weight attack node does not store any block, it needs to implement the logic to relay *getdata* requests from the victim to the full attack nodes, and the corresponding response from the full attack nodes back to the victim. As shown in Figure 9, if a light weight attack node m succeeds in Phase 2 to send a *headers* message first, the victim replies with a *getdata* request. As we do not want to actually delay the block to the victim, m needs to respond promptly with the actual block. For this purpose, m relays the request to the full attack node $\mathcal{A}$. $\mathcal{A}$ responds accordingly, sending the actual block to m. Finally, m relays the response back to the victim.

When light weight attack nodes operate in Mode 2, the adversary could substitute original full attack nodes by light weight nodes, leaving only one full attack node. It is cheaper to deploy light weight nodes in the cloud than a full attack node. We demonstrate the cost reduction in Section V.C.

V. Experiment

In Section V.A, we setup *3* full attack nodes in the cloud with different geographic locations to show the feasibility of the attack. We randomly select victims around the globe and perform “partial” attack on them. We demonstrate that the attack is more effective as attack nodes are closer to the victims. In Section V.B, we perform attack with *3* full attack nodes and a group of light weight attack nodes. The light weight attack nodes operate in Mode 1 as a proxy for the full attack nodes. We demonstrate that higher percentage of blocks could be delayed with the adoption of light weight attack nodes than otherwise. In Section V.C, we perform the attack with one full attack node and multiple light weight nodes operating in Mode 2. We demonstrate that the attack could be more cost-effective compared to the attack with full attack nodes only.

	Percentage of delayed blocks		
Attack location	US	Europe	Asia
US	**0.60**	0.11	0.03
Europe	0.15	**0.65**	0.02
Asia	0.11	0	**0.60**

Table 1.a. Average Pdelay

	Link latency (ms)		
Attack location	US	Europe	Asia
US	**39.4**	112.8	211.4
Europe	130.1	**27.7**	232.6
Asia	231.7	309.5	**33.5**

Table 1.b. Average link latency

A. *Attack with full attack nodes only*

We show that as long as the victim is moderately close to the attack nodes, the attack could succeed with high probability. We setup *3* full attack nodes, each connecting to *200* neighbors. The three attack nodes are deployed in the Google Cloud Platform. We repeat the experiments in *3* different regions, US East, Europe and Asia respectively. This is to remove any geographical bias of the attack. We randomly select victims around the globe and perform “partial attack” on the selected victims without actually delaying the blocks. We limit our experiment to a “partial attack” to validate our attack without actually impacting the operation of Bitcoin nodes. In attack Phase 2, when an attack node manages to send a *headers* message first and receive the *getdata* request from the victim, the attack node responds to the request with no delay. We treat each *getdata* request in Phase 2 as validation of a successful attack. We implement the attack in a way that attack nodes could carry out attack targeting at multiple victims in parallel. The node maintains a separate attack state shown in Figure 8 for each victim. When receiving a new block, the node takes corresponding action independently for each victim.

We define the *success rate* as the percentage of delayed blocks during the attack. Figure 10 shows the success rate with different round-trip time between the attack nodes and the victim. Each data point represents a trial containing from *150* to *200* blocks. The measurement starts from the first block after the attack is launched and ends at a block when an attack node is removed from victim’s HBN list. Table 1 summarizes the result in each individual region. As attack nodes are closer to the victims, the attack could be more effective with higher success rate. When the round-trip time is less than *80ms*, the success rate can be as high as *85%.* Considering the wide availability of cloud providers over the globe, TendrilStaller could be easily deployed.

We also observe a notable difference in the result even when two victims have similar round-trip delay to the attack nodes. As we do not have access to the victim nodes, we conjecture the reasons as follows. First, the number of connections varies across the victims. Some victims may only connect to few nodes, while other victims establish many more connections. Attacking a node with more connections are harder, as the attack nodes need to compete with more non-attack nodes in the attack Phase 2. Second, certain victims may be closer to the sources of new blocks on average. They seem to be connected to pool gateways.

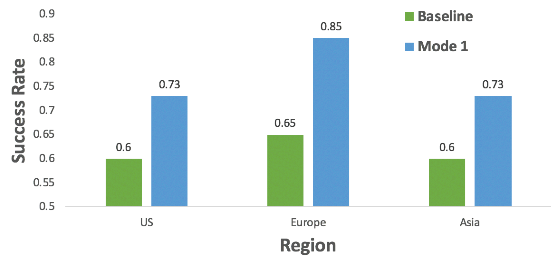


Figure 11. Light weight attack nodes in Mode 1

In such scenario, the victims receive newly mined blocks much more promptly. It is harder for the attack to succeed. Third, the victim may not follow the default protocol. For example, victims may increase its HBN list size, or never evicts any neighbor from its HBN list.

B. *Attack with light weight attack nodes in Mode 1*

We now perform attacks with light weight attack nodes operating in Mode 1. In each region we set up *3* full attack nodes and *5* light weight attack nodes. We choose to deploy light weight nodes in the same region to minimize latency between the full attack nodes and its light weight counterparts. These light weight nodes are set to connect to *200* random nodes in the network, and instruct all their neighbors to only send compact blocks. They relay the compact blocks on to the three attack nodes upon receipt. Justified by the result from Section V.A, we only select victims from the same geographical region as the attack nodes. Again, we perform a partial attack where the response to the victim's *getdata* request is not delayed. We treat the victim's *getdata* request as an indicator of the successful attack. We demonstrate that full attack nodes can receive new blocks faster with the help of light weight attack nodes. Consequently, more blocks could be delayed to the victim.

Attack location	US	Europe	Asia
Percentage	0.590	0.491	0.578

Table 2. Blocks relayed by light weight attack nodes

As in Figure 11, there is a marked improvement on the success rate in all of the three regions. As the light weight nodes act as a proxy, the success rate increases by *13%*, *20%* and *13%* in the three regions. We further validate this by recording the first senders of the new blocks received by the full attack nodes. As shown in Table 2, around *59%, 49%* and *58%* percent of the new blocks are relayed by the light weight nodes respectively. These blocks would have been received at a later time if not for the adoption of the light weight nodes.

This increase in efficacy is achievable through minimal increase in resources, due to the low-cost nature of the light weight nodes, as we demonstrate in Section V.C.

C. *Attack with light weight attack nodes in Mode 2*

We now perform attacks with light weight attack nodes operating in Mode 2. We demonstrate that the (partial) substitution of the full attack nodes with light weight attack nodes could reduce the cost of attack while achieving the same attack success rate. We set up two light weight attack nodes and

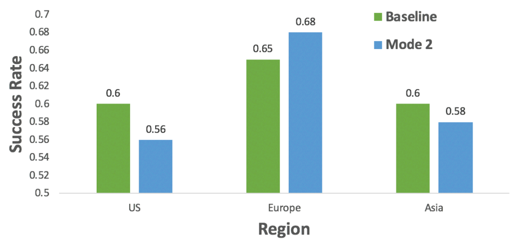


Figure 12. Light weight attack nodes in Mode 2

	CPU	Disk	Bytes sent
Full attack node	71.3%	>235G	88.4 kbps
Light weight node	40.3%	<1G	7.4 kbps

Table 3. Resource consumption comparison

one full attack node in each region. The victims are randomly selected nodes in the same geographical region as the attackers.

Figure 12 shows that the attack success rate is *56%, 68%* and *58%* respectively, which is comparable to that of the *3*-full-attack-node scenario in Section V.A. This demonstrates that we can use light weight attack nodes in conjunction with a full attack node to effectively delay blocks while utilizing fewer resources. Table 3 compares the resource consumption of the full attack node and light weight attack node. As light weight attack nodes do not store the blocks, operation requires much less disk space. Light weight attack nodes do not verify transaction/blocks, which leads to the reduced CPU consumption. We also configure the light weight nodes not to relay any blocks to the neighbors other than the full attack node and victims. Consequently, the outbound bandwidth consumption is lower.

The reduced cost for the attack indicates that the adversary could deploy light weight attack nodes on smaller, cheaper-priced instances. For the same resource cost, the adversary could setup more nodes and attack more victims simultaneously. This would allow malicious entities to attack larger sections of the network than possible with the same resource consumption.

D. *Impact of the block delay attack*

The block delay attack effectively reduces the victim's mining power. The victim node could not mine on the newest block. The percentage of non-effective mining power can be calculated as follows. Assuming the worst case where each newly mined block is delayed by *10* minutes. The time interval between two blocks follows an exponential distribution and the average block interval is *10* minutes [14]. The expected time that a victim could mine on the newest block is:

$$\mathrm{T} = \int_{10}^{\infty} (t-10)\lambda e^{-\lambda t} dt$$

Where $\lambda = 0.1\ min^{-1}$. This is *36.7%* of the total mining power of the victim. The attack also encourages miners to form mining pools, making the problem of centralization of mining power even worse. Pool gateway nodes are known to operate modified Bitcoin source code for enhanced security and faster block relay. For example, pool gateway nodes often establish many more connections [23]. Based on the observations in Section V.A, the attack success rate would be lower for nodes with more connections or not following the default Bitcoin behavior, such as pool gateway nodes. On the other hand, when a miner joins a mining pool, it only establishes connections with the pool manager. The TendrilStaller attack is thus not feasible on these miners. Given these observations, an individual miner would prefer to join the mining pool to avoid TendrilStaller attack.

VI. Countermeasures

Some of the countermeasures proposed in [12] are already integrated into current Bitcoin protocol such as the *headers-first* announcement. A node only sends a *getdata* request if it has received a valid header of the new block. In this paper, we propose a few countermeasures that can improve the reliability of the newly introduced compact block propagation protocol.

A. Partially-random selection of HB neighbors

According to the current protocol, a node by default selects its HB neighbors if that neighbor has sent a valid block recently. This gives attack nodes the chance to manipulate victim's HBN list, as in our proposed attack algorithm. To limit the impact of attack nodes on the HBN list, a Bitcoin node could only select *2* HB neighbors according to the *most-recent-sent-block* rule and select the one remaining HB neighbor randomly. Attack nodes then could not fully control the victim's HBN list and with high probability there will always be a non-attack node that can send compact blocks unsolicitedly to the victim.

B. HB neighbor on outgoing connections

Similarly, a Bitcoin node could select HB neighbors only from the nodes on outgoing connections. Since attack nodes usually initiate the connections to the victim, they will not be selected as HB neighbors. Attack nodes could then induce the victim to connect to them, though. For example, the adversary could perform complex operations to manipulate the victim's address tables as in the eclipse attack in [19]. Such process is both resource and time consuming.

C. Shorter block download timeout

We observe that as of Dec 2018, the *90*-percentile block propagation time has reduced significantly to less than *4* seconds as measured in [18]. This indicates that the network bandwidth provisioned to the Bitcoin nodes has dramatically increased. This also makes the current *10*-minute timeout of the *getdata* request obsolete. Reducing the block download timeout reduces the impact of the attack. Given a k-minute timeout, the reduced mining power due to the block delay attack is $e^{-\lambda k}$. If the download timeout is shortened to *1* minute, an attack node succeeding delaying every block to the victim can only reduce the victim's mining power to *90.4%* of its original.

VII. Conclusion

We present TendrilStaller, an attack targeting Bitcoin's peer-to-peer network. Our attack allows an adversary to delay block propagation to a victim for *10* minutes. We propose and implement the new attack algorithm and demonstrate that the attack success rate can be as high as *85%.* Furthermore, the adoption of light weight attack nodes can either increase the attack success rate (in Mode 1) or reduce the resource consumption of an attack (in Mode 2). Moreover, we showed that a successful attack could reduce the victim's mining power to *36.7%*. This allows the adversary to gain much higher mining advantage. We further propose several countermeasures to mitigate this TendrilStaller attack.

References

[1] https://www.ethereum.org

[2] Mettler, M., 2016, September. Blockchain technology in healthcare: The revolution starts here. In e-Health Networking, Applications and Services (Healthcom), 2016 IEEE 18th International Conference on (pp. 1-3). IEEE.

[3] Su, S., Wang, K. and Kim, H.S., 2018, July. SmartSupply: Smart contract based validation for supply chain blockchain. In *The 2018 IEEE International Conference on Blockchain*. IEEE.

[4] Wang, K., Zhang Z., and Kim, H.S., 2018, July. ReviewChain: Smart contract based review system with multi-blockchain gateway. In *The 2018 IEEE International Conference on Blockchain*. IEEE.

[5] Liang, X., Shetty, S., Tosh, D., Kamhoua, C., Kwiat, K. and Njilla, L., 2017, May. Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (pp. 468-477). IEEE Press.

[6] Eyal, I. and Sirer, E.G., 2018. Majority is not enough: Bitcoin mining is vulnerable. Communications of the ACM, 61(7), pp.95-102.

[7] M. Rosenfeld, "Analysis of hashrate-based double spending," Tech. Rep., 2012 [Online]. Available: https://bitcoil.co.il/Doublespend.pdf

[8] Ghassan O. Karame, Elli Androulaki, Marc Roeschlin, Arthur Gervais, and Srdjan Cˇapkun. Misbehavior in bitcoin: A study of double-spending and accountability. ACM Trans. Inf. Syst. Secur., 18(1):2:1–2:32, May 2015.

[9] Vector67. (2011). Fake Bitcoins? [Online]. Available: https://bitcointalk.org/index.php?topic=36788.msg463391\#msg463391

[10] Nicolas T. Courtois and Lear Bahack. On subversive miner strategies and block withholding attack in bitcoin digital currency. CoRR, abs/1402.1718, 2014.

[11] corbixgwelt. (2011, May). Timejacking & Bitcoin [Online]. Available: http://culubas.blogspot.de/2011/05/timejacking-bitcoin_802.html.

[12] Gervais, A., Ritzdorf, H., Karame, G.O. and Capkun, S., 2015, October. Tampering with the delivery of blocks and transactions in bitcoin. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (pp. 692-705). ACM.

[13] https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki

[14] Nakamoto, S., 2008. Bitcoin: A peer-to-peer electronic cash system.

[15] https://github.com/bitcoin/bitcoin

[16] Gencer, A.E., Basu, S., Eyal, I., van Renesse, R. and Sirer, E.G., 2018. Decentralization in bitcoin and ethereum networks. arXiv preprint arXiv:1801.03998

[17] https://github.com/kewangcmu/block_delay_attack

[18] https://dsn.tm.kit.edu/bitcoin/index.html

[19] Heilman, E., Kendler, A., Zohar, A. and Goldberg, S., 2015, August. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In USENIX Security Symposium (pp. 129-144).

[20] https://bitcoin.org/en/glossary/unspent-transaction-output

[21] Apostolaki, M., Zohar, A. and Vanbever, L., 2017, May. Hijacking bitcoin: Routing attacks on cryptocurrencies. In Security and Privacy (SP), 2017 IEEE Symposium on (pp. 375-392). IEEE

[22] Nayak, K., Kumar, S., Miller, A. and Shi, E., 2016, March. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In Security and Privacy (EuroS&P), 2016 IEEE European Symposium on (pp. 305-320). IEEE.

[23] Miller, A., Litton, J., Pachulski, A., Gupta, N., Levin, D., Spring, N. and Bhattacharjee, B., 2015. Discovering bitcoin's public topology and influential nodes. et al.