Tampering with the Delivery of Blocks and Transactions in Bitcoin

Arthur Gervais[†], Hubert Ritzdorf[†], Ghassan O. Karame[‡] and Srdjan Čapkun[†]
[†]ETH Zurich, Switzerland, [‡]NEC Laboratories Europe, Germany
[†]firstname.lastname@inf.ethz.ch, [‡]firstname.lastname@neclab.eu

ABSTRACT

Given the increasing adoption of Bitcoin, the number of transactions and the block sizes within the system are only expected to increase. To sustain its correct operation in spite of its ever-increasing use, Bitcoin implements a number of necessary optimizations and scalability measures. These measures limit the amount of information broadcast in the system to the minimum necessary.

In this paper, we show that current scalability measures adopted by Bitcoin come at odds with the security of the system. More specifically, we show that an adversary can exploit these measures in order to effectively delay the propagation of transactions and blocks to specific nodes—without causing a network partitioning in the system. We show that this allows the adversary to easily mount Denial-of-Service attacks, considerably increase its mining advantage in the network, and double-spend transactions in spite of the current countermeasures adopted by Bitcoin. Based on our results, we propose a number of countermeasures in order to enhance the security of Bitcoin without deteriorating its scalability.

1. INTRODUCTION

Bitcoin has received more adoption and popularity than any other digital currency proposed to date. Currently, Bitcoin holds the largest market share amongst all existing digital currencies, with a market cap exceeding 3 billion USD [1]. There are also numerous businesses [7], exchange platforms, and banks [11] that are currently built around the Bitcoin ecosystem.

Transaction security is ensured in Bitcoin by means of *blocks* which instantiate a hash-based Proof of Work (PoW) mechanism; Bitcoin's PoW requires 10 minutes on average to be solved and 6 consecutive blocks are recommended to confirm any single transaction in the system. This suggests that transactions will be confirmed in the system after almost one hour—provided that the majority (>50%) of the computing power in the network is honest.

Given the growing investments in Bitcoin, and the increasing adoption by users, the security, scalability, and reliability of Bitcoin has received considerable attention in the literature. Recently, several studies have challenged the security assumptions adopted by Bitcoin [10, 16, 17]. For instance, Eyal and Sirer [12] showed that selfish miners that command more than 33% of the total computing power of the network can acquire a considerable mining advantage in the network. Additionally, Karame *et al.* [17] showed that the measures adopted in Bitcoin to handle fast payments are not enough to deter double-spending attacks, and proposed a countermeasure which is currently integrated in Bitcoin. Recently, Heilman *et al.* showed how to attack the Bitcoin mining protocol by monopolizing the connections of nodes in the system.

Given that the Bitcoin overlay network is densely connected, most of these studies assume that all transactions and blocks (and their order of execution) advertised in the system will be almost immediately available to the majority of Bitcoin nodes soon after their release into the network. In this paper, we challenge this assumption, and we show that an adversary can deny the delivery of blocks and transactions to victim Bitcoin nodes for a considerable amount of time. This is achieved by exploiting Bitcoin bandwidth optimization techniques and the measures that are in place to tolerate network delays and congestion. In our attack we, however, do not require or cause any network partitioning in the Bitcoin network. We show instead that the delivery of blocks and transactions can be delayed by a resource-constrained adversary and does not require the adversary to monopolize the connections of his victims [16]. The minimal requirement for this attack to succeed is simply that the attacker can establish at least one connection to the victim. An even more powerful attack resulting in almost indefinite delays at the victim node only requires that the attacker can fill the victim's remaining open connection slots.

Our results therefore motivate the need for a careful design of the scalability mechanisms adopted in Bitcoin. While existing mechanisms limit the amount of propagated information in the system to the minimum necessary, these techniques come at odds with security and reduce the ability of the network to e.g., detect double-spending attacks, resolve, or prevent blockchain forks. For instance, our findings suggest that the 33% bound outlined by Eyal and Sirer [12] might be even further reduced. Namely, if miners were able to temporally prevent the propagation of recently mined blocks in the network, these miners can further increase their advantage (e.g., by following a similar strategy to [12])—while commanding less than 33% of the total computing power in the network. In this respect, we propose a modification of the block request process in Bitcoin to deter this misbehavior.

Our findings additionally uncover new vulnerabilities in the Bitcoin network's ability to handle fast payments. Fast payments refer to payments where the time between the exchange of currency and goods is short (in the order of a minute). In light of the vulnerabilities discovered in [17], Bitcoin incorporated a new countermeasure which consists of relaying the first double-spent transaction in order to enhance the security of fast payments [3]. In this respect, we show analytically and experimentally that an adversary can leverage our findings to circumvent this countermeasure and double-spend fast payments without bearing any risk of losing her money. We also show that payments which were only confirmed by few blocks might be reverted by the adversary. Based on our findings, we explore a number of solutions to deter this misbehavior and we estimate a lower-bound on the waiting time required to ensure the security of fast payments.

More specifically, our contributions in this paper can be summarized as follows:

- We show that a resource-constrained adversary can abuse existing scalability measures adopted in Bitcoin clients in order to deny information about newly generated blocks and transactions to Bitcoin nodes for at least 20 minutes. We then extend our findings and show how an adversary can continuously deny the delivery of such information.
- We validate our analysis by means of implementation using a handful of hosts located around the globe. Our results demonstrate the feasibility and easy realization of our attacks in current Bitcoin client implementations.
- We show that our results allow the adversary to easily mount Denial-of-Service attacks, considerably increase its mining advantage in the network or double-spend transactions in spite of the current countermeasures adopted by Bitcoin.
- We propose a number of mitigations for hardening the security of the network against such a misbehavior without deteriorating the scalability of Bitcoin. Namely, we propose a modification of the block request management system in Bitcoin in order to detect any misbehavior in the delivery of blocks. Additionally, we leverage our findings to estimate the minimum amount of waiting time required to ensure, with considerable probability, the security of fast payments in Bitcoin.

The remainder of the paper is organized as follows. In Section 2, we overview Bitcoin and summarize the measures deployed in the system to minimize the amount of propagated information. In Section 3, we show how an adversary can effectively delay information propagation in the network. In Section 4, we extend our analysis and discuss possible techniques to continuously prevent the delivery of Bitcoin blocks. In Section 5, we discuss the impact of our findings on the security of Bitcoin, and we outline possible countermeasures in Section 6. In Section 7, we overview related work in the area, and we conclude the paper in Section 8.

2. SCALABILITY MEASURES IN BITCOIN

In this section, we overview Bitcoin, and summarize the main scalability measures integrated in the system.

Background.

Bitcoin is a loosely-connected P2P network, where nodes can join and leave the network at any moment. Bitcoin nodes are connected to the overlay network over TCP/IP. Initially, peers bootstrap to the network by requesting peer address information from Domain Name System (DNS) seeds which provide a list of current Bitcoin node IP addresses. Newly connected nodes advertise peer IP addresses via Bitcoin *addr* messages. Notice that a default full Bitcoin client establishes a maximum of 125 TCP connections, of which up to 8 are outgoing TCP connections.

Bitcoin transactions transfer coins (BTCs) among peers; these peers are referenced in each transaction by means of virtual pseudonyms—referred to as *Bitcoin addresses*. Each address maps to a unique public/private key pair; these keys are used to transfer the ownership of BTCs among addresses. A Bitcoin transaction is formed by digitally signing a hash of the previous transaction where this coin was last spent along with the public key of the future owner and incorporating this signature in the coin. Transactions take as input the reference to an output of another transaction which spends the same coins, and outputs the list of addresses which can collect the transferred coins. A transaction output can only be redeemed once, after which the output is no longer available to other transactions. Once ready, the transaction is signed by the user and broadcast in the P2P network. Any peer can verify the authenticity of a BTC by checking the chain of signatures.

The difference between the input and output amounts of a transaction is collected in the form of fees by Bitcoin *miners*. Miners are peers, which participate in the generation of Bitcoin blocks. These blocks are generated by solving a hash-based proof-of-work (PoW) scheme; miners must find a nonce value that, when hashed with additional fields (e.g., the Merkle hash of all valid transactions, the hash of the previous block), the result is below a given target value. If such a nonce is found, miners then include it in a new block thus allowing any entity to verify the PoW. Since each block links to the previously generated block, the Bitcoin *blockchain* grows upon the generation of a new block in the network. A Bitcoin block is mined on average every 10 minutes and currently awards 25 BTCs to the generating miner. It was shown in [17] that Bitcoin block generation follows a shifted geometric distribution with parameter 0.19. This also suggests that there is considerable variability in the generation times; for example, one of the longest block generation time so far lasted almost 99 minutes (this corresponds to block 152,218).

During normal operations, miners typically work on extending the longest blockchain in the network. Due to the underlying PoW scheme, however, different miners can potentially find different blocks nearly at the same time—in which case a fork in the blockchain occurs. Forks are inherently resolved by the Bitcoin system; the longest blockchain which is backed by the majority of the computing power in the network will eventually prevail.

Scalability Measures.

Currently, almost 1 transaction per second (tps) [2] is executed in Bitcoin; this results in an average block size of almost 400 KB[1]. In an experiment that we conducted, we measured the amount of traffic observed by a full Bitcoin node[2]; our results show that Bitcoin nodes in- and outbound traffic heavily depends on the number of connections of the node. For instance, a node with about 70 connections witnesses on average 8.5 GB daily traffic[3], while a node with about 25 connections witnesses on average 3 GB traffic within 24 hours[4]. Given the increasing adoption of Bitcoin, the number of transactions, and the block sizes are only expected to increase. For example, if Bitcoin were to handle 1% of the transactional volume of Visa[5], then Bitcoin needs to scale to accommodate almost 500 tps—requiring considerably larger blocks to be broadcasted in the network.

Motivated by these factors, the current Bitcoin protocol implements various bandwidth optimizations, and measures in order to sustain its scalability and correct operation in spite of ever-increasing use. In what follows, we detail the existing measures taken by Bitcoin developers.

MEASURE 1. *Bitcoin combats the broadcasting of ill-formed blocks and transactions by maintaining an internal reputation management system.*

Whenever a node receives objects (e.g., blocks, transactions), it checks their correctness before forwarding them to other peers in the network. First, objects are validated based on their respective

[1]The maximum block size is currently limited to 1 MB which corresponds to less than 7 transactions per second.

[2]Our measurements were conducted over a period of 18 days, during which our node had on average 70 active connections.

[3]Measured over 20 days.

[4]Measured over 2 days.

[5]Currently, the Visa network is designed to handle peak volumes of 47,000 tps [5].

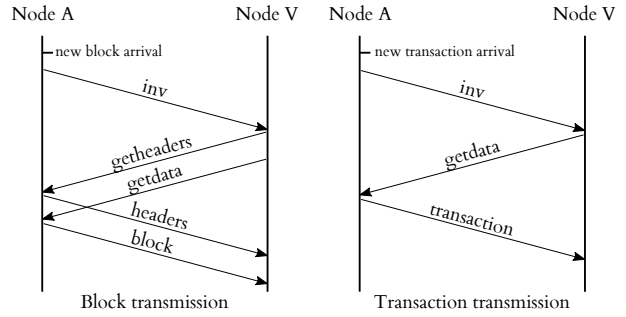

Figure 1: Summary of the request management system in Bitcoin.

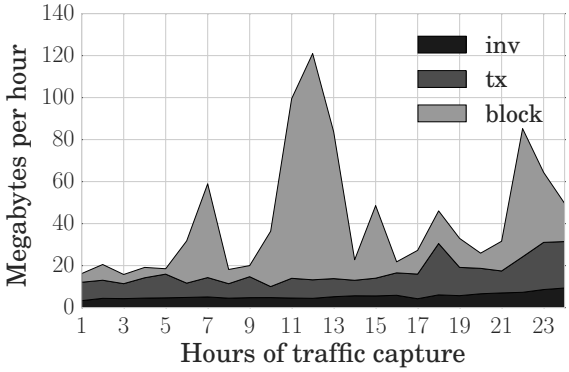

Figure 2: Hourly traffic distribution of a Bitcoin node, up and downstream, over 24 hours, w.r.t. to different Bitcoin messages. Here, 'tx' denotes transactions.

syntax and sizes, e.g., oversized objects are discarded. If this verification passes, the contents of the objects are subsequently validated. For transactions, this includes verifying the signature, the input and output coins used in the transaction; similarly, the PoW included in block headers is verified with respect to the current network difficulty.

To prevent any abuse of the Bitcoin overlay network (e.g. Denial of Service attacks), a receiving node locally assigns a penalty to peers who broadcast ill-formed objects. Once a node has reached 100 penalty points, the receiving node disconnects from the misbehaving node for 24 hours. For example, if a node broadcasts invalid alerts, then it will be given 10 penalty points. Nodes which attempt more serious misbehavior, such as inserting invalid transaction signatures, are immediately assigned 100 points, and therefore directly banned. Penalties also apply to ill-formed control messages such as *inv* (inventory) or *addr* commands. Notice that locally assigned penalties are not transmitted to other peers.

MEASURE 2. *Bitcoin uses an advertisement-based request management system to minimize the information spread in the network.*

To minimize information spread in the network, messages are propagated in the Bitcoin network with the help of an advertisement-based request management system. Namely, if node $\mathcal{A}$ receives information about a new Bitcoin object (e.g., a transaction or a block) from another node, $\mathcal{A}$ will advertise this object to its other connections (e.g. node $\mathcal{V}$) by sending them an *inv* message; these messages are much smaller in size than the actual objects, because they only contain the hash and the type of object which is advertised. Only if node $\mathcal{V}$ has not previously received the object advertised by the *inv* message, $\mathcal{V}$ will request the object from $\mathcal{A}$ with a *getdata* request. Following the Bitcoin protocol, node $\mathcal{A}$ will subsequently respond with a Bitcoin object, e.g., the contents of a transaction or a block.

By doing so, inventory messages limit the amount of data broadcast in the network. Notice that in case the object advertised corresponds to a block, neighbor $\mathcal{V}$ first requests the block header before the actual block data. Here, when a block header is advertised via a *headers* message, the receiving node internally stores the highest block known by the sending peer. The receiving node also validates any received header, by verifying its corresponding PoW. Transactions, on the other hand, are requested directly using a transaction *inv* message. This process is summarized in Figure 1.

To minimize bandwidth consumption, Bitcoin nodes request a given object only from a single peer, typically the first peer which first advertises the object. In an experiment that we conducted, we measured the traffic (cf. Figure 2) witnessed by a default Bitcoin client over a period of 24 hours[6]. We observe that, indeed, the transmission of blocks consumes a significant part of the bandwidth of our client. Requesting the same object from multiple peers entails downloading the same data several times, and therefore can only increase the bandwidth consumption of the system.

MEASURE 3. *Bitcoin relies on static timeouts in order to prevent blocking while tolerating network outages, congestion, and slow connections.*

Given that Bitcoin runs atop an overlay network, communication latency and reliability pose a major challenge to the correct operation of the system. Currently, Bitcoin relies on timeouts in order to tolerate network delays—while preventing blocking. Blocking can occur e.g., when a node stops responding during communication.

For example, in Bitcoin version 0.10, the Bitcoin developers introduced a block timeout download of 20 minutes[7]. Similarly, for transactions, the Bitcoin developers introduced a 2-minute timeout. Notice that the choice of the timeout is a non-trivial task and depends on a number of parameters such as bandwidth, object size, latency, processing power, and the Bitcoin version of each node. On the one hand, overly long timeouts might deteriorate the quality of service of the whole network and can be abused to conduct e.g., double-spending attacks [4]. On the other hand, short timeouts might hinder effective communication under varying network conditions, or when communicating with slow peers.

In an experiment that we conducted, we measured the block transmission times in Bitcoin with respect to the nodes' bandwidths. Our results (cf. Figure 3) show that block retrieval times significantly vary depending on the connection of the node. For instance, fast nodes with 500 Mbps downstream take an average of 1.55 seconds to download a block, while slow nodes equipped with a 256 Kbps connection take an average of 71.70 seconds to download blocks. It is interesting to note that, during our measurements on the slow node, 0.5% of the generated blocks exceeded the 20 minute timeout adopted by current clients. *This shows that the current timeouts adopted in Bitcoin when sending/receiving blocks are indeed necessary for the correct delivery of blocks in the network.*

MEASURE 4. *Bitcoin clients keep track of the order of the received transaction advertisements. If a request for a given transaction is not delivered, the next peer in the list is queried.*

[6]During our measurements, the client was connected to approximately 30 neighbors.

[7]Available from `https://github.com/bitcoin/bitcoin/pull/5608`

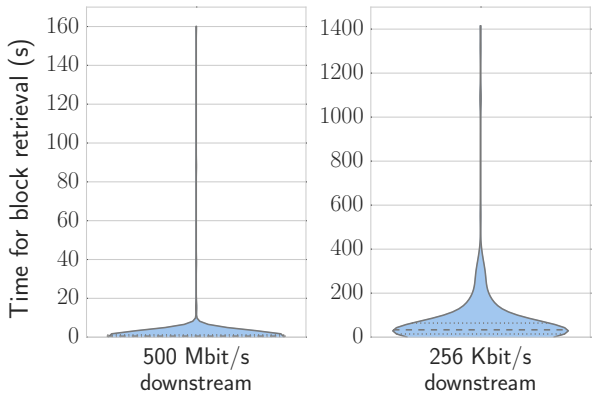

Figure 3: Block transmission times with respect to the connection speed. Here, we evaluate the time to download 400 consecutive Bitcoin blocks on a 500 Mbps and 256 Kbps connection.

When a transaction T is advertised via an *inv* message to a given node, the latter keeps track of the order of announcements with a first-in first-out (FIFO) buffer. Each time a peer advertises T, the peer is inserted in the buffer. Transaction T is only requested from one peer at a time. For each entry in the buffer, Bitcoin clients maintain a 2-minute timeout, after which, the next entry in the buffer is queried for T.

3. DELAYING INFORMATION DELIVERY

In this section, we show how an adversary can exploit the scalability measures adopted in Bitcoin in order to delay the delivery of message advertisements.

For the purpose of our analysis, we assume that the adversary $\mathcal{A}$ is a full Bitcoin node, and has access to the entire blockchain. Here, the objective of $\mathcal{A}$ is to deny the delivery of an object O for a short period of time from a specific node denoted by $\mathcal{V}$. In Section 4, we extend this analysis and show how an adversary can arbitrarily prolong the prevention of message delivery.

3.1 Necessary Requirements

We start by outlining the necessary conditions which need to be satisfied for $\mathcal{A}$ to successfully prevent the delivery of object information from a given Bitcoin node $\mathcal{V}$.

Requirement 1: $\mathcal{A}$ must be the first peer to send a message advertisement of object O to $\mathcal{V}$. If $\mathcal{V}$ requests an object O from a node which is not under the control of $\mathcal{A}$, then little can be done by $\mathcal{A}$ to prevent the node from sending O to $\mathcal{V}$. However, if $\mathcal{A}$ is able to first advertise O via an *inv* message to $\mathcal{V}$, then $\mathcal{V}$ will temporarily abstain from requesting O from any other node in the network. As explained earlier, Bitcoin nodes only request the same data element from a single peer in order to minimize bandwidth consumption (cf. Measure 2). In order to be the first to advertise an object O to $\mathcal{V}$, $\mathcal{A}$ needs to be an immediate neighbor of $\mathcal{V}$ in the Bitcoin overlay network (i.e., there is a direct TCP connection between $\mathcal{A}$ and $\mathcal{V}$).

Requirement 2: $\mathcal{V}$ should wait sufficiently long after a *getdata* message before requesting the data from another peer. The longer $\mathcal{V}$ waits for $\mathcal{A}$ to send O, the more damaging is the misbehavior of $\mathcal{A}$ (cf. Section 5). As mentioned earlier, current Bitcoin implementations inflict a timeout of 20 minutes on $\mathcal{V}$ for blocks, and 2 minutes for transactions; after the timeout, $\mathcal{V}$ can request O from another node (cf. Measure 3).

In what follows, we show how these requirements can be satisfied.

Satisfying Requirement 1.

As mentioned earlier, Bitcoin nodes verify the correctness of every received object before they re-broadcast it in the network. Notice that this process requires considerable time; for example, when receiving transactions, nodes verify the transactions' signatures, and check that the input coins have not been spent previously. Similarly, for blocks, nodes check the correctness of the PoW in relation with the current difficulty in the network, and verify the correctness of every transaction that is confirmed in the block.

This offers a clear advantage for an adversary in order to satisfy Requirement 1. Indeed, an adversary can simply forward a given object of choice O immediately after it receives it—without verifying its correctness. Since all remaining nodes in the system will have to perform a series of verification steps to verify O, the adversary is likely to be the first node to advertise O to its neighbors. As explained in Measure 2, this ensures that $\mathcal{A}$'s neighbors will not request the object from any other node in the network until the timeout expires. Notice, that if $\mathcal{A}$ is interested in denying the delivery of O to a specific node $\mathcal{V}$, then the adversary can advertise O solely to $\mathcal{V}$. Here, in the case where O is created by $\mathcal{A}$ (e.g., O is a transaction), $\mathcal{A}$ can satisfy Requirement 1 by first advertising O to $\mathcal{V}$, before broadcasting it in the network.

We stress here that $\mathcal{A}$ should be directly connected to $\mathcal{V}$. This is a reasonable assumption, since most full Bitcoin nodes do not exhaust their maximum 125 connections; in typical cases, nodes are therefore likely to accept connection request originating from $\mathcal{A}$.

Satisfying Requirement 2.

To optimize bandwidth, recall that objects are only requested from one peer at a time (cf. Measure 2). This means that if $\mathcal{A}$ is the first node to advertise O to $\mathcal{V}$, $\mathcal{V}$ will simply wait for $\mathcal{A}$ to send O and will not request the same object from any other peer. During this period, $\mathcal{A}$ simply has to respond to the active ping messages of $\mathcal{V}$.

For blocks, the default timeout for $\mathcal{V}$ is set to 20 minutes; transactions, however, have a waiting timeout of 2 minutes. Given Measure 4, it is easy for $\mathcal{A}$ to increase the timeout for transactions simply by sending x back to back *inv* messages for the same transaction. By doing so, $\mathcal{A}$ effectively increases the timeout specific to the advertised transaction by $2x$ minutes. We validate this analysis experimentally in Section 3.2.

3.2 Experimental Validation

In what follows, we empirically assess the probability that $\mathcal{A}$ succeeds in relaying first an *inv* message of an object O to a given node $\mathcal{V}$ (cf. Requirement 1). For this purpose, we implement a proxy which relays specific *inv* messages directly before passing them to the Bitcoin software. Based on this proxy, we evaluate how likely $\mathcal{A}$ can be the first node to advertise an *inv* message to $\mathcal{V}$.

Our experimental setup is sketched in Figure 4. Here, we assume that $\mathcal{A}$ is located in Europe and operates an Intel i7 CPU (3.40GHz) with 32 GB RAM and a 500/400 Mbps Internet connection. In our setup, $\mathcal{A}$ connects to $\mathcal{V}$ through a direct TCP connection.

To remove any bias that might occur from a given network topology, we consider 10 different nodes emulating $\mathcal{V}$ geographically dispersed around the globe. We also vary the connectivity of $\mathcal{A}$ and $\mathcal{V}$ during our measurements (i.e., between 40 and 800 TCP connections to different full Bitcoin nodes for $\mathcal{A}$ and between 40 and

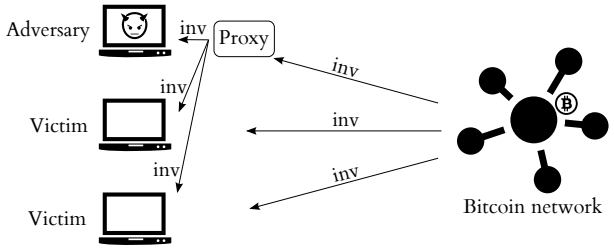

Figure 4: Satisfying Requirement 1. The adversary can use a simple relay proxy to forward *inv* messages without validating the correctness of the corresponding object.

200 for $\mathcal{V}$). Because only full Bitcoin nodes forward blocks and transactions, only connections to full Bitcoin nodes are relevant for this experiment. We, therefore, modify the Bitcoin client, such that only connections to full Bitcoin clients are established and such that the maximum connection limit is kept constantly.

We measure the probability $P_r = \frac{r}{N}$, that $\mathcal{A}$ satisfies Requirement 1 as follows. We relay N advertisements for objects to $\mathcal{V}$, and we compute the number of *getdata* and *getheaders* replies r originating from $\mathcal{V}$ received within the subsequent 30 seconds; recall that these messages provide sufficient evidence that $\mathcal{A}$ is the first relayer of an object to $\mathcal{V}$, and as such $\mathcal{V}$ will not request this object from elsewhere.

Denying the Delivery of Blocks.

We start by investigating the success probability of $\mathcal{A}$ in denying the delivery of blocks. Here, for different combinations of the number of connections of $\mathcal{A}$ and $\mathcal{V}$ (cf. Table 1), we forward $N = 100$ blocks generated after block height 350,000 and compute P_r.

Recall that to prevent the delivery of a block object O, $\mathcal{A}$ needs to directly relay the *inv* corresponding to O, without validating it. By doing so, $\mathcal{A}$ is faster in relaying the *inv* message than any other node in the system. To better assess the advantage of $\mathcal{A}$ in this case, we measured the time required to validate 100 blocks (from block height 353,000 onwards). Our results show that a single block requires on average 174 milliseconds to validate on an Amazon EC2 dual-core instance; notice that most of the overhead is spent in verifying the correctness of the transactions. In addition to the network latency for retrieving a Bitcoin block (cf. Measure 3), this gives 174 milliseconds advantage on average for $\mathcal{A}$ to succeed in denying the delivery of a block object.

In Table 1, we measure P_r with respect to *(i)* the location of $\mathcal{V}$, *(ii)* $\mathcal{A}$'s number of connections and *(iii)* $\mathcal{V}$'s number of connections (cf. Requirement 1). By gradually increasing the number of connections of $\mathcal{A}$ from 40 to 800, we observe, that $\mathcal{A}$'s success increases with more connections. Namely, our results show that $P_r \approx 0.89$ when $\mathcal{A}$ has 800 connections. The more connections $\mathcal{A}$ maintains, the bigger is $\mathcal{A}$'s likelihood to receive a new block *inv* message before $\mathcal{V}$. Similarly, by gradually increasing the number of connections of $\mathcal{V}$ from 40 to 200, we observe, that the adversary's success decreases when $\mathcal{V}$ maintains more connections. We do not observe a strong correlation between the physical location, and the resulting network latency from $\mathcal{A}$ to $\mathcal{V}$. This implies, that it is crucial for $\mathcal{A}$ to be connected to nodes advertising blocks first.

In summary, our results clearly indicate that an adversary can successfully prevent the delivery of blocks to a particular node with considerable probability.

In another experiment, we measured the amount of time that a particular Bitcoin block can be denied to $\mathcal{V}$ (cf. Requirement 2). Here, when $\mathcal{A}$ forwards a block B successfully as a first node to $\mathcal{V}$, $\mathcal{A}$ actively responds to $\mathcal{V}$'s *ping* requests, but refrains from answering with a *block* message. We repeated this experiment for a number of times and observed that in all cases, $\mathcal{V}$ actively disconnects from $\mathcal{A}$ after exactly 20 minutes.

Notice that the costs borne by $\mathcal{A}$ to deny $\mathcal{V}$ a given block are modest. Namely, $\mathcal{A}$ needs to maintain an active TCP connection with $\mathcal{V}$ and to frequently reply to $\mathcal{V}$'s ping requests. Moreover, $\mathcal{A}$ needs to simply transmit 101 bytes comprising the *inv* message (i.e., 40 bytes for IP and TCP header, and 61 bytes for an *inv* message advertising one block object).

Denying the Delivery of Transactions.

In the following, we investigate the probability of $\mathcal{A}$ in successfully denying the delivery of a transaction. In our experiments, we assume that the transaction T is created by $\mathcal{A}$ (for the reasoning why, refer to Section 5.2), and therefore, $\mathcal{A}$ is guaranteed to be the first node to relay the corresponding *inv* message.

Our evaluation is conducted as follows. We connect only two nodes to $\mathcal{V}$; one node corresponds to a machine controller by $\mathcal{A}$, while the other node $\mathcal{C}$ emulates an honest machine in the Bitcoin network. In our experiments, $\mathcal{A}$ creates an *inv* message for transaction T and forwards the *inv* to $\mathcal{V}$ and $\mathcal{C}$. Upon reception of the *inv* message, $\mathcal{C}$ also forwards the message to $\mathcal{V}$. This captures a realistic transaction announcement in the Bitcoin network. We repeated this experiment 100 times during which $\mathcal{A}$ was able to successfully deny the delivery of T each time for 2 minutes. Namely, for all 100 experiments, $\mathcal{V}$ was always requesting T with a *getdata* message from $\mathcal{A}$, which was not responding to the request. After 120 seconds, $\mathcal{V}$ issued a *getdata* request for T to the other node $\mathcal{C}$.

We then extended this attack such that $\mathcal{A}$ sends T's *inv* message 10 times to $\mathcal{V}$, even before $\mathcal{C}$ issues T's *inv* message. We repeated this experiment 100 times during which $\mathcal{V}$ did not request T from any other peer for a period of 20 minutes. This is consistent with our observations in Section 2. Namely, every node which advertises a transaction object T is inserted in a FIFO buffer and allocated a 2-minute timeout to deliver T. After 2 minutes, T is requested from the next node in the FIFO buffer. By advertising T x-times, the adversary can therefore deny the delivery of T for $2x$ minutes and arbitrarily extend the denial time.

We point out that the communication overhead of this attack basically consists of 101 bytes for each 2 minutes of delay[8].

4. EXTENDING THE BLOCK DELIVERY TIME

We showed in Section 3.2 that $\mathcal{A}$ can deny the delivery of a block object to $\mathcal{V}$ for at least 20 minutes. In this section, we build on our findings and study the possibility of extending the time a block is denied from $\mathcal{V}$. The necessary requirements are that *(i)* $\mathcal{V}$ accepts at least one connection from $\mathcal{A}$, and *(ii)* $\mathcal{A}$ is capable of filling $\mathcal{V}$'s *remaining* open connection slots. For example, if $\mathcal{V}$ maintains 50 connections, $\mathcal{A}$ initiates one connection for advertising block *inv* messages, and fills the remaining $125 - 51 = 74$ open connection slots with default Bitcoin connections. Note that $\mathcal{V}$'s existing 50 connections do not prevent the attack.

In Bitcoin, blocks are transmitted by performing a header-first synchronization. Given the header of a block, any node can find the longest chain and verify the proof of work. Once the headers have been synchronized, Bitcoin nodes can selectively request the corresponding blocks from their peers.

[8]Note that multiple *inv* messages could be embedded into one *inv* message in order to lower the TCP overhead.

Connections of $\mathcal{A}$	40	80	200	800	80	80	
Connections of $\mathcal{V}$	40	40	40	40	80	200	
Victim node	P_r	P_r	P_r	P_r	P_r	P_r	avg. latency $\mathcal{A}$, $\mathcal{V}$ (ms)
Zurich	0.63	0.40	0.97	0.90	0.35	0.29	0.95 ± 0.3
Frankfurt	0.29	0.46	0.61	0.83	0.29	0.34	43.1 ± 0.3
Ireland	0.67	0.47	0.94	0.85	0.24	0.18	28.6 ± 0.5
N.Virginia	0.55	0.92	0.88	0.96	0.34	0.18	91.0 ± 0.0
Oregon	0.38	0.80	0.82	0.90	0.36	0.12	171.0 ± 3.0
N.California	0.36	0.46	0.83	0.96	0.68	0.25	180.0 ± 0.0
Tokyo	0.55	0.86	0.96	0.98	0.22	0.16	246.0 ± 4.9
Singapore	0.40	0.51	0.76	0.92	0.63	0.19	303.0 ± 4.6
Sydney	0.31	0.37	0.60	0.77	0.35	0.21	303.0 ± 4.6
São Paulo	0.29	0.45	0.63	0.83	0.20	0.20	400.0 ± 6.3
Average P_r	0.44 ± 0.14	0.57 ± 0.20	0.80 ± 0.14	0.89 ± 0.07	0.37 ± 0.16	0.21 ± 0.06	176.67 ± 128.3

Table 1: P_r with respect to the number of connections of $\mathcal{A}$ and $\mathcal{V}$. Each experiment is measured over 100 consecutive blocks and across 10 different geographical locations. Each data point of P_r corresponds to the average of 100 measurements; where appropriate, we report the standard deviation (labelled as '$\pm X$'). Note that we exclusively report the number of connections to full Bitcoin nodes.

Headers are typically synchronized on two different occasions:

1. If $\mathcal{V}$ receives a new, previously unknown block *inv* advertisement from a peer $\mathcal{A}$, $\mathcal{V}$ actively requests the block headers from $\mathcal{A}$ with a *getheaders* message.
2. Once a node $\mathcal{A}$ connects to $\mathcal{V}$, both nodes exchange a Bitcoin *version* message. Each version message contains a counter for the most recent block height the respective node is aware of. If $\mathcal{A}$ has a higher block height than $\mathcal{V}$, $\mathcal{V}$ actively requests the block headers from $\mathcal{A}$ with a *getheaders* message.

The header-first synchronization is especially important for the purpose of our analysis, since it allows $\mathcal{V}$ to actively request a block from its peers as soon as it learns about new headers. More specifically, if $\mathcal{A}$ prevents the delivery of block B to $\mathcal{V}$ and $\mathcal{V}$ receives B's header from another peer, then $\mathcal{A}$ can prevent the delivery of B for at most 20 minutes. That is, because after 20 minutes, $\mathcal{V}$ actively disconnects from $\mathcal{A}$ and requests the block B from another peer. Clearly, if the latter peer is not under $\mathcal{A}$'s control, then $\mathcal{V}$ is likely to synchronize with the current main blockchain in the network. If $\mathcal{V}$, however, does not receive the header for block B, $\mathcal{V}$ does not actively request block B from another peer after the 20-minute timeout, even if $\mathcal{V}$ received the block *inv* message from additional peers during the 20-minute timeout.

In order to continuously deny the delivery of block information from $\mathcal{V}$, $\mathcal{A}$ therefore needs to make sure that $\mathcal{V}$ does not receive block headers from the network. This can be achieved when the following two conditions are met.

First relayer for all blocks: $\mathcal{A}$ must be the first node to forward all block *inv* messages to $\mathcal{V}$. For example, when $\mathcal{A}$ starts to deny the delivery of block B from $\mathcal{V}$, and wants to deny the delivery of all blocks up to and including block $B + 5$, $\mathcal{A}$ needs to be the first node relaying all block *inv* messages between and including B and $B + 5$. Note that after a 20-minute timeout, $\mathcal{A}$ is required to resend the corresponding block *inv* message before other peers. Other nodes are unlikely to advertise older blocks, and therefore $\mathcal{A}$ can reliably extend the time a block is denied.

Connection depletion: In the mean time, $\mathcal{V}$ must not receive a new version message from another peer. This can be ensured if $\mathcal{A}$ can keep all *remaining* open connection slots of $\mathcal{V}$ occupied, such that $\mathcal{V}$ does not receive any *version* message from other peers. Notice that existing connections do not exchange *version* messages and, do not impact this condition.

We exemplify the process of denying the delivery of 2 consecutive blocks in Figure 5. Here, we assume that $\mathcal{V}$ successfully receives B's header and appends block B to its local chain. Once block $B + 1$ is found, $\mathcal{A}$ prevents it from being delivered to V. Here, a 20-minute timeout for $\mathcal{A}$ starts for block $B + 1$. Once block $B + 2$ is found, $\mathcal{A}$ also prevents the delivery of this block to $\mathcal{V}$. Here again, a 20-minute timeout for $\mathcal{A}$ starts for block $B + 2$. When the timeout for block $B + 1$ expires, $\mathcal{V}$ does not actively request a block, because it has not learned about any new headers. $\mathcal{V}$ would only receive the headers if $\mathcal{V}$ establishes a new connection to a peer with an up-to-date blockchain, or receives $B + 1$'s *inv* message from another peer after the timeout. $\mathcal{A}$ now, however, can *re-advertise* block $B + 1$ and thus re-activate another 20-minute timeout for block $B + 1$.

Now, assume that when block $B + 3$ is found, $\mathcal{A}$ is not the first node to relay $B + 3$'s *inv* message. $\mathcal{V}$ consequently requests *getheaders* and *getdata* from the advertising peer. Because blocks $B+1$ and $B + 2$, however, are currently being awaited from $\mathcal{A}$, $\mathcal{V}$ only receives block $B + 3$. In order to validate block $B + 3$ in the main blockchain, $\mathcal{V}$ requires the intermediate blocks $B + 1$ and $B + 2$. When the timeout for $B + 2$ expires, $\mathcal{V}$ disconnects from $\mathcal{A}$ and requests block $B + 2$ immediately from another peer. Still, $\mathcal{V}$ is not able to connect $B + 2$ and $B + 3$ to the blockchain, because $B + 1$ is missing. Finally, when the timeout for $B + 1$ expires, $\mathcal{V}$ requests block $B + 1$ from another peer and is able to synchronize with the main blockchain.

Let P_r^x denote the probability that $\mathcal{A}$ can successfully prevent the delivery of x consecutive blocks. Clearly, the delivery of different blocks is an independent process, then $P_r^x = (P_r)^x$, where P_r denotes the average probability of denying the delivery of a single Bitcoin block. We confirm this analysis by means of experiments in the following paragraphs.

Experimental Validation

In what follows, we assess the probability P_r^x that an adversary $\mathcal{A}$ can prevent the delivery of at least x blocks from $\mathcal{V}$.

Our experimental setup is designed as follows. The adversary—in addition to trying to be the first to advertise consecutive blocks—makes sure that $\mathcal{V}$ does not establish new connections by filling all remaining connection slots of $\mathcal{V}$. In our setup, we attempt to continuously prevent the delivery of blocks to 5 different nodes (emulating 5 different $\mathcal{V}$) running the default Bitcoin clients[9]. During our

[9]These nodes were synchronized to the blockchain for almost 72

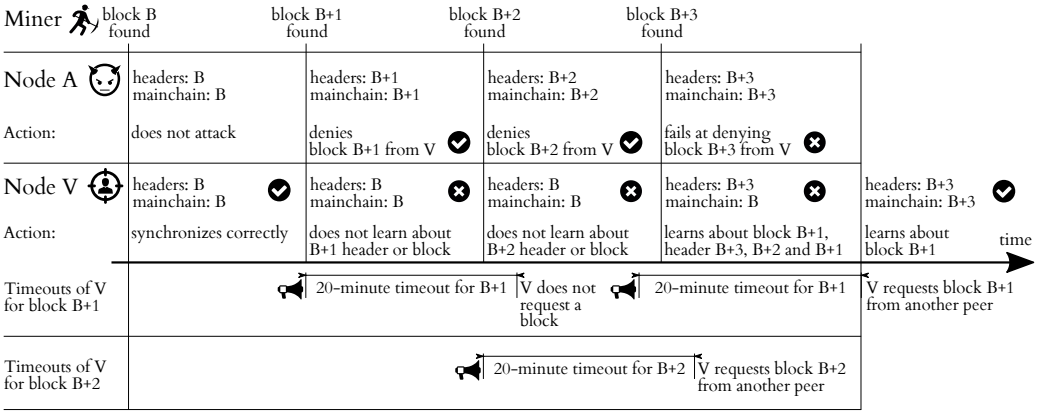

Figure 5: Example process of denying the delivery of multiple blocks. Here, $\mathcal{A}$ succeeds to deny the delivery of 2 consecutive blocks.

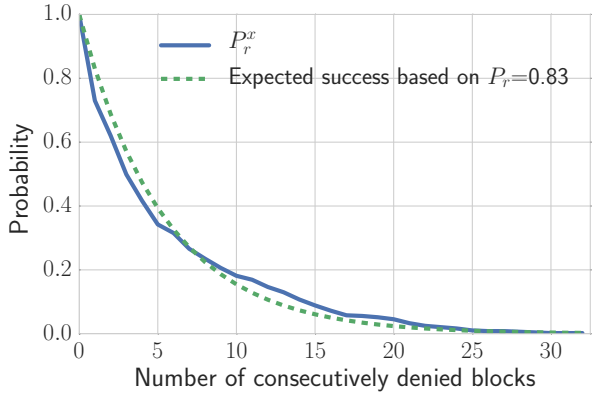

Figure 6: P_r^x w.r.t. the number x of consecutively denied blocks. Here, $\mathcal{A}$ maintains 80 and $\mathcal{V}$ between 11 and 25 connections to full Bitcoin nodes.

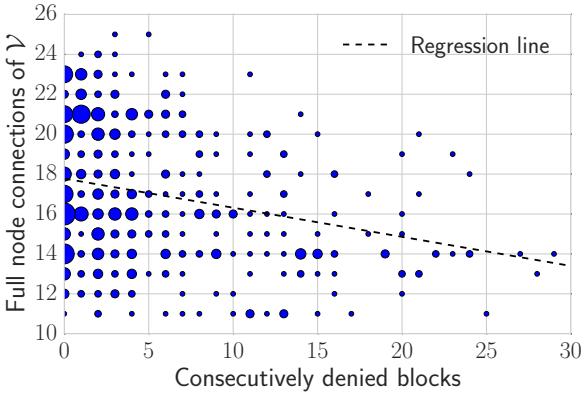

Figure 7: Number of consecutively denied blocks w.r.t. the connections of $\mathcal{V}$. Here, $\mathcal{A}$ maintains 80 full Bitcoin node connections.

study, the nodes' connections to full Bitcoin nodes[10] varied over time (cf. Figure 7). On the other hand, the adversary maintained at all times a fixed number of 80 connections to full Bitcoin clients. Here, all of the machines emulating $\mathcal{V}$ were located in Europe, with a latency below 200ms to $\mathcal{A}$.

In our experiments, the adversary performed a total of 2849 block denying attempts on all 5 nodes; $\mathcal{A}$ was successful in denying the delivery of a total of 2364 blocks—resulting in $P_r = 0.83$. In Figure 6, we measure P_r^x, the probability to prevent the delivery of at least x blocks, as follows. We count the number of consecutive blocks, x, that are only requested (i.e., using a *getheaders* and *getdata*) from the adversary. Here, if we do not receive a *getheaders* and *getdata* message within 30 seconds for a given advertised block or any intermediate re-advertised block, we assume that the victim has requested that block from elsewhere, compute the resulting x, and restart the process. Our results (cf. Figure 6) confirm the analysis in Section 4, indeed show that $P_r^x \approx (P_r)^x$ and indicate that the adversary can succeed in preventing the delivery of consecutive blocks with considerable probability. For instance, the probability to prevent the delivery of 5 consecutive blocks is approximately 0.4.

hours and had 11 to 25 connections to full Bitcoin nodes.

[10]Recall that lightweight Bitcoin clients do not forward blocks or transactions and are therefore not relevant for our study.

In Figure 7, we evaluate the number of consecutively denied blocks with respect to the number of connections of $\mathcal{V}$. Our results confirm our previous observation (cf. Table 1) that the fewer connections $\mathcal{V}$ maintains, the more blocks can be denied. Recall that the more connections a node has (to full nodes), the earlier the node can receive information from the network.

5. IMPLICATIONS

In the previous sections, we thoroughly investigated how Bitcoin's request management system can be abused by a malicious adversary in order to delay information delivery in the Bitcoin network. In what follows, we discuss the impact of our findings on the security of the Bitcoin system.

5.1 Increasing Mining Advantage

In [12], Eyal and Sirer show that a mining pool which controls more than 33% of the computing power in the network can considerably increase its mining advantage (by more than 10%) by withholding its mined blocks until new blocks are found in the network. We show in what follows that the 33% bound of Eyal and Sirer can be even further reduced by leveraging our findings.

Eyal and Sirer's Selfish Mining.

In the selfish mining game (adapted from [12]), a selfish miner

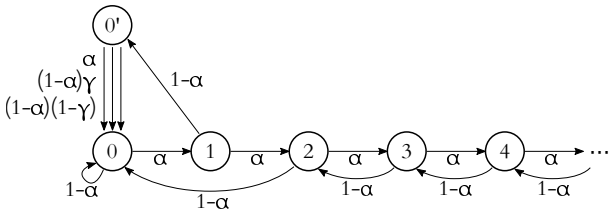

Figure 8: Selfish mining state machine adapted from Eyal and Sirer [12].

does not directly announce its newly mined blocks in the network and instead keeps them secret, until the remaining network finds new blocks. This strategy is captured in the state machine shown in Figure 8.

The state machine depicts the adversary's view on the current state of the selfish mining attack. In state 0, $\mathcal{A}$ and the network have the same view of the currently longest blockchain. $\mathcal{A}$ controls a fraction α of the computing power in the network, and is therefore likely to mine a block with probability α. With probability $1-\alpha$, the network finds and publishes a block, leading to state 0. Once $\mathcal{A}$, however, finds a block, she keeps it secret from the network and state 1 is reached. Moving on, different cases can arise.

First, the network can find a competing block B_N with probability $1-\alpha$, moving the state to $0'$. In this case, $\mathcal{A}$ has an incentive to spread his single secret block $B_\mathcal{A}$ in the network as fast as possible, such that a fraction γ of the network continues mining on $B_\mathcal{A}$. Namely, γ is the fraction of the network that received $B_\mathcal{A}$ before B_N. Subsequently, three cases arise: *(i)* $\mathcal{A}$ finds a block with probability α, *(ii)* the network finds a block by building upon $B_\mathcal{A}$ with probability $(1-\alpha)\gamma$, or *(iii)* the network finds a block building on B_N, with probability $(1-\alpha)(1-\gamma)$. For $\mathcal{A}$, these three cases generate two, one and zero block rewards, respectively.

Second, $\mathcal{A}$ can find a second block and also keep it secret, moving to state 2. Reaching state 2 guarantees $\mathcal{A}$ at least two block rewards. If the network finds a block (transition from $2 \rightarrow 0$ with probability $1-\alpha$), $\mathcal{A}$ can publish the two secret blocks, generate the longest chain and earn two block rewards. $\mathcal{A}$ keeps additional blocks secret moving to states further right and publishes these blocks individually while moving left in the state machine whenever the network finds a block.

Beyond Selfish Mining.

To deter this misbehavior, Eyal and Sirer [12] propose the following countermeasure. When a miner is aware of two competing blocks, the miner should propagate both blocks and select a random block to mine on. This solution does not take into account the case where an adversary can selectively deny miners from receiving a particular block—as confirmed by our results.

Even worse, by preventing the delivery of blocks to a fraction of the network, an adversary can create multiple virtual partitions in the network—each mining on different blocks. As we show below, this grants the adversary with an additional advantage in the selfish mining scenario.

To this end, we extend the state machine from Eyal and Sirer by modeling the fact that $\mathcal{A}$ can deny the delivery of blocks to a fraction of the mining power. The modified state machine is depicted in Figure 9. Let's assume that the adversary currently reached state 3, representing 3 secret blocks. With probability $1-\alpha$ the honest network finds a block $B_{1'}$, resulting in state change to state 2_1. $\mathcal{A}$ manages to deny the delivery of this block to a fraction P of the network, and consequently, only a fraction $1-P$ has learned

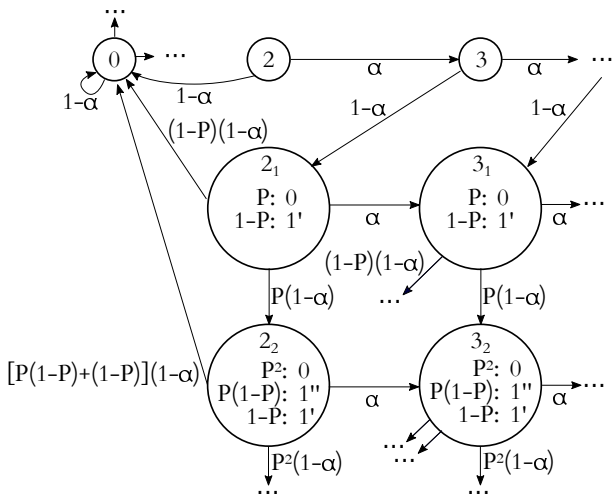

Figure 9: Extending Eyal and Sirer's state machine to capture the case where a selfish miner can deny the delivery of recently mined blocks to a fraction P of the network.

about the new block $B_{1'}$. Three states arise: *(i)* the fraction $1-P$ finds a block (with probability $(1-P)(1-\alpha)$) and $\mathcal{A}$ publishes his remaining 2 secret blocks, resulting in state 0; *(ii)* $\mathcal{A}$ finds a block with probability α (resulting in state 3_1), or *(iii)* the fraction P which has not yet seen block $B_{1'}$ finds with probability $P(1-\alpha)$ a block resulting in state 2_2. All other cases follow the same principle; $\mathcal{A}$ virtually partitions the miners by denying the delivery of new blocks to other honest miners. This strengthens the selfish mining game as it leads to even more wasted computations by the honest miners.

Based on the state machine of Figure 9, we measure the revenue of the adversary in comparison with findings of [12]. For this purpose, we simulate both state machines; in each state, our simulator chooses the next state by generating the probability distribution of the possible transitions according to the given parameters. We performed up to 1,000,000 iterations for both state machines and per data point of Figure 10. Here, we assume that $\mathcal{A}$ is able to deny the delivery of 2 and 5 consecutive blocks with probabilities 0.5 and 0.25, respectively. This assumption conforms with our findings in Section 4. To compute the revenue, we adapt the scheme from [12]; we refer the readers to Appendix A for more details on the adopted revenue estimation.

Our results (cf. Figure 10) show that the 33% bound advertised by Eyal and Sirer can be considerably lowered. For instance, an adversary which succeeds in denying the delivery of 2 consecutive blocks from 50% of the network will profit from selfish mining if he controls 26.5% of the computing power in the network. This adversary effectively controls the Bitcoin mining network given only 35% of the actual computing power. *Even worse, our results show that an adversary (with $P = 0.5$ and 5 consecutively denied blocks) which commands less than 34% of the computing power in the network can effectively sustain the longest block chain and therefore control the entire network.*

5.2 Double-Spending

In what follows, we show how an adversary can leverage our findings to double-spend *(i)* fast payments in which the corresponding transactions have not yet been included in any block of the main blockchain (also referred to as zero-confirmation transactions) [4], and *(ii)* 1-confirmation transactions, i.e., transactions which have

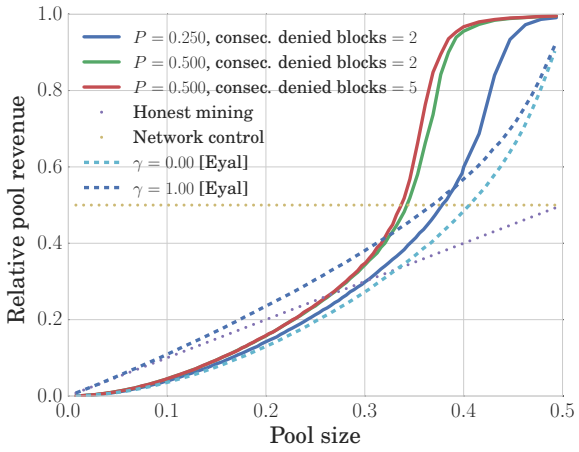

Figure 10: Relative revenue gain of selfish miners with and without the delaying of block information. For our simulations, we assume $\gamma = 0$ and always perform better than [12].

been already confirmed in one Bitcoin block in the main blockchain. Notice that zero-confirmation and one-confirmation transactions are common to handle payments in which the time between exchange of the currency and goods is short.

Here, we assume that the adversary $\mathcal{A}$ creates two transactions T_d and T_l sharing the same input coins. T_d is a transaction whose output addresses are owned by $\mathcal{A}$, while the outputs of T_l are owned by a vendor $\mathcal{V}$. Similar to [4], the goal of $\mathcal{A}$ is to convince $\mathcal{V}$ to accept T_l, acquire service from $\mathcal{V}$, while ensuring that T_d is confirmed by miners and included in the main blockchain.

Double-Spending of Zero-Confirmation Transactions.

Zero-confirmation transactions are essential for the operation of several businesses. Recall that Bitcoin blocks are generated within 10 minutes on average which prevents the daily operation of businesses where the exchange between the currency and the service is short (e.g. in the order of few minutes). In light of the double-spending analysis in [17], an enhanced version of Bitcoin, referred to as Bitcoin XT [3], currently broadcasts the first double-spend transaction in the network. This allows a vendor to observe any double-spending attempt for an upcoming payment, and to consequently deny the processing of a fast payment. By February 2015, almost 16 full nodes in the Bitcoin network had adopted the double-spend relay protection of Bitcoin XT.

Motivated by our findings, we show in what follows that the protection of Bitcoin XT is not effective in preventing double-spending attacks of fast payments. *We also show that an adversary can perform double-spending attacks—without the risk of loosing money.*

To do so, the adversary first sends an *inv* message advertising T_d to $\mathcal{V}$; this prevents $\mathcal{V}$ from receiving T_d from the network—in spite of the protection embedded in Bitcoin XT. $\mathcal{A}$ subsequently broadcasts T_d in the network, and sends T_l to $\mathcal{V}$ shortly after (cf. Figure 11). From the viewpoint of $\mathcal{V}$, T_l does not conflict with any other transaction, and therefore the trade can be concluded. Although other nodes in the network can observe the two conflicting transactions, T_d and T_l, and issue the corresponding warnings, $\mathcal{V}$ cannot receive T_d from its neighbors, and is therefore unable to detect this attack. Shortly after, the first observed transaction in the network (i.e. T_d) is typically included in the blockchain.

$\mathcal{A}$ can further prevent nodes in the network from detecting the

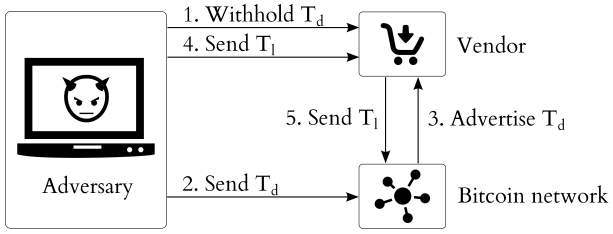

Figure 11: Circumventing the double-spend relay protection of Bitcoin XT.

attack, as follows. Similar to before, $\mathcal{A}$ issues T_d in the network, while denying its delivery to $\mathcal{V}$ (e.g. for a 20 minute period). As soon as T_d is confirmed in a block B_d, $\mathcal{A}$ directly prevents the delivery of B_d to $\mathcal{V}$ by sending the latter an *inv* advertising B_d (cf. Section 3.2). This ensures that $\mathcal{V}$ does not receive B_d from the network for at least 20 minutes. In the meantime, $\mathcal{A}$ sends T_l to $\mathcal{V}$ which will broadcast it in the network. Because Bitcoin XT, however, only relays conflicting transactions from the memory pool, a pool of not yet confirmed transactions, T_l is not considered double-spent. T_l, nonetheless, is considered a conflicting transaction w.r.t. the already confirmed T_d, and therefore is not relayed in the network. In this way, $\mathcal{A}$ does not bear any risk of loosing her money since T_d was already included in a block and will likely be accepted by most nodes in the network.

We empirically confirm our analysis using a testbed comprising of three Bitcoin nodes emulating a vendor $\mathcal{V}$, the adversary $\mathcal{A}$ and a Bitcoin XT node[11]. All three nodes maintain a direct TCP connection in order to exchange Bitcoin transactions. We performed the aforementioned double-spending attack a number of times; our results confirm that although the Bitcoin XT node attempts to broadcast T_d, $\mathcal{V}$ does not receive T_d. This clearly shows that the protection of Bitcoin XT cannot deter double-spending.

Double-Spending of 1-Confirmation Transactions.

Notice that if $\mathcal{A}$ is connected directly to a (honest) miner $\mathcal{M}$, then $\mathcal{A}$ can also attempt to double-spend a transaction T_l which has already been confirmed by one block B_l in the blockchain. In this case, the only means for $\mathcal{A}$ to double-spend T_l would be to include the double-spending transaction T_d in a fork of the blockchain. Recall that forks frequently occur in the Bitcoin network [10], and are resolved automatically by choosing the longest fork chain. If the blockchain fork, which contains T_d, eventually emerges as the longest fork, T_d will be accepted by the majority of the peers in the network and double-spending is successful, since $\mathcal{V}$ can no longer redeem T_l.

To double-spend T_l, $\mathcal{A}$ connects directly to both $\mathcal{V}$ and $\mathcal{M}$ in order to prevent the delivery of T_d to both prior to broadcasting it in the network. As mentioned earlier, this ensures that $\mathcal{V}$ and $\mathcal{M}$ do not receive T_d, but the rest of the network receives and mine for blocks which confirm T_d. $\mathcal{A}$ then sends T_l to $\mathcal{M}$, with the aim that $\mathcal{M}$ confirms T_l in a block.

With some probability, the Bitcoin network eventually includes T_d in block B_d. Because B_d contains the double-spend transaction T_d, $\mathcal{A}$ has to again prevent B_d from being received by $\mathcal{V}$ and $\mathcal{M}$. Later on, if $\mathcal{M}$ finds and broadcasts a block B_l, then B_l will be accepted by $\mathcal{V}$ since the latter does not receive any conflicting transaction T_d nor conflicting block B_d. This process is summarized in Figure 12. Here, we assume that $\mathcal{M}$ does not control a

[11]We employ the same setup as visualized in Figure 11, but replace the Bitcoin network with the Bitcoin XT node.

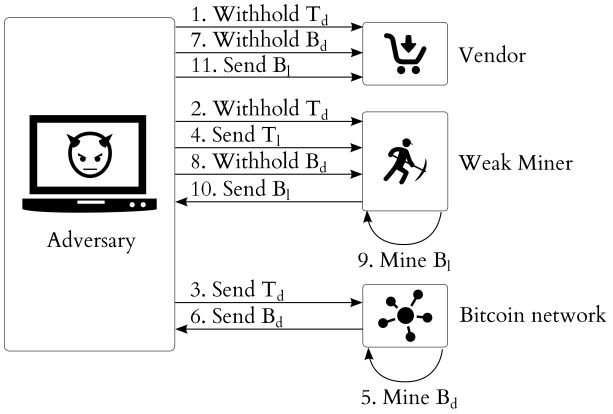

Figure 12: The adversary $\mathcal{A}$ performs double-spending of a *1-confirmation* transaction against a vendor, by leveraging the computing power of a weak miner. T_d corresponds to the double-spending, and T_l to the legitimate transaction.

large fraction of the computing power—otherwise B_l will be likely included in the main blockchain. Notice that $\mathcal{A}$ can collude with $\mathcal{M}$ (or can mine for blocks by herself) to further increase the success probability of the attack.

5.3 Denial of Service

Delaying information propagation in a P2P-based crypto-currency network such as Bitcoin can be very damaging as it results in the delay of processing payments. Our findings clearly suggest that e.g., Denial of Service (DoS) attacks on Bitcoin can be made more easily realizable by exploiting the object request management of Bitcoin. That is, an adversary that controls a number of nodes in the network (and establishes connections with many Bitcoin nodes) can effectively prevent the dissemination of information, such as blocks and transactions, in the entire network.

Notice that there are approximately 6000 reachable Bitcoin nodes in the network[12]. A resource-constrained adversary can simply connect to all these nodes and deny them the delivery of transactions and blocks.

Given an average block generation time g, and a timeout of 20 minutes per block (cf. Section 3.2), we estimate the number n of required *inv* messages to prevent the delivery of consecutive blocks during time t:

$$n(t) = \sum_{i=1}^{\left\lceil \frac{t}{g} \right\rceil - 1} \left\lceil \frac{t - g \cdot i}{20} \right\rceil \quad (1)$$

In addition, $\mathcal{A}$ needs to fill and maintain the open connection slots of $\mathcal{V}$. To prevent the delivery of blocks to all 6000 reachable full Bitcoin nodes, $\mathcal{A}$ needs to maintain approximately 450,000 active Bitcoin TCP connections[13] and transmit about $101 \cdot 6000 \approx$ 600KB of *inv* messages per block. During time t, every node requires moreover $n(t)$ *inv* messages. Assuming that $\mathcal{A}$ can operate 10000 connections per node[14], $\mathcal{A}$ requires approximately 45 nodes for attempting to constantly prevent block information propagation in the Bitcoin network.

Notice that by denying the delivery of blocks from full Bitcoin nodes, $\mathcal{A}$ implicitly prevents the reception of these blocks by the various lightweight Bitcoin clients[15].

This analysis complements the work of [16] and shows that $\mathcal{A}$ can deny the delivery of blocks from the entire network using almost 45 nodes, and by transmitting only 600 KB per denied block for every 20 minute delay.

6. COUNTERMEASURES

Based on our findings, we discuss and explore possible avenues for enhancing the security of Bitcoin without deteriorating its scalability.

Dynamic Timeouts.

As described in Measure 3, Bitcoin relies on static timeouts in order to tolerate network delays. This implicitly assumes that all nodes and resources in the network are homogeneous—which is clearly not the case. As we show in Figure 3, slow nodes require considerable time to download blocks, while fast nodes can secure the download of blocks in few seconds.

We believe that dynamic timeouts would suit better the heterogeneity of resources in the Bitcoin network. For this purpose, we suggest the inclusion of the size of the message at advertisement time, which would allow each node to dynamically estimate the timeout value according to its resources, and the object size. For instance, when sending block advertisements, we suggest that the miner includes the block size into the block header[16]—which would allow receiving nodes to know the total block size and to appropriately estimate a dynamic timeout for any given block. By doing so, we argue that the advantage of an adversary in abusing timeouts to delay block delivery can be considerably reduced.

We observe that the current timeouts employed by Bitcoin only capture the time starting from the *getdata* advertisement until the full data reception. Here, a timeout between the data request and the beginning of the data transfer could additionally be considered. We argue that such an approach would increase the costs of the adversary in delaying the delivery of information.

Updating Block Advertisements.

Based on our observations, we recommend updating the current block request system as follows:

No *inv* messages: We suggest to drop the advertisement of *inv* messages for blocks, and solely advertise the block headers before transmitting the blocks. By doing so, every receiver can immediately verify the correctness of the PoW, and learn about any new discovered blocks in the network. As mentioned in Section 4, this will ensure that an honest node will always learn about new blocks in the network even if the adversary fills all of its remaining connections to deny the delivery of such information. Notice that each block header is 80 bytes, while a block's *inv* message occupies 36 bytes.

[12] Available from: `https://getaddr.bitnodes.io/`

[13] Assuming all 6000 Bitcoin nodes accept no more than 125 connections and maintain on average 50 connections. Clearly, some Bitcoin nodes accept significantly more than 125 connections (1500 or more).

[14] To limit the amount of information received from these 10,000 nodes, $\mathcal{A}$ can outsource a Bloom filter (similar to existing lightweight clients [15]) which match a small fraction of transactions in the system. By doing so, $\mathcal{A}$ reserves his bandwidth to perform his DoS attack.

[15] Lightweight Bitcoin clients (e.g. SPV nodes) do not validate nor maintain the full blockchain, and therefore get their information from a subset of the reachable Bitcoin nodes. Users typically prefer operating lightweight clients, because they require significantly less processing power and disk storage.

[16] Block headers contain an unused field which was intended for the number of transactions confirmed in the block.

Therefore, we do not expect a considerable increase in the communication overhead due to this modification.

Keep track of block advertisers: Similar to transaction advertisements, we suggest that Bitcoin nodes keep track of the block headers' advertisers. This recommendation goes hand in hand with the advertisement based on block headers since it allows the node to request the blocks from the peers announcing the longest chain. Additionally, this allows the node to request the block from (a randomly chosen) advertising peer in case the chosen relay delays the delivery of the block.

Handling Transaction Advertisements.

As mentioned in Measure 4, a transaction is currently requested from the peers that advertised it first. If a given peer does not respond within an appropriate timeout, the transaction will be requested from the next peer stored in the FIFO queue. As mentioned in Section 3.2, this gives considerable advantage for the adversary to prolong the timeout before the receiver requests the transaction from any other peer.

To remedy this, we suggest the following hardening measures:

Filtering by IP address: One way to deter against such an adversary would be to accept only one *inv* of the same transaction per IP address. Notice that this cannot entirely prevent the adversary from advertising the same transaction using different IP addresses.

Randomly choosing sender: Another complementary approach would be to randomly choose an incrementing number of peers to contact from the list of advertising peers if the first peer did not answer to the *getdata* request. Here, a transaction is first queried from the first advertising peer. If this peer however does not transmit the transaction within the specified timeout, the transaction is requested from two randomly chosen peers simultaneously, then from three peers, until the transaction is finally received. This will limit the advantage of an adversary which tries to advertise first the same transaction several times.

Given these suggestions, the probability to receive a transaction after the n's timeout, when the first advertising peer is controlled by $\mathcal{A}$, given i_a *inv* messages sent by $\mathcal{A}$ and a total of i_t *inv* messages is computed as follows:

$$P_t(n) = \sum_{i=0}^{n}[\prod_{j=1}^{i}(\frac{\binom{i_a - \frac{j(j+1)}{2}}{j+1}}{\binom{i_t - \frac{j(j+1)}{2}}{j+1}}) \cdot (1 - \frac{\binom{i_a - \frac{(i+1)(i+2)}{2}}{i+2}}{\binom{i_t - \frac{(i+1)(i+2)}{2}}{i+2}})] \quad (2)$$

Equation 2 allows one to compute the probability $P_t(n)$ that a fast payment in Bitcoin is secure after a waiting time of nt, where t is the timeout set by the node when handling transactions. Figure 13 depicts $P_t(n)$ with respect to the waiting time. Here, we assume that $i_t = 125$,[17] $t = 30$ seconds and that the adversary controls a fraction $\frac{i_a}{i_t}$ of the advertised *inv* messages. Our results show, that after 5 minutes waiting time, the node would be almost certain to receive a transaction, even if the adversary controls 95% of the advertised *inv* messages (in return about 6 out of 125 *inv* messages originate from honest peers). Notice that the receiving peer can be alerted if a sudden increase in *inv* advertisements occurs.

We conclude, that fast payments should only be accepted after a waiting time nt to ensure with probability $P_t(n)$ that the transaction cannot be double-spent—even if the double-spend first-relay protection is implemented in the network. For a transaction request timeout of $t = 30$ seconds, this waiting time amounts to almost 5 minutes— which is half the duration of the block generation time.

[17]Recall that the default maximum of neighbors per node is 125.

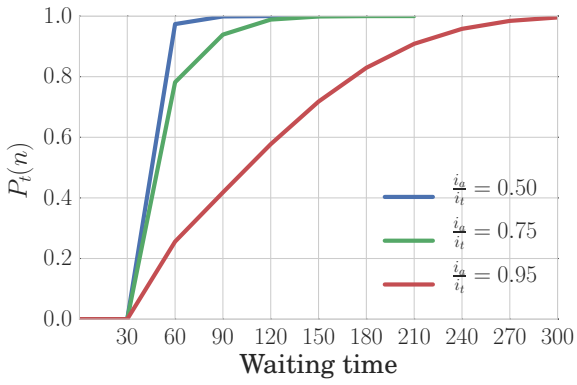

Figure 13: Waiting time vs $\frac{i_a}{i_t}$. Here, $t = 30$ seconds and $i_t = 125$.

Penalizing Non-responding Nodes.

We also suggest extending the penalty system employed by Bitcoin (cf. Measure 1) to penalize non-responding peers. Namely, nodes which constantly delay information delivery after a *getdata* message should be penalized, and when appropriate disconnected from. Here, a careful design of the appropriate penalty is needed in order not to penalize slow nodes in the network.

7. RELATED WORK

The security and privacy of Bitcoin has received considerable attention from the research community. Bonneau *et al.* [8] provide a comprehensive exposition of the second generation cryptocurrencies, including Bitcoin and the many alternatives that have been implemented as alternate protocols.

In [10], Decker and Wattenhofer investigate transaction and block propagation time in Bitcoin. Gervais *et al.* analyze in [14] the limits of decentralization in Bitcoin, and show that the vital operations and decisions that Bitcoin is undertaking are not decentralized.

Finney [13] describe a double-spending attack in Bitcoin where the attacker includes in her generated blocks transactions that transfer some coins between her own addresses; these blocks are only released in the network after the attacker double-spends the same coins using fast payments and acquires a given service. Barber *et al.* [6] analyze possible ways to enhance the resilience of Bitcoin against a number of security threats. Eyal and Sirer [12] show that Bitcoin is not safe against attacks by selfish miners that command more than 33% of the total computing power of the network. Similarly, Courtois and Bahack [9] study subversive mining strategies where miners withhold their recently mined blocks in order to increase their advantage in the network. Karame *et al.* [17] investigate double-spending attacks in Bitcoin and show that double-spending fast payments in Bitcoin can be performed in spite of the measures recommended by Bitcoin developers. The authors also proposed a countermeasure which is currently integrated in Bitcoin.

All these studies, however, assume that the information disseminated in the Bitcoin network is directly received by nodes. In this paper, we show that an adversary can effectively prevent the delivery of such information from the network peers for a considerable amount of time. Our work share similarities with [16]. Here,

Heilman *et al.* showed that by monopolizing the connections of nodes in the system, an adversary can perform selfish mining, and abuse Bitcoin's consensus protocol. Unlike [16], our work however shows that a resource-constrained attacker can achieve considerable damage in the network using only a handful of connections. By doing so, our results suggest that the attacks outlined in [9, 12, 16, 17] can be even more aggravated (cf. Section 5).

8. CONCLUDING REMARKS

In this paper, we showed that the current scalability measures adopted by Bitcoin come at odds with the security of the system. More specifically, we showed that an adversary can exploit these measures in order to effectively delay the propagation of transactions and blocks to specific nodes—without causing a network partitioning in the system.

We analyzed the implication of our findings and showed that these threats enable an adversary to easily mount Denial-of-Service attacks on the entire network by preventing the delivery of blocks in the system. Moreover, mining pools can exploit this vulnerability to claim a higher mining advantage in the network. When combined with the results of Eyer and Sirer [12], our findings therefore suggest that selfish mining pools which command less than 33% of the computing power can considerably increase their mining advantage. Finally, our findings show that the countermeasure adopted in Bitcoin XT to prevent the double-spending of fast payments can be easily circumvented by a resource-constrained adversary.

Based on our findings, we explored a number of countermeasures in order to enhance the security of Bitcoin without deteriorating its scalability. We therefore hope that our findings solicit more research towards the re-design of the request management system of Bitcoin.

9. REFERENCES

[1] Bitcoin market cap, 2015. Available from: `https://blockchain.info/charts/market-cap`.

[2] Bitcoin Scalability, 2015. Available from: `https://en.bitcoin.it/wiki/Scalability`.

[3] Bitcoin XT, 2015. Available from: `https://github.com/bitcoinxt/bitcoinxt`.

[4] Double spending in Bitcoin, 2015. Available from: `https://medium.com/@octskyward/double-spending-in-bitcoin-be0f1d1e8008`.

[5] Stress Test Prepares VisaNet for the Most Wonderful Time of the Year, 2015. Available from: `http://www.visa.com/blogarchives/us/2013/10/10/stress-test-prepares-visanet-for-the-most-wonderful-time-of-the-year/index.html`.

[6] S. Barber, X. Boyen, E. Shi, and E. Uzun. Bitter to Better - How to Make Bitcoin a Better Currency. In *Proceedings of Financial Cryptography and Data Security*, 2012.

[7] Trade - Bitcoin, 2013. Available from: `https://en.bitcoin.it/wiki/Trade`.

[8] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, May 2015.

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

[10] C. Decker and R. Wattenhofer. Information Propagation in the Bitcoin Network. In *13-th IEEE International Conference on Peer-to-Peer Computing*, 2013.

[11] Bitcoin exchanges, 2013. Available from: `https://en.bitcoin.it/wiki/Exchanges`.

[12] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[13] The Finney Attack, 2013. Available from: `https://en.bitcoin.it/wiki/Weaknesses#The_.22Finney.22_attack`.

[14] Arthur Gervais, Ghassan Karame, Srdjan Capkun, and Vedran Capkun. Is bitcoin a decentralized currency? In *IEEE Security and Privacy*, 2014.

[15] Arthur Gervais, Ghassan O. Karame, Damian Gruber, and Srdjan Capkun. On the privacy provisions of bloom filters in lightweight bitcoin clients. In *Proceedings of the 30th Annual Computer Security Applications Conference, ACSAC 2014, New Orleans, LA, USA, December 8-12, 2014*, 2014.

[16] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. 2015.

[17] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, New York, NY, USA, 2012. ACM.

APPENDIX

A. REVENUE FOR SELFISH MINING

In what follows, we detail Eyal and Sirer's revenue scheme.

1. **Current state:** Any state, except two branches of length 1.
   Event: The pool finds a block and adds it to its secret chain.
   Reward: Block reward is determined later.
2. **Current state:** Two branches of length 1.
   Event: Pool finds a block.
   Reward: Pool publishes two blocks, revenue of two blocks.
3. **Current state:** Two branches of length 1.
   Event: Other find block on previous pool block.
   Reward: Pool and other obtain revenue of one block each.
4. **Current state:** Two branches of length 1.
   Event: Other find block on previous other block.
   Reward: Other obtain revenue of two blocks.
5. **Current state:** No secret block.
   Event: Other find block.
   Reward: Other obtain revenue of one block.
6. **Current state:** One secret block.
   Event: Other find block. Pool publishes secret block.
   Reward: Block reward is determined later, depends on γ.
7. **Current state:** Leading two secret blocks.
   Event: Other find block, only 1 secret left.
   Reward: Pool publishes secret blocks, two blocks revenue.
8. **Current state:** Leading more than two secret blocks.
   Event: Other find block, only 1 secret left.
   Reward: Pool publishes one block, revenue of one block.