Double-Spending Fast Payments in Bitcoin

Ghassan O. Karame
NEC Laboratories Europe
Heidelberg, Germany
ghassan.karame@neclab.eu

Elli Androulaki
Dept. of Computer Science
ETH Zurich, Switzerland
elli.androulaki@inf.ethz.ch

Srdjan Čapkun
Dept. of Computer Science
ETH Zurich, Switzerland
capkuns@inf.ethz.ch

ABSTRACT

Bitcoin is a decentralized payment system that relies on Proof-of-Work (PoW) to verify payments. Nowadays, Bitcoin is increasingly used in a number of fast payment scenarios, where the time between the exchange of currency and goods is short (in the order of few seconds). While the Bitcoin payment verification scheme is designed to prevent double-spending, our results show that the system requires tens of minutes to verify a transaction and is therefore inappropriate for fast payments. An example of this use of Bitcoin was recently reported in the media: Bitcoins were used as a form of *fast* payment in a local fast-food restaurant.

Until now, the security of fast Bitcoin payments has not been studied. In this paper, we analyze the security of using Bitcoin for fast payments. We show that, unless appropriate detection techniques are integrated in the current Bitcoin implementation, double-spending attacks on fast payments succeed with overwhelming probability and can be mounted at low cost. We further show that the measures recommended by Bitcoin developers for the use of Bitcoin in fast payments are not always effective in detecting double-spending; we show that if those recommendations are integrated in future Bitcoin implementations, double-spending attacks on Bitcoin will still be possible. Finally, we propose and implement a modification to the existing Bitcoin implementation that ensures the detection of double-spending attacks against fast payments.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General – Security and protection.

General Terms

Security, Measurement, Experimentation.

Keywords

Bitcoin, Double-spending, Fast Payments, Countermeasures.

1. INTRODUCTION

First introduced in 2008, Bitcoin [22] is an emerging digital currency that has, as of September 2011, approximately 60,000 users [1]. Bitcoin is currently integrated across a number of businesses [2] and has several exchange markets (e.g., [3]). It is also foreseen that Bitcoin ATMs will be deployed in locations around the globe [4] in order to bridge the gap between digital currency and cash.

Bitcoin is a Proof-of-Work (PoW) based currency that allows users to generate digital coins by performing computations. Users execute payments by digitally signing their transactions and are prevented from double-spending their coins (i.e., signing-over the same coin to two different users) through a distributed time-stamping service [22]. This service operates on top of the Bitcoin Peer-to-Peer (P2P) network that ensures that all transactions and their order of execution are available to all Bitcoin users.

Nowadays, Bitcoin is increasingly used in a number of "fast payment" scenarios, where the exchange time between the currency and goods is short. Examples include vending machine payments and fast-food payments (recently featured in media reports on Bitcoin [5]), where the payment is followed by fast (< 30 seconds) delivery of goods. While the Bitcoin PoW-based time-stamping mechanism is essential for the detection of double-spending attacks (i.e, in which an adversary attempts to use some of her coins for two or more payments), it requires tens of minutes to verify a transaction and is therefore inappropriate for fast payments. Since Bitcoin users are anonymous and users (are encouraged to) hold many accounts, there is only limited value in verifying the payment after the user has obtained the goods (and e.g., left the store) or services (e.g., access to on-line content). The developers of Bitcoin implicitly acknowledge the problem of verifying fast payments and inform users that they do not need to wait for the payment to be verified as long as the transaction has been released in the network [6]; this, however, does not solve this problem but merely limits the damage as the system still remains vulnerable to double-spending attacks.

Until now, double-spending attacks on fast payments in Bitcoin or mechanisms for their immediate detection and/or prevention have not been studied. In this work, we analyze double-spending attacks on fast Bitcoin payments in detail and we show that, unless appropriate detection techniques are integrated in current Bitcoin clients, double-spending attacks on fast payments succeed with overwhelming probability and can be mounted against current Bitcoin clients at low cost. We further show that the detection measures recommended by Bitcoin developers are not always effective in detecting

Hash: 000000000043a8c0fd1d6f726790caa2a406010d19efd2780db27bdbbd93baf6
Previous block: 00000000001937917bd2caba204bb1aa530ec1de9d0f6736e5d85d96da9c8bba
Next block: 00000000000036312a44ab7711afa46f475913fbd9727cf508ed4af3bc933d16
Time: 2010-09-16 05:03:47
Difficulty: 712.884864
Transactions: 2
Merkle root: 8fb300e3fdb6f30a4c67233b997f99fdd518b968b9a3fd65857bfe78b2600719
Nonce: 1462756097

Input/Previous Output	Source & Amount	Recipient & Amount
N/A	Generation: 50 + 0 total fees	Generation: 50 + 0 total fees
f5d8ee39a430...:0	1JBSCVF6VM6QjFZyTnbpLjoCJ...: 50	16ro3Jptwo4asSevZnsRX6vf..: 50

Table 1: Example Block of Bitcoin. The block contains 2 transactions, one of which awards the generator peer with 50 BTCs.

double-spending; we argue that even if those recommendations are followed[1], double-spending attacks on Bitcoin are still possible.

Given the increasing use of Bitcoin, the reliance on fast Bitcoin payments is only expected to increase. We therefore argue that the integration of effective techniques to detect double-spending attacks against fast payments emerges as a necessity since these attacks might severely impact the trustworthiness and economic standing of Bitcoin. Leveraging our findings, we propose and implement a modification to the current Bitcoin implementation that ensures the detection of double-spending attacks against fast payments. Our modified client implementation is available for download at [7].

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

- We measure and analyze the time required to confirm transactions in Bitcoin. Our analysis shows that transaction confirmation in Bitcoin can be modeled with a shifted geometric distribution with an average transaction confirmation time of 10 minutes and a standard deviation of approximately 15 minutes. We argue that this hinders the reliance on transaction confirmation when dealing with fast payment scenarios.
- We thoroughly analyze the conditions for performing successful double-spending attacks against fast payments in Bitcoin. We then present the first realization of double-spending attacks on fast payments in Bitcoin using a handful of hosts located around the globe[2].
- We explore and evaluate empirically a number of solutions for detecting double-spending attacks against fast payments in Bitcoin. We show that the recommendations of Bitcoin developers on how to detect double-spending in fast payment scenarios are not always effective. Leveraging our results, we propose and implement a modification to the existing Bitcoin implementation to enable the detection of double-spending attacks.

The remainder of the paper is organized as follows. In Section 2, we briefly describe Bitcoin. In Section 3, we review how Bitcoin payments are processed. In Section 4, we analyze and evaluate the security of fast payments with existing Bitcoin clients. We then evaluate the security of possible measures to detect double-spending against fast payments in Bitcoin. In Section 6, we overview related work and we conclude the paper in Section 7.

[1]These techniques are still not integrated in the current Bitcoin implementation.

[2]In our experiments, we solely used Bitcoin wallets and accounts that we own; other Bitcoin users were not affected by our experiments.

2. BACKGROUND ON BITCOIN

Bitcoin is a decentralized P2P payment system [22] that was introduced in 2008. Electronic payments are performed by generating *transactions* that transfer Bitcoin coins (BTCs) among Bitcoin peers. These peers are referenced in each transaction by means of virtual pseudonyms—referred to as *Bitcoin addresses*. Generally, each peer has hundreds of different Bitcoin addresses that are all stored and managed by its (digital) wallet. Each address is mapped through a transformation function to a unique public/private key pair. These keys are used to transfer the ownership of BTCs among addresses.

Peers transfer coins to each other by issuing a transaction. A 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 [22]. Any peer can verify the authenticity of a BTC by checking the chain of signatures.

Transactions are included in Bitcoin *blocks* that are broadcasted in the entire network. To prevent double-spending of the same BTC, Bitcoin relies on a hash-based PoW scheme. More specifically, to generate a block, Bitcoin peers must find a nonce value that, when hashed with additional fields (i.e., the Merkle hash of all valid and received transactions, the hash of the previous block, and a timestamp), the result is below a given target value. If such a nonce is found, peers then include it (as well as the additional fields) in a new block thus allowing any entity to publicly verify the PoW. Upon successfully generating a block, a peer is typically granted 50 new BTCs. This provides an incentive for peers to continuously support Bitcoin. The resulting block is forwarded to all peers in the network, who can then check its correctness by verifying the hash computation. If the block is deemed to be "valid"[3], then the peers append it to their previously accepted blocks. Since each block links to the previously generated block, the Bitcoin block *chain* grows upon the generation of a new block in the network. As an example, Table 1 depicts the information included in Bitcoin block number 80,000 as reported in the Bitcoin block explorer [8].

The main intuition behind Bitcoin is that for peers to double-spend a given BTC, they would have to replace the transaction where the BTC was spent and the corresponding block where it appeared in, otherwise their misbehavior would be detected immediately. This means that for malicious peers to double-spend a BTC without being detected, they would not only have to redo all the work required to compute the block where that BTC was spent, but also recompute all the subsequent blocks in the chain. This

[3]That is, the block contains correctly formed transactions that have not been previously spent, and has a correct PoW.

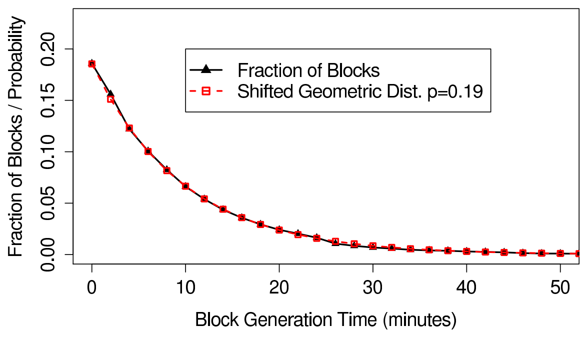

Figure 1: Block generation times in Bitcoin. Assuming a (time) bin size of 2 minutes, the block generation function can be fitted to a shifted geometric distribution with $p = 0.19$. Refer to Appendix A for further details.

ensures that the Bitcoin network can counter such misbehavior as long as the fraction of honest peers in the network exceeds that of malicious colluding peers [22].

Further details on Bitcoin can be found in [9, 10, 22]. In what follows, we provide a summary (adapted from [22]) of the steps that peers undergo in Bitcoin when a payment occurs.

- New transactions are broadcasted by peers in the network.

- When a new transaction is received by a peer, it checks whether the transaction is correctly formed, and whether the BTCs have been previously spent in a block in the block chain. If the transaction is correct, it is stored locally in the *memory pool* of peers until it is included in a valid block. In the paper, we refer to a transaction that appears in the memory pools of peers by a zero-confirmation transaction.

- Peers work on constructing a new block. If they find a nonce that solves the PoW, they include all the transactions that appear in their memory pool within the newly-formed block. Peers then broadcast the block in the network. Transaction that are included in well-formed blocks are called "confirmed transactions". As the block chain grows, it becomes hard to modify/double-spend confirmed Bitcoin transactions[4].

- When peers receive a new block, they verify that the block hash is valid and that every transaction included within the block has not been previously spent. If the block verification is successful, peers continue working towards constructing a new block using the hash of the last accepted block in the "previous block" field (cf. Table 1).

3. PAYMENT VERIFICATION IN BITCOIN

Bitcoin is currently being used in both slow payments (e.g., delivery by post) and fast payment scenarios (i.e., where the exchange between currencies and services happens simultaneously). In this section, we review and analyze how Bitcoin transactions are processed in these two scenarios.

[4]In current Bitcoin clients, a transaction has to receive 6 confirmations before it is added to the user's wallet.

3.1 Slow Payment Verification—Transaction Confirmation

As described in Section 2, the most conventional and secure way for a vendor $\mathcal{V}$ to accept a payment made by a customer $\mathcal{C}$ is to wait until the transaction issued from $\mathcal{C}$ to $\mathcal{V}$ is confirmed in at least one block before offering service to $\mathcal{C}$. Note that the Bitcoin client can inform $\mathcal{V}$ whether its transactions have been confirmed or not. Since confirmed transactions are likely to be accepted by honest peers in the Bitcoin network, a malicious client $\mathcal{A}$ has negligible advantage in tricking $\mathcal{V}$ to accept incorrect or double-spent transactions.

Transaction Confirmation Time.

In what follows, we briefly analyze the time it takes for a given transaction to be confirmed in Bitcoin.

Bitcoin is designed so that blocks are generated every 10 minutes, on average. For that purpose, the difficulty of the work required to construct a block is adjusted dynamically depending on the time it took to solve the previous blocks. More specifically, Bitcoin requires that the hash of the block to be constructed is below a given target value. This target is interpolated from the overall time it took to solve the previous 2016 blocks [11]. Since the fields required to construct the hash of a block also change with time (e.g., timestamp, previous block hash), the probability to successfully construct a valid block in Bitcoin is almost constant with respect to the number of trials [12].

To measure the generation time of existing Bitcoin blocks, we created a Python script that parses the block chain of Bitcoin and extracts the time intervals between the generation of consecutive blocks. Our findings show that while the average block generation time is approximately 10 minutes (9 minutes and 54 seconds), the standard deviation of the measurements is about 881.24 seconds which corresponds to almost 15 minutes[5]. Thus, there is a considerable variability among the block generation times. In Figure 1, we depict the distribution of the generation times of the extracted blocks. As explained in Appendix A, this distribution can be modeled by a shifted geometric distribution with success probability 0.19. Our results also show that only 64% of the blocks were generated in less than 10 minutes. The remaining 36% of the blocks require between 10 and 40 minutes to be generated.

3.2 Fast Payment Verification—Transaction Reception

Our analysis in Section 3.1 shows that the time required to confirm transactions impedes the operation of many businesses that are characterized by a fast-service time (i.e., when the exchange of currency and goods is shorter than a minute). As such, it is clear that vendors, such as supermarkets, vending machines, take-away stores [13], Bitcoin Point of Sale terminals [14], etc., cannot rely on transaction confirmation when accepting Bitcoin payments.

Given that a considerable number of businesses require fast processing of payments, Bitcoin developers encourage vendors to accept fast Bitcoin payments with zero-confirmations as soon as the vendor receives a transaction from the network transferring the correct amount of BTCs to one of its addresses [6, 13]. We point out that this is the only available option for vendors to currently "verify" fast payments. Note that the vendor can search in his wallet for the client's transaction to verify its reception. The main intuition here is that this constitutes sufficient proof that the transaction was indeed broadcasted in the network. We emphasize that it typically

[5]The maximum time of block generation was 25 hours and 8 minutes.

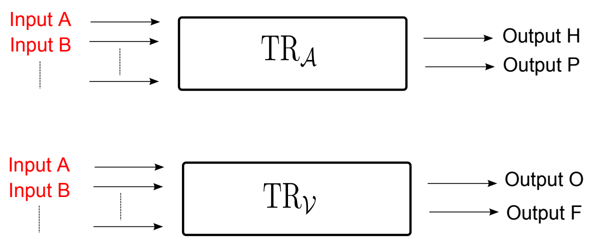

Figure 2: Example construction of $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$. Here, we show two different transactions with the same inputs and different outputs. $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ can share a subset of their inputs—in which case $\mathcal{A}$ only tries to double-spend a subset of the BTCs that she pays with.

takes few seconds (< 3 seconds) for a transaction to propagate between two Bitcoin peers.

4. SECURITY ANALYSIS OF FAST BITCOIN PAYMENTS

In what follows, we analyze the security of fast payments in Bitcoin. Namely, we focus on the analysis and implementation of double-spending attacks on fast Bitcoin payments.

4.1 Attacker Model

Our system consists of a malicious client $\mathcal{A}$, and a vendor $\mathcal{V}$, connected through a Bitcoin network. We assume that $\mathcal{A}$ is motivated to acquire a service from $\mathcal{V}$ without having to spend its BTCs. More specifically, $\mathcal{A}$ could try to double-spend the coin she already transferred to $\mathcal{V}$. By double-spending, we refer to the case where $\mathcal{A}$ can redeem and use the same coins with which she payed $\mathcal{V}$ so as to acquire a different service elsewhere; where appropriate, we also discuss the impact of the n-time spending of the same coin, where $n > 2$, to acquire multiple services.

We assume that $\mathcal{A}$ knows the Bitcoin and IP addresses of $\mathcal{V}$[6]. We further assume that $\mathcal{A}$ can only control few peers in the network (that she can deploy since Bitcoin does not restrict membership) and does not have access to $\mathcal{V}$'s keys or machine. The remaining peers in the network are assumed to be honest and to correctly follow the Bitcoin protocol. We point out here that the computing power harnessed by $\mathcal{A}$ and its helpers does not exceed the aggregated computer power of all the honest peers in the network. This prevents $\mathcal{A}$ from inserting/confirming incorrect blocks in the Bitcoin block chain. In this paper, we consider the (worst-case) scenario where $\mathcal{A}$ does not participate in the block generation process. This also suggests that when a transaction is confirmed in a block, this transaction cannot be modified by $\mathcal{A}$.

Conforming with the operation of Bitcoin, we assume that $\mathcal{A}$ generates a new Bitcoin address whenever it communicates with $\mathcal{V}$. We also assume that the set of addresses used by $\mathcal{A}$ are insufficient to identify $\mathcal{A}$. This suggests that the identity of $\mathcal{A}$ is unlikely to be revealed even if her misbehavior may be detected some time after she has acquired the service.

4.2 Necessary Conditions for Successful Double-Spending

To perform a successful double-spending attack, the attacker $\mathcal{A}$ needs to trick the vendor $\mathcal{V}$ into accepting a transaction $\mathrm{TR}_{\mathcal{V}}$ that $\mathcal{V}$ will not be able to redeem subsequently.

[6]When issuing a transaction, a Bitcoin client could either specify a recipient Bitcoin address or an IP address.

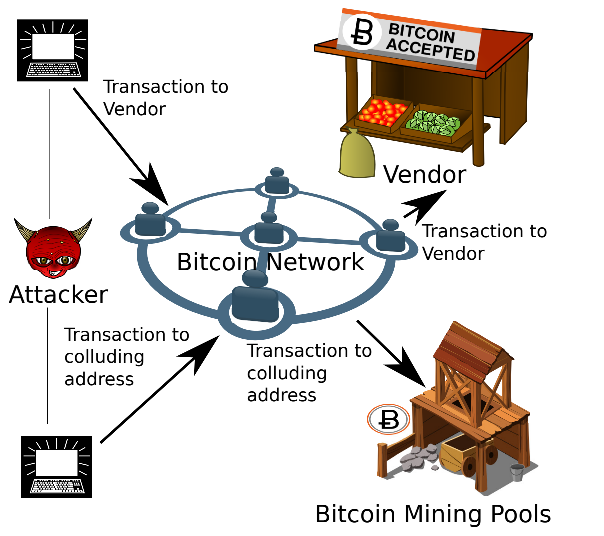

Figure 3: Sketch of a double-spending attack against fast payments in Bitcoin. Here, the attacker $\mathcal{A}$ dispatches two transactions that use the same BTCs in the Bitcoin network. The double-spending attack is deemed successful if the BTCs that $\mathcal{A}$ used to pay for $\mathcal{V}$ cannot be redeemed (i.e., when the second transaction is included in the upcoming Bitcoin block).

In this case, $\mathcal{A}$ creates another transaction $\mathrm{TR}_{\mathcal{A}}$ that has the same inputs as $\mathrm{TR}_{\mathcal{V}}$ (i.e., $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ use the same BTCs) but replaces the recipient address of $\mathrm{TR}_{\mathcal{V}}$—the address of $\mathcal{V}$— with a recipient address that is under the control of $\mathcal{A}$.[7]

An example construction of $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ is depicted in Figure 2. If both transactions are sent adjacently in time, they are likely to have similar chances of getting confirmed in an upcoming block. This is the case since Bitcoin peers will not accept multiple transactions that share common inputs; they will only accept the version of the transaction that reaches them first which they will consider for inclusion in their generated blocks and they will ignore all remaining versions. Given this, a double-spending attack can succeed if $\mathcal{V}$ receives $\mathrm{TR}_{\mathcal{V}}$, and the majority of the peers in the network receive $\mathrm{TR}_{\mathcal{A}}$ so that $\mathrm{TR}_{\mathcal{A}}$ is more likely to be included in a subsequent block. This process is sketched in Figure 3.

Let $t_i^{\mathcal{V}}$ and $t_i^{\mathcal{A}}$ denote the times at which node i receives $\mathrm{TR}_{\mathcal{V}}$ and $\mathrm{TR}_{\mathcal{A}}$, respectively. As such, $t_{\mathcal{V}}^{\mathcal{V}}$ and $t_{\mathcal{V}}^{\mathcal{A}}$ denote the respective times at which $\mathcal{V}$ receives $\mathrm{TR}_{\mathcal{V}}$ and $\mathrm{TR}_{\mathcal{A}}$.

Given this, we outline the necessary conditions for $\mathcal{A}$'s success in mounting a double-spending attack.

Requirement 1 — $\mathrm{TR}_{\mathcal{V}}$ is added to the wallet of $\mathcal{V}$.

If $\mathrm{TR}_{\mathcal{V}}$ is not added to the memory pool of $\mathcal{V}$, then $\mathcal{V}$ cannot check that $\mathrm{TR}_{\mathcal{V}}$ was indeed broadcasted in the network. Note that for $\mathrm{TR}_{\mathcal{V}}$ to be included in $\mathcal{V}$'s wallet, then $t_{\mathcal{V}}^{\mathcal{V}} < t_{\mathcal{V}}^{\mathcal{A}}$; otherwise, $\mathcal{V}$ will first add $\mathrm{TR}_{\mathcal{A}}$ to its memory pool and will reject $\mathrm{TR}_{\mathcal{V}}$ as it arrives later. After waiting for few seconds without having $\mathrm{TR}_{\mathcal{V}}$ in its wallet, $\mathcal{V}$ can ask $\mathcal{A}$ to re-issue a new payment.

Requirement 2 — $\mathrm{TR}_{\mathcal{A}}$ is confirmed in the block chain.

If $\mathrm{TR}_{\mathcal{V}}$ is confirmed first in the block chain, $\mathrm{TR}_{\mathcal{A}}$ can never appear in subsequent blocks. That is, $\mathcal{V}$ will not have its BTCs

[7]Note that our analysis is not restricted to the case where the recipient address is controlled by $\mathcal{A}$ and applies to other scenarios, where the recipient is another vendor.

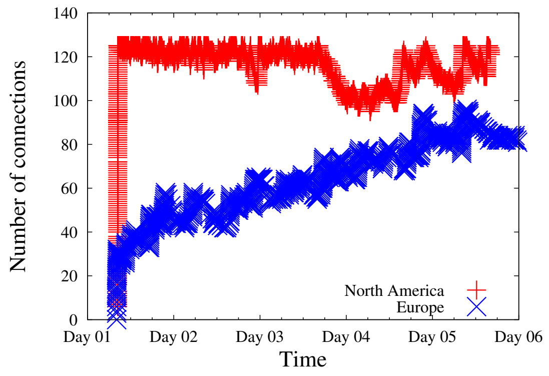

Figure 4: Number of connections that two Bitcoin nodes (in North America and in Europe, respectively) witnessed over the period of 6 consecutive days. Since the connectivity of peers in Bitcoin varies with time, $\mathcal{A}$ has considerable opportunities to establish at least one direct connection with $\mathcal{V}$.

back. In this case, $\mathcal{V}$ will receive its BTCs, and can redeem them at its convenience. Recall that the goal of $\mathcal{A}$ is to acquire a service offered by $\mathcal{V}$ without having to spend her BTCs.

Requirement 3 — $\mathcal{V}$'s service time is smaller than the time it takes $\mathcal{V}$ to detect misbehavior.

Since Bitcoin users are anonymous and users hold many accounts, there is only limited value in $\mathcal{V}$ detecting misbehavior after the user has obtained the service (and e.g., left the store). This is the case since the Bitcoin address of $\mathcal{A}$ is insufficient to identify her. As such, for $\mathcal{V}$ to successfully detect any misbehavior by $\mathcal{A}$, the detection time must be smaller than the service time.

4.3 Mounting Double-Spending Attacks in Bitcoin

In this section, we discuss how $\mathcal{A}$ can satisfy Requirements **(1)**, **(2)**, and **(3)**.

Satisfying Requirement 1.

$\mathcal{A}$ connects directly to $\mathcal{V}$ in the Bitcoin P2P network[8]. Given the Bitcoin protocol specification, $\mathcal{V}$ will always accept the connection requests by other peers as long as the maximum number of its current inbound connections has not been reached. By default, this number is set to 125.[9] As shown in Figure 4, the connectivity of Bitcoin peers is heavily dependent on the churn of the Bitcoin network (i.e., peers departing/joining); this gives a considerable number of opportunities for $\mathcal{A}$ to establish a direct connection with $\mathcal{V}$ (e.g., $\mathcal{A}$ could try to connect with $\mathcal{V}$ over the weekend or in the evening, when the number of connections of $\mathcal{V}$ drops below the maximum).

In the sequel, we assume that $\mathcal{A}$ has access to one or more helpers, denoted by $\mathcal{H}$. $\mathcal{A}$ and $\mathcal{H}$ do not necessarily have to be on physically disjoint machines (e.g., $\mathcal{H}$ could run as a thread/process on the same machine as $\mathcal{A}$). We further assume that *(i)* $\mathcal{A}$ and $\mathcal{H}$ communicate using a low-latency confidential channel (e.g., by exchanging encrypted messages using a direct TCP connection) and that *(ii)* $\mathcal{H}$ never connects directly to $\mathcal{V}$ in the Bitcoin P2P network.

$\mathcal{A}$ sends $\mathrm{TR}_\mathcal{V}$ to $\mathcal{V}$ at time $\tau_\mathcal{V}$ and $\mathrm{TR}_\mathcal{A}$ to $\mathcal{H}$ at time $\tau_\mathcal{A}$, such that $\tau_\mathcal{A} = \tau_\mathcal{V} + \Delta t$. $\mathcal{V}$ and $\mathcal{H}$ relay the transactions that they received from $\mathcal{A}$ in the network. Let $\delta t^{\mathcal{A}}_{\mathcal{VH}}$ refer to the time it takes $\mathrm{TR}_\mathcal{A}$ to propagate in the Bitcoin P2P network from $\mathcal{V}$ to $\mathcal{H}$ and $\delta t^{\mathcal{V}}_{\mathcal{AV}}$ denote the time it takes $\mathrm{TR}_\mathcal{V}$ to reach $\mathcal{V}$. In this case, $t^{\mathcal{A}}_{\mathcal{V}} - t^{\mathcal{V}}_{\mathcal{V}}$ can be estimated as follows:

$$t^{\mathcal{A}}_{\mathcal{V}} - t^{\mathcal{V}}_{\mathcal{V}} \approx \tau_\mathcal{A} + \delta t^{\mathcal{A}}_{\mathcal{VH}} - (\tau_\mathcal{V} + \delta t^{\mathcal{V}}_{\mathcal{AV}}) \quad (1)$$
$$\approx \Delta t + \delta t^{\mathcal{A}}_{\mathcal{VH}} - \delta t^{\mathcal{V}}_{\mathcal{AV}}. \quad (2)$$

Note that since $\mathcal{H}$ is never a neighbor of $\mathcal{V}$, there is at least one hop on the path between $\mathcal{H}$ and $\mathcal{V}$. Since $\mathcal{A}$ is an immediate neighbor of $\mathcal{V}$ and assuming no congestion at network paths, then $\delta t^{\mathcal{A}}_{\mathcal{VH}} \geq \delta t^{\mathcal{V}}_{\mathcal{AV}}$. In this case, $t^{\mathcal{V}}_{\mathcal{V}} < t^{\mathcal{A}}_{\mathcal{V}}$ for reasonably chosen Δt (e.g., $\Delta t \geq 0$), thus satisfying Requirement **(1)**.

Satisfying Requirement 2.

Since $\mathcal{H}$ and $\mathcal{V}$ are highly likely to have different neighbors, the broadcasted transactions are likely to spread in the network till the point where either *(i)* all Bitcoin peers accept in their memory pools $\mathrm{TR}_\mathcal{V}$ or $\mathrm{TR}_\mathcal{A}$ or *(ii)* either $\mathrm{TR}_\mathcal{V}$ or $\mathrm{TR}_\mathcal{A}$ gets confirmed in a block.

In what follows, we estimate the probability that $\mathrm{TR}_\mathcal{A}$ is confirmed in a block first. In our analysis, we denote by t_0 the time at which both transactions $\mathrm{TR}_\mathcal{A}$ and $\mathrm{TR}_\mathcal{V}$ first coexist in the network[10], and we assume that no block containing either one of them has been generated till that time. We argue that this is a realistic assumption given that $\mathrm{TR}_\mathcal{A}$ and $\mathrm{TR}_\mathcal{V}$ need to be typically broadcasted back to back given a small delay (in the order of few seconds); it is therefore unlikely that one of them is confirmed within the first few seconds in a new block. In the experiments in Section 4.4, we relax this assumption and we evaluate the general case where either $\mathrm{TR}_\mathcal{A}$ and $\mathrm{TR}_\mathcal{V}$ can be confirmed immediately when they are broadcasted in the network.

We divide time into equal intervals of size δt, such that, the probability of successful block generation in each δt can be modeled as a Bernoulli trial with success probability $\eta \cdot p$, where η is the number of peers that work towards block generation and p the success probability of a peer in generating a block within δt (for the reasoning why, refer to Appendix A).

Let $t_k = k \cdot \delta t + t_0$ and $\eta^k_\mathcal{V}$ and $\eta^k_\mathcal{A}$ denote the number of Bitcoin peers that have received $\mathrm{TR}_\mathcal{V}$ and $\mathrm{TR}_\mathcal{A}$ respectively until time t_k. Each Bitcoin node will only add to its memory pool the transaction it receives first among $\mathrm{TR}_\mathcal{V}$ and $\mathrm{TR}_\mathcal{A}$. Since only the transactions that appear in the memory pool of peers are eligible to be confirmed in subsequent blocks, the probability that $\mathrm{TR}_\mathcal{V}$ is included in a block within time interval $]t_k,\ t_{k+1}]$ is: $\mathrm{Pr}^k_\mathcal{V} = \eta^k_\mathcal{V} \cdot p$. Similarly, for $\mathrm{TR}_\mathcal{A}$, the corresponding probability is $\mathrm{Pr}^k_\mathcal{A} = \eta^k_\mathcal{A} \cdot p$. Thus, the probability $\mathsf{p}_\mathcal{V}(k)$ that a block containing $\mathrm{TR}_\mathcal{V}$ is generated within the time interval $]t_k,\ t_{k+1}]$ is given by:

$$\mathsf{p}_\mathcal{V}(k) = \mathrm{Pr}^k_\mathcal{V} \cdot \prod_{i=0}^{k-1}(1 - \mathrm{Pr}^i_\mathcal{V}) = \eta^k_\mathcal{V} p \cdot \prod_{i=0}^{k-1}(1 - \eta^i_\mathcal{V} p).$$

Similarly, the probability that a block containing $\mathrm{TR}_\mathcal{A}$ is generated at the same time interval is given by:

$$\mathsf{p}_\mathcal{A}(k) = \mathrm{Pr}^k_\mathcal{A} \cdot \prod_{i=0}^{k-1}(1 - \mathrm{Pr}^i_\mathcal{A}) = \eta^k_\mathcal{A} p \cdot \prod_{i=0}^{k-1}(1 - \eta^i_\mathcal{A} p).$$

If at time $t_s = s \cdot \delta t + t_0$ every node in the network has received

[8]Recall that the IP address of $\mathcal{V}$ is public.

[9]Note that the maximum number of connections can be modified using the "-maxconnections" command [15].

[10]This does not necessarily mean that $\mathrm{TR}_\mathcal{A}$ and $\mathrm{TR}_\mathcal{V}$ are broadcasted at the same time.

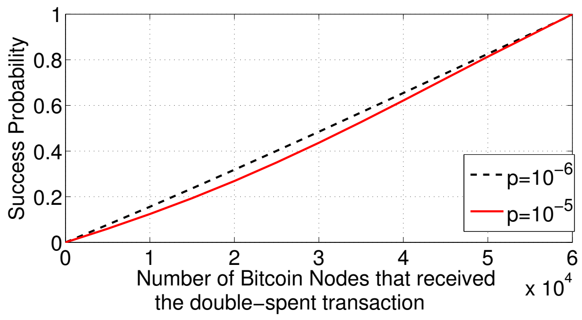

Figure 5: $\mathrm{P_S}^{(2)}$ with respect to various values of $\eta_{\mathcal{A}}^{k}$ and $\eta_{\mathcal{V}}^{k}$. Here, $p = 10^{-6}$, $\delta t = 10$ seconds, $t_0 = 0$, $t_s = \delta t$ and the number of peers in the network is 60000.

at least one of the transactions $\mathrm{TR}_{\mathcal{V}}$ or $\mathrm{TR}_{\mathcal{A}}$, the following holds:

$$\eta_{\mathcal{A}}^{k} \leq \eta_{\mathcal{A}}^{k+1} \text{ and } \eta_{\mathcal{V}}^{k} \leq \eta_{\mathcal{V}}^{k+1}, \text{if } k < s$$

$$\eta_{\mathcal{A}}^{k} = \eta_{\mathcal{A}}^{k+1} = \eta_{\mathcal{A}}^{s} \text{ and } \eta_{\mathcal{V}}^{k} = \eta_{\mathcal{V}}^{k+1} = \eta_{\mathcal{V}}^{s}, \text{otherwise.}$$

This suggests that $\forall i \geq s$, $\eta_{\mathcal{V}}^{i} + \eta_{\mathcal{A}}^{i} = \eta_{\mathcal{V}}^{s} + \eta_{\mathcal{A}}^{s}$.

To compute the probability of success of the double-spending attack, we make the assumption that, $\forall k$, $\eta_{\mathcal{V}}^{k}$ and $\eta_{\mathcal{A}}^{k}$ do not exchange their newly constructed blocks; in this way, the time $t_{g_{\mathcal{V}}}$ required by peers that are mining in favor of $\mathrm{TR}_{\mathcal{V}}$ to generate a new block is independent of that required by the peers that are mining in favor of $\mathrm{TR}_{\mathcal{A}}$, $t_{g_{\mathcal{A}}}$. Given this, the probability that Requirement **(2)** is satisfied, $\mathrm{P_S}^{(2)}$, is computed as follows:

$$\mathrm{P_S}^{(2)} = \mathrm{Prob}(t_{g_{\mathcal{A}}} < t_{g_{\mathcal{V}}}) + \frac{1}{2}\mathrm{Prob}(t_{g_{\mathcal{A}}} = t_{g_{\mathcal{V}}}). \qquad (3)$$

That is, $\mathrm{P_S}^{(2)}$ is composed of two components; one corresponds to the event that the block containing $\mathrm{TR}_{\mathcal{A}}$ is first generated and the second to the event where the blocks containing $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ are generated at the same time, i.e., $t_{g_{\mathcal{A}}} = t_{g_{\mathcal{V}}}$. In the latter case, the probability that the block containing $\mathrm{TR}_{\mathcal{A}}$ is eventually adopted by the Bitcoin peers is 0.5. Here, $\mathrm{Prob}(t_{g_{\mathcal{A}}} < t_{g_{\mathcal{V}}})$ and $\mathrm{Prob}(t_{g_{\mathcal{A}}} = t_{g_{\mathcal{V}}})$ are computed as follows.

$\mathrm{Prob}(t_{g_{\mathcal{A}}} < t_{g_{\mathcal{V}}}) = \sum_{g_{\mathcal{A}}=0}^{\infty} \mathsf{p}_{\mathcal{A}}(g_{\mathcal{A}}) \cdot \mathsf{p}_{\mathcal{V}}(g_{\mathcal{V}} > g_{\mathcal{A}} | g_{\mathcal{A}})$

$= \eta_{\mathcal{A}}^{0} p(1 - \eta_{\mathcal{V}}^{0} p) +$

$$\sum_{g_{\mathcal{A}}=0}^{\infty} \eta_{\mathcal{A}}^{g_{\mathcal{A}}} p \cdot (1 - \eta_{\mathcal{V}}^{g_{\mathcal{A}}} p) \cdot \prod_{j=0}^{g_{\mathcal{A}}-1} (1 - \eta_{\mathcal{V}}^{j} p)(1 - \eta_{\mathcal{A}}^{j} p).$$

$\mathrm{Prob}(t_{g_{\mathcal{A}}} = t_{g_{\mathcal{V}}}) =$

$$\sum_{g_{\mathcal{A}}=1}^{\infty} p^2 \eta_{\mathcal{V}}^{g_{\mathcal{A}}} \eta_{\mathcal{A}}^{g_{\mathcal{A}}} \cdot \prod_{j=0}^{g_{\mathcal{A}}-1} (1 - \eta_{\mathcal{V}}^{j} p)(1 - \eta_{\mathcal{A}}^{j} p).$$

In Figure 4.3, we depict $\mathrm{P_S}^{(2)}$ for various values of $\eta_{\mathcal{V}}^{k}$, $\eta_{\mathcal{A}}^{k}$ and p when $\delta t = 10$ seconds, $t_s = \delta t$ and the number of peers in the network is 60000.

Our analysis therefore shows that $\mathcal{A}$ can maximize $\mathrm{P_S}^{(2)}$ by increasing the number of peers that receive $\mathrm{TR}_{\mathcal{A}}$, $\eta_{\mathcal{A}}^{k}, \forall t_k$. $\mathcal{A}$ can achieve this: *(i)* by sending $\mathrm{TR}_{\mathcal{A}}$ before $\mathrm{TR}_{\mathcal{V}}$ and therefore giving $\mathrm{TR}_{\mathcal{A}}$ a better advantage in spreading in the network and/or *(ii)* by relying on multiple helpers to spread $\mathrm{TR}_{\mathcal{A}}$ faster in the network. In the former case, $\mathcal{A}$ can delay the transmission of $\mathrm{TR}_{\mathcal{V}}$ by a maximum of $\Delta t = \delta t_{\mathcal{VH}}^{\mathcal{A}} - \delta t_{\mathcal{AV}}^{\mathcal{V}}$ (cf. Equation 1) after sending $\mathrm{TR}_{\mathcal{A}}$ while ensuring that $\mathcal{V}$ first receives $\mathrm{TR}_{\mathcal{V}}$. In this way, both Requirements **(1)** and **(2)** can be satisfied.

Location	Number of Hosts
Europe	4
North America	3
Asia Pacific	2
South America	1

Table 2: Geographic location of the Bitcoin nodes that we used in our measurements. We made use of 10 Bitcoin nodes located around the globe. All the hosts were running Ubuntu 11.10 with at least 613 MB of RAM.

Satisfying Requirement 3.

With the exception of the verification of transaction reception (Requirement **(1)**), current Bitcoin clients do not employ any technique to detect double-spending attacks against fast payments. Recall that since the service time is much smaller than the time to confirm a transaction, the vendor cannot verify that $\mathrm{TR}_{\mathcal{V}}$ has been included in a recent Bitcoin block before giving away it service.

Moreover, even if $\mathcal{V}$'s client receives both $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ before it offers its service to $\mathcal{A}$, no message/warning is propagated to $\mathcal{V}$ in the current Bitcoin implementation. We point that the Bitcoin daemon locally generates an error if it receives a transaction whose inputs have already been spent. However, this error is not displayed to the Bitcoin user. Therefore, Requirement **(3)** is always satisfied in current Bitcoin clients (up to version 0.5.2) if Requirement **(1)** has already been satisfied.

It is interesting to note that a Bitcoin node located at `http://blockchain.info/` keeps track of all transactions sent by Bitcoin peers in an attempt to identify malicious addresses (e.g., that perform double-spending attacks). Note that this information is not propagated to peers in the network. Since $\mathcal{A}$ always uses newly generated addresses (that cannot be linked to each other by default), then she is unlikely to bear any consequences even if some of her addresses were labelled "malicious".

4.4 Experimental Evaluation

We now present the experimental results of double-spending experiments in the Bitcoin network. Our experiments aim at investigating the satisfiability of the aforementioned Requirements **(1)**, **(2)** and **(3)**. Recall that, in current Bitcoin clients, Requirement **(3)** is satisfied provided that Requirement **(1)** was satisfied. We denote by $\mathrm{P_S}$, the probability that the attack succeeds. In Section 5, we extend our analysis and experiments and we evaluate the effectiveness of possible detection techniques to alleviate double-spending.

Experimental Setup.

We adopt the setup described in Section 4.3 in which the attacker $\mathcal{A}$ is equipped with one or more helper nodes $\mathcal{H}$ that help her relay the double-spent transaction. In our experiments, we made use of 10 Bitcoin nodes located around the globe (Table 2); this serves to better assess the different views seen from multiple points in the Bitcoin overlay network and to abstract away the peculiarities that might originate from specific network topologies. Appendix **??** includes the Bitcoin addresses that we used while performing our experiments.

To perform the attack, we modified the C++ implementation of Bitcoin client version 0.5.2. Conforming with our analysis in Section 4.3, our new client ensures that:

- The attacker only connects to the vendor's machine. This connection is accepted by $\mathcal{V}$ if it has fewer than 125 connections. If the connection is refused, $\mathcal{A}$ can wait until a neighbor of $\mathcal{V}$ disconnects before attempting to connect again.

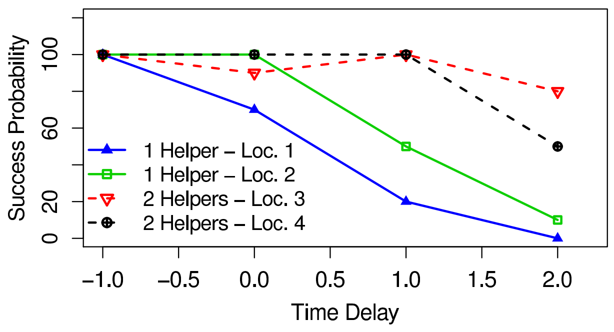

Figure 6: $\mathrm{P_S}$ **versus** Δt **when** $\mathcal{V}$ **has 8 connections.**

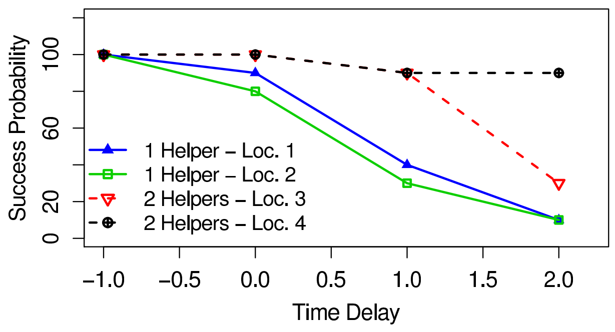

Figure 7: $\mathrm{P_S}$ **versus** Δt **when** $\mathcal{V}$ **has 40 connections.**

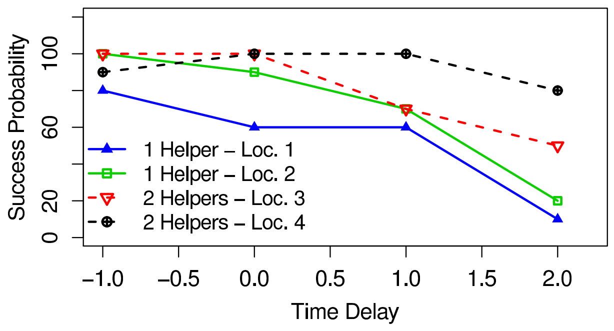

Figure 8: $\mathrm{P_S}$ **versus** Δt **when** $\mathcal{V}$ **has 125 connections.**

Location	**# Helpers**	Δt (sec)	$\mathrm{P_S}$
Asia Pacific 1, 125 connections	2	0	100%
Asia Pacific 2, 125 connections	2	0	100%
North America 1, 8 connections	1	0	100%
North America 2, 40 connections	1	0	90%
Asia Pacific 1, 8 connections	2	1	100%
Asia Pacific 2, 125 connections	2	1	100%
North America 1, 40 connections	1	-1	100%

Figure 9: Summary of Results. Here, "Location" denotes the location of $\mathcal{V}$**, "connections" denote the number of** $\mathcal{V}$**'s connections.**

- The attacker creates transactions $\mathrm{TR}_{\mathcal{V}}$ and $\mathrm{TR}_{\mathcal{A}}$ constructed using the same coins. She sends $\mathrm{TR}_{\mathcal{V}}$ using the Bitcoin network to the neighboring vendor and $\mathrm{TR}_{\mathcal{A}}$ via a direct TCP connection to one or more helper nodes with an initial delay Δt of -1, 0, 1, and 2 seconds. Here, Δt refers to the time delay between the transmission of $\mathrm{TR}_{\mathcal{V}}$ and $\mathrm{TR}_{\mathcal{A}}$ by $\mathcal{A}$.

- Upon reception of $\mathrm{TR}_{\mathcal{A}}$, each helper node broadcasts it in the Bitcoin network.

With this setup, we performed double-spending attempts when the vendors are located in 4 different network locations (2 vendors were in North America and the remaining 2 were in Asia Pacific). In our experiments, $\mathcal{A}$ was located in Europe. However, since $\mathcal{A}$ does not contribute in spreading in the Bitcoin network any transaction herself, her location does not affect the outcome of the attack. That is, the sole role of $\mathcal{A}$ is to send $\mathrm{TR}_{\mathcal{V}}$ to $\mathcal{V}$ using a direct connection in the Bitcoin network and $\mathrm{TR}_{\mathcal{A}}$ to the helper nodes using a direct TCP connection.

We conducted our experiments with a varying number of connections of the vendor (8, 40 and 125 connections) and by varying the number of helper nodes (1 and 2) chosen randomly from the nodes in Table 2. The helper nodes were connected to at least 125 other Bitcoin peers. Each data point in our measurements corresponds to 10 different measurements, totaling approximately 500 double-spending attempts with a total of 20 BTCs. We point out that since all the hosts in our measurements were using wallets that were under our control, other Bitcoin users were not affected by our measurements. We also created a Python script that, for each conducted measurement, parses the generated logs along with the Bitcoin block explorer [16] to check whether Requirement **(2)** is satisfied.

Results.

To assess the feasibility of double-spending in fast Bitcoin payments, we evaluate empirically the success probability, $\mathrm{P_S}$, with respect to the number of helper nodes, the number of connections of the vendor and Δt.

Our experimental results, depicted in Figures 6, 7, 8, and 9 show that, irrespective of a specific network topology, the probability that $\mathcal{A}$ succeeds in mounting double-spending attacks is significant.

Confirming our previous analysis, $\mathrm{P_S}$ decreases as Δt increases. As explained in Section 4.3, this is due to the fact that the higher is Δt, the larger is the number of peers that receive $\mathrm{TR}_{\mathcal{V}}$; in turn, the probability that $\mathrm{TR}_{\mathcal{A}}$ is confirmed before $\mathrm{TR}_{\mathcal{V}}$ decreases. As shown in Figures 6, 7, and 8, this can be remedied if the number of helper nodes that spread $\mathrm{TR}_{\mathcal{A}}$ increases. Our results show that even for a large Δt of 2 seconds, relying on 2 helper nodes still guarantees that double-spending succeeds with a considerable probability; when $\Delta t = 1$ seconds, the attack is guaranteed to succeed ($\mathrm{P_S}$ approaches 1) using 2 helpers. This is summarized in Figure 9. Generalizing these results, it is clear that $\mathcal{A}$ succeeds, with high probability, in spending the same coin to $n \geq 1$ different recipients as long as the number of helpers that assist $\mathcal{A}$ in spreading $\mathrm{TR}_{\mathcal{A}}$ is greater or equal to n. As we show in Section 5, the larger is n, the higher is the probability that $\mathcal{A}$'s misbehavior is detected.

The number of $\mathcal{V}$'s connections considerably affects $\mathrm{P_S}$ especially when $\mathcal{A}$ controls only one helper; in the case where $\mathcal{V}$ has a similar number of connections when compared to the number of connections of the helper, $\mathrm{P_S}$ approaches 0.5. This corresponds to the case where both $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ are spread equally in the network (Figure 8). On the other hand, as the connectivity of $\mathcal{V}$ decreases, $\mathrm{TR}_{\mathcal{A}}$ spreads faster in the network. This is depicted in

Figures 6 and 7. We thoroughly investigate the impact of the connectivity of $\mathcal{V}$ in Section 5.1.

Our experiments show that double-spending attacks against fast payments succeed with overwhelming probability in the current Bitcoin versions and do not incur significant cost on the attacker. Since Bitcoin users are anonymous (i.e., only the Bitcoin addresses are revealed to vendors), clients do not risk being detected after performing double-spending attempts; this gives Bitcoin users considerable incentives to mount double-spending attacks against fast payments (given the absence of detection techniques). It is interesting to note that we have performed thousands of double-spending attempts using fixed Bitcoin addresses without having to bear any penalty.

Given that the economic standing of Bitcoin is still in its infancy, this vulnerability in the use of Bitcoin might severely hinder its growth and trustworthiness. We therefore argue that, unless effective detection techniques are integrated in current Bitcoin clients, vendors should not accept fast Bitcoin payments.

5. DETECTING DOUBLE-SPENDING IN FAST BITCOIN PAYMENTS

In this section, we evaluate a number of techniques that can be used to detect double-spending in fast payments. We start by analyzing two detection strategies originally recommended by Bitcoin developers, namely: *using a listening period* and *inserting observers*. We evaluate the prospects of both strategies and we show that these techniques are not always effective in detecting double-spending. We then propose an efficient detection technique that is based on the propagation of double-spent transactions in the network and we show that it does not incur significant modifications to existing Bitcoin clients.

5.1 Using a "Listening Period"

As advocated in [13], one possible way for $\mathcal{V}$ to detect double-spending attempts is to adopt a "listening period", of few seconds, before delivering its service to $\mathcal{A}$; during this period, $\mathcal{V}$ monitors all the transactions it receives, and checks if any of them attempts to double-spend the coins that $\mathcal{V}$ previously received from $\mathcal{A}$. This detection technique is based on the intuition that since it takes every transaction few seconds[11] to propagate to every node in the Bitcoin network, then it is highly likely that $\mathcal{V}$ would receive both $\mathrm{TR}_{\mathcal{V}}$ and $\mathrm{TR}_{\mathcal{A}}$ within the listening period (and before granting service to $\mathcal{A}$). Note that this technique requires a modification of current Bitcoin clients (i.e., to process and compare all incoming transactions).

Defeating the Listening Period.

We show that this detection technique can be circumvented by $\mathcal{A}$ as follows. $\mathcal{A}$ can attempt to delay the transmission of $\mathrm{TR}_{\mathcal{A}}$ such that $t = (t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}})$ exceeds the listening period (Requirement **(3)**) while $\mathrm{TR}_{\mathcal{A}}$ still has a significant chance of being spread in the network. On one hand, as t increases, the probability that all the immediate neighbors of $\mathcal{V}$ in the Bitcoin P2P network receive $\mathrm{TR}_{\mathcal{V}}$ first also increases; when they receive $\mathrm{TR}_{\mathcal{A}}$ later on, $\mathrm{TR}_{\mathcal{A}}$ will not be added to the memory pool of $\mathcal{V}$'s neighbors and as such $\mathrm{TR}_{\mathcal{A}}$ will not be forwarded to $\mathcal{V}$. On the other hand, $\mathcal{A}$ should make sure that $\mathrm{TR}_{\mathcal{A}}$ was received by enough peers so that Requirement **(2)** can be satisfied. To that end, $\mathcal{A}$ can increase the number of helpers it controls.

[11]Our experiments in Section 4.4 show that the average time for a peer to receive both $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ is approximately 3.354 seconds if both transactions were sent adjacently in time.

To validate this claim, we conducted experiments using the setup described in Section 4.4. Our experiments aimed at finding triplets (Δt, $N_{\mathcal{H}}$, C), where $N_{\mathcal{H}}$ is the number of helper nodes, and C is the number of $\mathcal{V}$' connections, such that the probability $\mathrm{P_D}$ that $\mathcal{V}$ receives $\mathrm{TR}_{\mathcal{A}}$ (after having received $\mathrm{TR}_{\mathcal{V}}$) is minimized. We illustrate our results in Table 3.

In Table 3(a), we include a number of triplets (Δt, $N_{\mathcal{H}}$, C) for which $\mathrm{P_S} > 0$ and $\mathrm{P_D} = 0$ (i.e., $t = t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}} = \infty$). These are instances of the case where *all* the neighbors of $\mathcal{V}$ receive $\mathrm{TR}_{\mathcal{V}}$ first and do not forward $\mathrm{TR}_{\mathcal{A}}$. We point out that in this case, although $\mathrm{P_S}$ is modest, the advantage of $\mathcal{A}$ in mounting double-spending attack is high since the probability that $\mathcal{V}$ detects her misbehavior is zero. In Table 3(b), we show other instances of (Δt, $N_{\mathcal{H}}$, C) for which $\mathrm{P_S} \geq \mathrm{P_D}$. Here, although $\mathrm{P_D} > 0$, $\mathcal{A}$ still has an advantage in mounting double-spending attacks, since these attacks are more likely to succeed.

Our findings therefore show that, even if $\mathcal{V}$ adopts a listening period of few tens of seconds, double-spending is still possible. Our experiments also show that the triplets (Δt, $N_{\mathcal{H}}$, C) resulting in $\mathrm{P_S} \geq \mathrm{P_D}$ do not depend on the location of $\mathcal{H}$ nor $\mathcal{V}$ nor on the time of the measurements[12]; as shown in Table 3, the same triplets can be used repeatedly by $\mathcal{A}$ to perform attacks at various times and in different network topologies.

The Connectivity of $\mathcal{V}$ as a Security Parameter.

As shown from our results in Table 3, the fewer connections of $\mathcal{V}$, the more likely is that *all* the neighbors of $\mathcal{V}$ receive $\mathrm{TR}_{\mathcal{V}}$ before $\mathrm{TR}_{\mathcal{A}}$ and thus that $\mathcal{V}$ does not receive $\mathrm{TR}_{\mathcal{A}}$. Conversely, as the number of connections of $\mathcal{V}$ increases, the effort (i.e., the number of helpers, the amount of delay) that $\mathcal{A}$ needs to invest to ensure that none of $\mathcal{V}$'s neighbors receive $\mathrm{TR}_{\mathcal{A}}$ becomes considerable. This is depicted in Figure 10; when $\mathcal{V}$ adopts a listening period of at least 15 seconds, $\mathrm{P_D}$ increases as the number of $\mathcal{V}$'s connections increases. When $\mathcal{V}$ has more than 100 connections, the probability that it does not receive $\mathrm{TR}_{\mathcal{A}}$ is negligible; in this case, using a listening period might be effective to detect double-spending.

However, since the connectivity of Bitcoin peers largely varies with the network churn[13] (Figure 4), $\mathcal{V}$ needs to constantly monitor its connection count to ensure that it does not drop below a threshold so that it can detect double-spending by adopting a listening period. In Section 5.3, we propose a technique that effectively detects double-spending even in the case when the number of the connections of $\mathcal{V}$ is small.

5.2 Inserting Observers in the Network

Another possible technique that naturally extends the aforementioned proposal based on the adoption of listening period would be for $\mathcal{V}$ to insert a node that it controls within the Bitcoin network—an "observer"—that would directly relay to $\mathcal{V}$ all the transactions that it receives. This technique was originally proposed by Bitcoin developers in [13]. In this case, $\mathcal{V}$ can be aware of a double-spending attempt if either it or its observer receive $\mathrm{TR}_{\mathcal{A}}$.

Evaluation.

We evaluated this technique using up to 5 observers, that were randomly chosen from the hosts listed in Table 2. Our findings in Table 3 show that this method can help detecting double-spending

[12]Our results were stable across different measurement times.

[13]For example, some of the nodes used in our experiments could not acquire more than 40 connections over a period of few days, due to the fact that these nodes were often restarted. In such cases, other Bitcoin peers will assign a low priority when connecting to these nodes.

(a) Example of triplets (Δt, $N_{\mathcal{H}}$, C) where $\mathrm{P_S} > 0$, $\mathrm{P_D} = 0$ and $t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}} = \infty$. In these cases, $\mathcal{V}$ never receives $\mathrm{TR}_{\mathcal{A}}$ and as such can not detect double-spending attacks, even if it adopts a very large listening period.

	$\mathrm{P_S}$	$\mathrm{P_D}$	$t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}}$ **(sec)**	**% Observed**
South America, 8 Connections, 3 Helpers, $\Delta t = 2.5$	7.7%	0%	∞	53%
South America, 8 Connections, 4 Helpers, $\Delta t = 3.0$	13.33%	0%	∞	57%
Asia Pacific, 8 Connections, 3 Helpers, $\Delta t = 2.75$	10%	0%	∞	57%
Asia Pacific, 8 Connections, 3 Helpers, $\Delta t = 2.75$	5%	0%*	∞	66%
North America, 20 Connections, 3 Helpers, $\Delta t = 2.75$	5%	0%	∞	47%
Asia Pacific, 60 Connections, 1 Helper, $\Delta t = 3.00$	10%	0%*	∞	20%

(b) Example of triplets (Δt, $N_{\mathcal{H}}$, C) where $\mathrm{P_S} \geq \mathrm{P_D}$. In these cases, $\mathcal{V}$ can detect double-spending attempts with some probability by adopting a listening period, but since $\mathrm{P_S} \geq \mathrm{P_D}$, then a number of $\mathcal{A}$'s double-spending attempts will not be detected, which gives her incentives to perform double-spending attempts.

	$\mathrm{P_S}$	$\mathrm{P_D}$	$t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}}$ **(sec)**	**% Observed**
Europe, 8 Connections, 3 Helpers, $\Delta t = 2.00$	10%	10%	8.664	53%
Europe, 8 Connections, 3 Helpers, $\Delta t = 2.25$	10%	10%*	5.65	47%
South America, 8 Connections, 2 Helpers, $\Delta t = 2.5$	20%	6.66%*	3.749	62%
Asia Pacific, 8 Connections, 2 Helpers, $\Delta t = 1.75$	55%	20%*	5.5	91%
North America, 20 Connections, 5 Helpers, $\Delta t = 3.00$	11%	11%	3.208	46%
North America, 20 Connections, 1 Helper, $\Delta t = 1.25$	30%	30%*	3.34	78%
North America, 20 Connections, 4 Helpers, $\Delta t = 2.00$	82%	63%	2.85	78%
North America, 20 Connections, 2 Helpers, $\Delta t = 2.00$	20%	20%*	4.79	60%
North America, 20 Connections, 1 Helper, $\Delta t = 1.50$	40%	30%*	3.51	60%
Europe, 20 Connections, 3 Helpers, $\Delta t = 1.0$	45%	45%*	3.844	87%
Europe, 30 Connections, 1 Helper, $\Delta t = 1.5$	15%	10%*	3.412	42%
Asia Pacific, 40 Connections, 1 Helper, $\Delta t = 2.9$	10%	10%*	4.946	42%
Europe, 40 Connections, 1 Helper, $\Delta t = 1.25$	10%	10%	1.841	36%
Europe, 40 Connections, 2 Helpers, $\Delta t = 1.5$	20%	20%*	3.075	36%
South America, 40 Connections, 1 Helper, $\Delta t = 2.0$	30%	40%	3.217	57%
Asia Pacific, 80 Connections, 1 Helper, $\Delta t = 3.7$	10%	20%	5.04	18%
Europe, 80 Connections, 1 Helper, $\Delta t = 2.75$	13.33%	26.67%	5.093	28%
Asia Pacific, 100 Connections, 1 Helper, $\Delta t = 1.5$	80%	80%	2.807	88%

Table 3: Monitoring received transactions at $\mathcal{V}$ (Section 5.1) and its observers (Section 5.2) to detect double-spending. Each data point corresponds to 20 measurements. The helpers used in these experiments had between 125 and 400 connections. "$\mathrm{P_D}$" denotes the probability that $\mathcal{V}$ receives $\mathrm{TR}_{\mathcal{A}}$. "% Observed" refers to the fraction of observers (among 5) that received $\mathrm{TR}_{\mathcal{A}}$. ($t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}}$) refers to the average of the time it takes $\mathcal{V}$ to receive $\mathrm{TR}_{\mathcal{A}}$ after having received $\mathrm{TR}_{\mathcal{V}}$, for those cases where $\mathcal{V}$ receives $\mathrm{TR}_{\mathcal{A}}$ ($\mathrm{P_D}$). * refers to the case where $\mathrm{TR}_{\mathcal{A}}$ was received, but after the listening period (i.e., after 15 seconds).

as all double-spent transactions were received by at least one observer within few seconds. However, given that $\mathcal{A}$ delays the transmission of $\mathrm{TR}_{\mathcal{A}}$, our results show that only a subset of the observers receive $\mathrm{TR}_{\mathcal{A}}$. As mentioned previously, this corresponds to the case where all the neighbors of these observers have received $\mathrm{TR}_{\mathcal{V}}$ first and as such they will not forward $\mathrm{TR}_{\mathcal{A}}$ back to the observers. Therefore, $\mathcal{V}$ needs to employ a considerable number of observers (≈ 3) (that connect to a large number of Bitcoin peers) to ensure that at least one observer detects any double-spending attempt; this, however, comes at the expense of additional costs for $\mathcal{V}$ to maintain the observers in the network.

Clearly, in the general case where $\mathcal{A}$ attempts to n-times spend the same coins, the larger is n, the bigger is the probability that this misbehavior is detected by fewer observers in the network.

5.3 Forwarding Double-Spending Attempts in the Network

In order to efficiently detect double-spending on fast Bitcoin payments, we propose that Bitcoin peers forward all transactions that attempt to double-spend the same coins in the Bitcoin network. Namely, our technique unfolds as follows. Whenever a peer receives a new transaction, it checks whether the transaction uses coins that have not been spent in any other transaction that resides in the block chain and in their memory pool. If so, then peers follow the current protocol of Bitcoin; peers add the transaction to their memory pool and forward it in the network. If, on the other hand, peers detect that there is another transaction in their memory pool that spends the same coins to different recipients, then peers forward the transaction to their neighbors (without adding the transaction to their memory pools).

The main intuition behind this technique is that while $\mathcal{A}$ might be able to prevent $\mathcal{V}$ and a subset of $\mathcal{V}$'s observers from receiving $\mathrm{TR}_{\mathcal{A}}$, a considerable number of Bitcoin peers receive both $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$. If the majority of these peers are honest[14], both transactions would eventually reach $\mathcal{V}$ within few seconds. The double-spending of $\mathcal{A}$ can be therefore detected before $\mathcal{A}$ actually receives the service from $\mathcal{V}$. We emphasize that our proposed technique

[14]This is the underlying assumption that ensures the correct operation of Bitcoin.

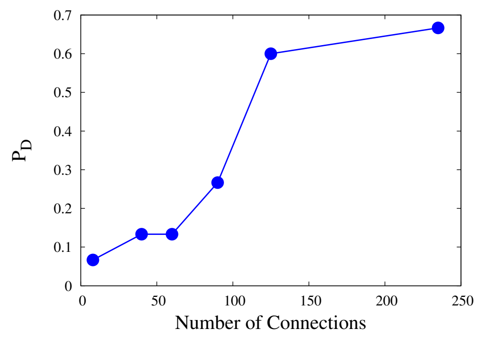

Figure 10: The connectivity of $\mathcal{V}$ as a security parameter. Here, $\mathcal{V}$ is located in Asia Pacific, $\Delta t = 3.0$ seconds and $\mathcal{A}$ controls 4 helpers.

does not change the spread of each transaction within the memory pools of Bitcoin peers (as such, it does not affect the success probability of the attack). Instead, this technique ensures that both transactions are received within few seconds by $\mathcal{V}$ and that any possible double-spending attempt is detected almost immediately. This intuition is based on our previous measurements: our experiments in Section 4.3 show that the average time for a peer a transaction is approximately 3.354 seconds after the transaction has been released in the Bitcoin network.

We implemented this technique and integrated it with Bitcoin client version 0.5.2. Our modified Bitcoin implementation is available for download at [7]. Our modified Bitcoin client also keeps track of the number of established connections to warn the user when this number drops below a threshold value (80 in our case) and uses our proposed detection technique to (visually) alert the user when a double-spending attempt was detected on any of its transactions.

We have evaluated the performance of our modified client by integrating it in the Bitcoin network for a period of 7 consecutive days[15]. During the evaluation period, our modified client was able to forward all double-spent attempts (that we manually injected in the network) with a detection rate of 100 % and a false negative rate of 0 % (i.e., only double-spent transactions were forwarded while duplicate and ill-formed transactions were appropriately filtered).

We acknowledge that this detection technique can result in the increase of the number of transactions circulating in the Bitcoin network and could be (ab)-used to affect the performance of the network. We argue however that, given the ongoing performance optimizations in the Bitcoin software and given that our technique solely relies on the forwarding of pending transactions (i.e., confirmed transactions should not be forwarded in our technique), the performance penalty that is incurred by the Bitcoin network due to our proposal can be, to a large extent, tolerated.

6. RELATED WORK

First introduced in 2008, Bitcoin has recently attracted the attention of the research community. In [21], Elias investigates the legal aspects of privacy in Bitcoin. In [35], Reid and Harrigan explore user anonymity limits in Bitcoin. In [25], Babaioff *et al.* address the lack of incentives for Bitcoin peers to include recently announced transactions in a block. In [20], Syed *et al.* propose a user-friendly technique for managing Bitcoin wallets.

Finney [17] describes 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 to the network after the attacker double-spends the same coins using fast payments and acquires a given service. Clark *et al.* [31] proposes the use of the Bitcoin PoW to construct veritable commitment schemes. Barber *et al.* [26] analyze possible ways to enhance the resilience of Bitcoin against a number of security threats; they do not, however, analyze the security of fast Bitcoin payments.

Double-spending in online payments has received considerable attention in the literature [24, 32]. In Credit-Card based payments, fairness is achieved through the existence of a bank (e.g., [28, 34]) or another trusted intermediary. Here, the intermediaries are trusted *(i)* to verify that the client has not already spent the funds he/she is paying the vendor with, and *(ii)* to reverse a charge, if the vendor has misbehaved.

Micropayments [23, 33, 36] is an efficient payment scheme aiming primarily at enabling low-cost transactions. Here, the payer provides signed endorsements of monetary transfers on the vendor's name. Digital signatures in these systems constitute the main double-spending resistance mechanism. ECash [29, 30] is another form of digital cash which supports payer anonymity. ECash offers strong accountability guarantees by relying on a set of cryptographic primitives that ensure that when a user double-spends a coin, his/her identity is revealed.

Similar to Bitcoin, a number of decentralized payment systems [27, 37] were designed to resist double-spending attacks. In [37], Yang and Garcia-Molina introduce a P2P payment system, in which the first owner of an electronic coin authorizes that coin's transfer among other peers in the network. Here, the generator of the coin is responsible for preventing double-spending.

In [27], Belenkiy *et al.*, introduce an ECash-based P2P payment scheme that provides accountability at the cost of privacy. In [33], Karame *et al.* propose a novel micropayment model based on verifiable microcomputations.

7. CONCLUSION

The ability of Bitcoin to support and process fast payments—in which the time to acquire a service is in the order of few seconds—is of paramount importance to its growth and popularity among users. Given the increasing reliance on Bitcoin among businesses, the use of Bitcoin in fast payment scenarios is only expected to increase. However, given its design, Bitcoin can only ensure the security of payments when a payment verification time of few tens of minutes can be tolerated. Moreover, our results show that the standard verification time of Bitcoin payments exhibits a large variance since the block generation time follows a shifted geometric distribution. This slow payment verification is clearly inappropriate for fast payments; it is, however, essential for the detection of double-spending attacks.

In this paper, we analyzed and evaluated the double-spending resilience of Bitcoin in fast payments. More specifically, we showed that not only these attacks succeed with overwhelming probability, but also that—contrary to common beliefs—they do not incur any significant overhead on the attacker. As far as we are aware, our experiments constitute the first comprehensive double-spending measurements in Bitcoin.

We also explored the solution space for detecting double-spending attacks against fast Bitcoin payments. Our findings show that the techniques recommended by Bitcoin developers for fast transactions are not always effective in detecting double-spending. Given that the vulnerability of existing clients to double-spending might

[15]During this evaluation period, the modified client was located in Europe.

severely harm the growth of Bitcoin, and impact its financial and economic standing, we argue that the integration of effective detection techniques to counter double-spending attacks against fast payments in Bitcoin emerges as a necessity. Based on our findings, we propose a modification to the current Bitcoin implementation in order to effectively detect double-spending attacks against fast payments.

Acknowledgements

The authors thank Matthias Herrmann and Hubert Ritzdorf for collecting various measurements in the Bitcoin network. The authors also thank the anonymous reviewers for their feedback and their helpful comments.

8. REFERENCES

[1] Bitcoin – Wikipedia, Available from https://en.bitcoin.it/wiki/Introduction.

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

[3] Bitcoin Charts, Available from http://bitcoincharts.com/.

[4] Bitcoin ATM, Available from http://bitcoinatm.com/.

[5] CNN: Bitcoin's uncertain future as currency, Available from http://www.youtube.com/watch?v=75VaRGdzMM0.

[6] FAQ - Bitcoin, Available from https://en.bitcoin.it/wiki/FAQ.

[7] Double Spending Fast Payments in Bitcoin , Available from http://www.syssec.ethz.ch/research/Bitcoin.

[8] Bitcoin Block 80000, Available from http://blockexplorer.com/b/80000.

[9] Protocol Rules – Bitcoin, Available from https://en.bitcoin.it/wiki/Protocol_rules.

[10] Protocol Specifications – Bitcoin, Available from https://en.bitcoin.it/wiki/Protocol_specification.

[11] Difficulty – Bitcoin, Available from https://en.bitcoin.it/wiki/Difficulty.

[12] Block hashing algorithm – Bitcoin, Availabe from https://en.bitcoin.it/wiki/Block_hashing_algorithm.

[13] Myths - Bitcoin, Available from https://en.bitcoin.it/wiki/Myths#Point_of_sale_with_bitcoins_isn.27t_possible_because_of_the_10_minute_wait_for_confirmation.

[14] Casascius Bitcoin POS system, Available from https://en.bitcoin.it/wiki/Casascius_Bitcoin_POS_system.

[15] Satoshi Client Node Connectivity, Available from https://en.bitcoin.it/wiki/Satoshi_Client_Node_Connectivity.

[16] Bitcoin Block Explorer, Available from http://blockexplorer.com/.

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

[18] Comparison of Mining Pools, Available from https://en.bitcoin.it/wiki/Comparison_of_mining_pools.

[19] Comparison of Mining Hardware, Available from https://en.bitcoin.it/wiki/Mining_hardware_comparison.

[20] Bitcoin Gateway, A Peer-to-peer Bitcoin Vault and Payment Network, 2011. Available from http://arimaa.com/bitcoin/.

[21] Bitcoin: Tempering the Digital Ring of Gyges or Implausible Pecuniary Privacy, 2011. Available from http://ssrn.com/abstract=1937769ordoi:10.2139/ssrn.1937769.

[22] SATOSHI NAKAMOTO. Bitcoin: A Peer-to-Peer Electronic Cash System, 2009.

[23] ANDROULAKI, E., RAYKOVA, M., STAVROU, A., AND BELLOVIN, S. M. PAR: Payment for Anonymous Routing. In *Proceedings of PETS* (2008).

[24] ASOKAN, N., JANSON, P., STEINER, M., AND WAIDNER, M. State of the Art in Electronic Payment Systems. *IEEE Computer* (1999).

[25] BABAIOFF, M., DOBZINSKI, S., OREN, S., AND ZOHAR, A. On Bitcoin and Red Balloons. *CoRR* (2011).

[26] BARBER, S., BOYEN, X., SHI, E., AND UZUN, E. Bitter to Better - How to Make Bitcoin a Better Currency. In *Proceedings of Financial Cryptography and Data Security* (2012).

[27] BELENKIY, M., CHASE, M., ERWAY, C., JANNOTTI, J., KÜPÇÜ, A., LYSYANSKAYA, A., AND RACHLIN, E. Making P2P Accountable without Losing Privacy. In *Proceedings of WPES* (2007).

[28] BELLARE, M., GARAY, J., HAUSER, R., KRAWCZYK, H., STEINER, M., HERZBERG, A., TSUDIK, G., VAN HERREWEGHEN, E., AND WAIDNER, M. Design, Implementation and Deployment of the iKP Secure Electronic Payment System. *IEEE Journal on Selected Areas in Communications* (2000).

[29] CAMENISCH, J., HOHENBERGER, S., AND LYSYANSKAYA, A. Compact E-Cash. In *Proceedings of Advances in Cryptology - EUROCRYPT* (2005).

[30] CHAUM, D., FIAT, A., AND NAOR, M. Untraceable electronic cash. In *Proceedings on Advances in Cryptology - CRYPTO* (1990).

[31] CLARK, J., AND ESSEX, A. (Short Paper) CommitCoin: Carbon Dating Commitments with Bitcoin. In *Proceedings of Financial Cryptography and Data Security* (2012).

[32] EVERAERE, P., SIMPLOT-RYL, I., AND TRAORE, I. Double Spending Protection for E-Cash Based on Risk Management. In *Proceedings of Information Security Conference* (2010).

[33] KARAME, G., FRANCILLON, A., AND ČAPKUN, S. Pay as you Browse: Microcomputations as Micropayments in Web-based Services. In *Proceedings of WWW* (2011).

[34] KRAWCZYK, H. Blinding of Credit Card Numbers in the SET Protocol. In *Proceedings of the International Conference on Financial Cryptography* (1999).

[35] REID, F., AND HARRIGAN, M. An Analysis of Anonymity in the Bitcoin System. *CoRR* (2011).

[36] RIVEST, R. Peppercoin Micropayments. In *Proceedings of Financial Cryptography* (2004).

[37] YANG, B., AND GARCIA-MOLINA, H. PPay: micropayments for peer-to-peer systems. In *Proceedings of the ACM Conference on Computer and Communication Security* (2003).

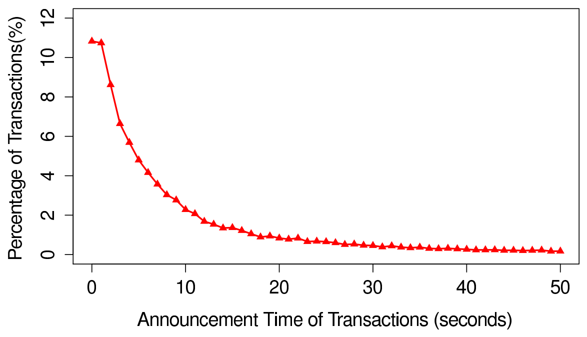

Figure 11: Distribution of the announcement times of transactions. To measure the transaction announcement times, we parsed the block chain of Bitcoin and we counted the number of transactions in each block and we assume that transactions are announced uniformly at random within two successive blocks.

APPENDIX

A. BLOCK GENERATION IN BITCOIN

To generate a block, peers *work* on solving a PoW problem. In particular, given the set of transactions that have been announced since the last block's generation, and the hash of the last block, Bitcoin peers need to find a nonce that would make the `SHA-256` hash of the formed block smaller than a 256-bit number ($\textit{target}$):

$$\texttt{SHA-256}\{\mathsf{Bl}_l \,||\, \mathrm{MR}(\mathrm{TR}_1, \dots, \mathrm{TR}_n) \,||\, \mathrm{No}\} \leq \textit{target}, \quad (4)$$

where Bl_l denotes the last generated block, $\mathrm{MR}(\overline{x})$ denotes the root of the Merkle tree with elements $\overline{x}$, $\mathrm{TR}_1 \,||\, \dots \,||\, \mathrm{TR}_n$ is a set of transactions which have been announced (and not yet confirmed) since Bl_l's generation, and No is the 32-bit nonce. To solve the PoW, each peer chooses a particular subset of the candidate solutions' space and performs brute-force search. It is apparent that the bigger $\textit{target}$ is, the easier is to find a nonce that satisfies the PoW. For the purpose of our analysis, we note the following:

1. The probability of success in a single nonce-trial is negligible. Taking in consideration that SHA-256 maps its arguments randomly to the target space, each of the 2^{32} nonces has $\frac{\textit{target}}{max(\textit{target})}$ probability of satisfying the PoW, where, $\textit{target}$ is the number of targets which are smaller than $\textit{target}$ and $max(\textit{target}) = 2^{256} - 1$ is the number of all targets.

2. Peers compute their PoW independently; as such, the probability that one of them succeeds does not depend on the progress of PoW of the others.

3. Peers are frequently required to restart their PoW. In particular, whenever a new transaction is added to the memory pool of a peer, the Merkle root (included in the block) changes; therefore, the effort in constructing the PoW is "reset". This also applies to the case when new blocks are generated; the "previous block" field needs to be updated (cf. Figure 1). For the sake of our analysis, we approximate the time interval between the announcement of successive transactions as follows. We extract the various block generation times from the Bitcoin block explorer (cf. Figure 1) and we assume that transactions are announced uniformly at random within two successive block generations. Our findings (Figure 11) show that the time interval between the announcement of most pairs of successive transactions is below 15 seconds. Therefore, we assume in the sequel that the PoW for block generation is "reset" approximately every $\mathrm{dt} \approx 15$ seconds.

Given the first two observations, the probability of a peer in succeeding in an *individual* block generation attempt can be modeled as an independent Bernoulli process with success probability $\varepsilon = \frac{\textit{target}}{max(\textit{target})}$.

Given this, and based on the last observation, we claim that *consecutive* block generation attempts can be modeled as sequential Bernoulli trials with *replacement*. Our claim for replacement is justified by the fact that maximum possible PoW progress performed by a peer (expressed as a number of hash calculations) before its PoW resets, is negligible in comparison to $max(\textit{target})$. This is the case since the PoW progress approximates $2^{35} \ll max(\textit{target})$ given the computing power of most Bitcoin peers [18, 19]. This also means that ε is always equal to $\frac{\textit{target}}{max(\textit{target})}$.

Let n_i refer to the number of attempts that a peer m_i performs within a time period δt. Typically, δt is in the order of few minutes. the probability p_i of m_i finding at least one correct PoW within these trials is given by $p_i = 1 - (1 - \varepsilon)^{n_i}$. Since ε and n_i are small, p_i can be approximated to $p_i = 1 - (1 - \varepsilon)^{n_i} \approx n_i\varepsilon$.

Therefore, the set of trials of m_i within δt can be unified to constitute a single Bernoulli process with success probability $n_i\varepsilon$.

Assuming that there are ℓ peers, m_i, $i = 1 \dots \ell$ with success probability p_i, $i = 1 \dots \ell$ respectively, the overall probability of success in block generation can be approximated to:

$$\mathsf{p} \approx 1 - \prod_{i=1}^{\ell}(1 - p_i), \text{ or } \mathsf{p} = 1 - (1 - p)^{\ell} \approx \ell \cdot p,$$

where the latter corresponds to the simplified case when the peers have equal computing power, i.e., $p_i = p, i = 1 \dots \ell$. Note that $p\ell < 1$.

Let time be divided into small intervals $]t_0, t_1], \dots,]t_{n-1}, t_n]$ of equal size δt, where $t_0 = 0$ denotes the time when the last block was generated. Here, each peer can make up to n_i trials for block generation within each interval. Let the random variable X_k denote the event of success in $\delta t = \{t_k, t_{k+1}\}$. That is,

$$X_k = \begin{cases} 1 & \text{if a block is created within}\{t_{k-1}, t_k\}, \\ 0 & \text{otherwise.} \end{cases}$$

It is evident that: $\mathsf{Prob}(X_k = 1) = \mathsf{p}$. Conceivably, after a success in block generation, peers stop mining for that particular block. We denote the number of attempts until a success is achieved by another random variable $\mathcal{Y}$.

$$\mathsf{Prob}(\mathcal{Y} = k) = \mathsf{Prob}(X_k = 1) \prod_{i=1}^{k-1} \mathsf{Prob}(X_i = 0)$$
$$= \mathsf{p}(1 - \mathsf{p})^{k-1}.$$

Assuming a constant rate of trials per time window δt, the number of failures until a success is observed in block generation is proportional to the time it takes for a block to be generated. Let $\mathcal{T}$ denotes the time period till a block is generated.

$$\mathsf{Prob}(\mathcal{T} = k \cdot \delta t) = \mathsf{Prob}(\mathcal{Y} = k) \;=\; \mathsf{p}(1 - \mathsf{p})^{k-1}.$$

Given this, we conclude that the distribution of block generation times can be modeled with a shifted geometric distribution with parameter p. In Figure 1, we confirm this analysis and we show that (experimental) block generation times in Bitcoin, can be fitted to a shifted geometric distribution with $p = 0.19$. For the purpose of our experiments, we considered δt to be 2 minutes.