Implementation, evaluation and detection of a doublespend-attack on Bitcoin

Master thesis

Matthias Herrmann

April 24, 2012

Supervisors: S. Capkun, E. Androulaki, G. Karame

Department of computer science, ETH Zürich

Abstract

Bitcoin is a peer-to-peer payment scheme with a monetary volume of over 40 million USD and estimated 60'000 users worldwide, including several businesses like Drupal, Bitbrew, rasselzoo.ch or Meze grill.

In this master thesis, we evaluate the potential of doublespend-attacks on Bitcoin. We analyse the Bitcoin system, in particular the doublespend-protection procedure, and identify a weakness in certain usage scenarios.

The doublespend-protection procedure works by forming consensus about transactions every 10 minutes, which means that the expected confirmation time for a transaction is 5 minutes. This time frame is acceptable for online shops like rasselzoo.ch, and in such a usage scenario, the procedure provides sufficient protection from doublespend-attacks. However, for a restaurant like Meze grill or for a vending machine, this time frame is too big. In such a usage scenario, the procedure does not protect the user from doublespend-attacks. Businesses with this usage scenario, mostly brick and mortar businesses, are at risk of being the victim of a doublespend-attack.

We implement a doublespend-attack that functions in this usage scenario. We evaluate the attack by performing measurements with varying parameter settings to determine how they influence the attack. Since the attack is probabilistic, we are especially interested in the success probability, and how it is influenced by different attacker- and victim-configurations. Based on our measurements, we name security parameters and determine thresholds for them.

Furthermore, we investigate the detectability of the attack, especially from the perspective of the victim. We measure the influence the attack parameter settings have on detectability. Even though the current Bitcoin client software does not detect doublespend-attacks, detection can be implemented, and thus we try to find a parameter setting which renders the attack undetectable from the perspective of the victim.

Finally, we implement such a detection-mechanism. The mechanism informs the user whether a doublespend-attack was detected or not within 10 seconds, which is a huge improvement. Businesses operating in the previously vulnerable usage scenario can most likely accept such a time frame, thus this mechanism greatly reduces the risk for those businesses. However, the mechanism only detects attacks on a limited set nodes, which consists of the nodes that are able to detect the attack.

The mechanism also increases the number of nodes that are able to detect the doublespend-attack. If sufficiently many nodes are updated, this mechanism renders the attack detectable for all nodes in the network.

This thesis is based on the Bitcoin software version number 50000, but it was tested in the live Bitcoin network, with the newest version number at the time of this writing being 60000.

Acknowledgements

I would like to thank my supervisors for providing me with the resources I required to complete this thesis, and for being very helpful whenever I had questions.

I would also like to thank Sarah Aepli and Luka Malisa for their comments on my work, and for interesting discussions which led to further insight into the subject matter.

Furthermore, I would like to especially thank Hubert Ritzdorf who was extremely helpful in multiple aspects of my thesis, and whose ideas and work allowed me to work much more efficiently.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Bitcoin is a digital payment scheme that can be used to purchase real goods and services. It has estimated 60'000 users [16], including multiple businesses, and has a monetary volume of over 40 million USD [3, 4] in Bitcoins, the currency of Bitcoin. Bitcoin is a peer-to-peer system, so there is no central authority (i.e. bank) that controls accounts or transactions.

If an attacker can perform a doublespend-attack on the system, they can gain illicit access to goods and services from businesses and private people. This poses a risk to entities using Bitcoin, in particular businesses, since they usually have a higher transaction volume than private users.

A doublespend-attack on the Bitcoin system does not only cause direct damage to the victim of the attack, but also destroys some of the trustworthiness of the Bitcoin system. Trustworthiness is essential for any monetary system, as people have to trust that the value of their money will not deteriorate. This is especially the case for a system that handles transactions as well, since people have to trust that the money transferred to them is accepted by others, so they can transfer the money onwards if they desire.

Every attack on a system is also a risk for the attacker, since they might be held liable if the attack is discovered. As stated, there is no central authority in the Bitcoin system which keeps a registry of links from accounts to real people, so the identity of the attacker is not easy to determine. Furthermore, a user can create a new account by themselves for free and transfer money to it, so the attacker can create an account for the sole purpose of the attack, which makes identification even harder. We see that the risk for the attacker is small.

An attacker thus has a high incentive to perform an attack (monetary gain) combined with small disincentive to perform an attack (low risk), which makes an attack overall probable. We therefore think it is crucial for the Bitcoin system and its users that we explore the potential of such an attack.

1.2 Contribution

In this thesis, we analyse the current doublespend-protection procedure in the Bitcoin system. We describe an inherent weakness in the procedure and imple-

ment an attack which exploits this weakness. We perform a series of experiments with the attack, with varying parameters, in the live Bitcoin network and collect measurement data, which we present in this thesis. We implement a detection mechanism that detects doublespend-attacks in certain cases and, if widely deployed in the Bitcoin network, makes the attacks fully detectable in the network without interfering with older versions of the Bitcoin client software.

1.3 Problem description

As already stated, there is no central authority in Bitcoin, so consensus about transactions is formed in a distributed manner. This consensus procedure is protecting against doublespending.

The procedure groups unconfirmed transactions together in a block, which also references the previous block to form a linked list of blocks. This list is called the blockchain. A transaction in the blockchain is considered confirmed to a degree, which can be improved if the blockchain grows on, so a transaction two blocks deep in the blockchain is considered more confirmed than one in the newest block. The system is expected to create a block every 10 minutes.

All monetary data in Bitcoin is represented as transactions, so every 10 minutes consensus about the monetary situation[1] of all users is formed.

Creating a block is a probabilistic process with adjustable difficulty which depends on, among other things, the computation power of the Bitcoin network. If the power increases, some parameters of the block finding process are adjusted to keep the expected time to find a block at 10 minutes. We analyse the blockchain in chapter 6 to investigate this claim and discover that this process has a high standard deviation. Even with a 10 minute time frame, a payee would, in expectation, have to wait 5 minutes until their payment is confirmed, or even longer, if they want a stronger confirmation.

There are businesses which can accept payment confirmation times like this, e.g. online bookstores. Brick and mortar businesses, on the other hand, usually rely on a much shorter time frame for payment processing. An example of this would be a take-away restaurant which serves a lot of customers in a short time at lunch hour. Such a restaurant probably accepts unconfirmed payments, and may be even unaware of the risk, i.e. treating Bitcoins like cash. For such a use case, the doublespend-protection procedure is inadequate.

We implement a doublespend-attack which tricks a victim into believing that they have received a payment, but after 5 minutes, in expectation, they will lose the payment again. The attack we implement could be detected by certain nodes in the network, but there is currently no mechanism in the Bitcoin software detecting them. Furthermore, if a node with the potential to detect the attack is not the victim itself, no steps are taken to alert the victim, or to make the attack also detectable for them.

Summarising, businesses which do not wait 5 minutes, in expectation, for payment confirmation, or longer, if they want a stronger confirmation, are at risk, with no protection and no warning mechanism. The only way to even be aware of an attack as a victim ex post facto is to realise that some Bitcoins are missing, which happens after 5 minutes, in expectation.

[1] account balance, transaction history etc.

1.4 The attack

As already stated, in Bitcoin, all monetary data is represented as transactions. This applies to Bitcoins as well, there is no direct representation of a Bitcoin. In order to doublespend, we create a pair of transactions spending the same Bitcoins ($t_{genuine}$, t_{rogue}) with $t_{genuine}$ transferring the Bitcoins to the victim and t_{rogue} transferring the Bitcoins to some other entity, i.e. another account controlled by the attacker.

The attacker uses another node in the Bitcoin network, which is also controlled by them, called helper. To attack, the attacker sends $t_{genuine}$ to the victim and t_{rogue} to the helper. The victim and the helper then both start to spread the transaction they received in the Bitcoin network.

Since $t_{genuine}$ and t_{rogue} use the same Bitcoins, they are in conflict with each other. Nodes in the Bitcoin network accept only one of the transactions and ignore the other one, i.e. they accept the one arriving first, so $t_{genuine}$ and t_{rogue} will separate the Bitcoin network in two parts, with one part having accepted $t_{genuine}$ and the other part having accepted t_{rogue}.

Since both transactions are present in the network, both have a chance of getting confirmed. The larger the part of one transaction is, the higher are its chances of getting confirmed. It is therefore desirable for the attacker that t_{rogue} spreads wide in the network.

The attack is considered successful if:

- The victim receives and accepts $t_{genuine}$
- t_{rogue} is confirmed, i.e. t_{rogue} is included in the blockchain

In this outcome, the victim thinks they received a payment, while, later, consensus is reached that the payment was to some other entity. This causes the victim to lose the payment again.

1.5 Detection mechanism

As previously mentioned, the attack splits the network in two parts. Nodes that have only neighbours belonging to one part never see one of the two transactions, e.g. they only see $t_{genuine}$, which, for them, is indistinguishable from a scenario where there only exists one transaction. This means that for those nodes, the attack is not detectable.

Nodes at the border of the two parts, i.e. nodes with at least one neighbour which accepts $t_{genuine}$ and at least one neighbour which accepts t_{rogue}, on the other hand receive both transactions.

We implement a detection mechanism that works on those nodes and that outputs a warning for the victim, if they are one of those nodes.

The mechanism works by comparing invalid transactions that would otherwise simply be ignored with the local transaction pool. It checks whether the received invalid transaction uses the same Bitcoins as a previously seen transaction.

This mechanism provides only limited protection for a node, since only nodes on the border of the two network parts can detect the attack and be warned. An attacker could potentially find parameter settings such that the victim does

not lie on the border. Note that the border is formed at random, since network topology, message forwarding schedule and network delays all contain randomness, but we cannot exclude the possibility of an attacker finding parameter settings that create the desired scenario with sufficiently high probability.

We thus also implement a mechanism that increases the detectability of the attack in the network. To achieve this, a node forwards a detected doublespend-transaction, and so increases the number of nodes that receive both transactions. If sufficiently many nodes are updated to behave this way, every node receives both $t_{genuine}$ and t_{rogue}, thus the attack is rendered fully detectable. Note that not all nodes need to be updated for full detectability, though only updated nodes output a warning.

This detection mechanism requires a much smaller time frame, which can be considered acceptable even for businesses like a take-away restaurant.

1.6 Thesis layout

In chapter 2, we present similar research on this topic and how this thesis relates to it.

In chapter 3, we present some background knowledge about the Bitcoin system and the doublespend-protection procedure currently in place, including an explanation of the weakness in the latter which we exploit in our attack.

In chapter 4, we present our attack both conceptually and with detailed information about the implementation.

We experimentally evaluate our implementation of the attack in the live Bitcoin network with varying parameter settings. In chapter 5, we present the setup of our experiments as well as our measurement methodology, and we also briefly mention some problems we encountered, and how we solved them. In chapter 6, we present the results of our experiments followed by an interpretation of the measurement data.

In chapter 7, we present our detection mechanism both conceptually and with detailed information about the implementation. We also present an evaluation of our detection mechanism.

Chapter 2

Related work

In this chapter, we present other research on the topic of doublespending and Bitcoin and how this thesis relates to it.

Doublespending is a well-known problem of electronic currencies, since creating a copy of a digital object is a trivial task. Different approaches to the problem exist.

Some systems employ a trusted central control instance (like Paypal [7]) which handles transactions. Chaum et al. presents a system that "guarantees untraceability, yet allows the bank to trace a 'repeat spender'" [5], based on cryptography. Even et al. [17] propose a system relying on a hardware token, called electronic wallet. Bitcoin itself uses a proof-of-work based system [12].

Security in Bitcoin is addressed from various perspectives. Reid [15] analyses the anonymity the Bitcoin system provides and Barber et al. [2] present several attacks on Bitcoin. There are, however, not a lot of publications about doublespending in Bitcoin, and we could not find any implementations. The issue is addressed in the Bitcoin wiki as a warning stating "To protect against double spending, a transaction should not be considered as confirmed until a certain number of blocks in the block chain confirm, or verify that the transaction [sic]." [18], which amounts to 1 hour, in expectation, of waiting time. There are some suggestions for businesses that cannot wait for a confirmation which are not very helpful or not yet implemented [19].

One solution that does exist is to use a trusted proxy for a transaction, called a green address [20], which transfer the risk from a merchant to the operator of the proxy. The proxy can mitigate the risk with a service fee. Another solution is to have a trusted service [14] that observes the network and tries to find doublespend-attacks. Both approaches depend on trust, and change the peer-to-peer idea of Bitcoin slightly by introducing a central service.

There is some discussion about theoretical doublespend-attacks in Bitcoin. Finney proposes an attack [8] and Satoshi writes "if a double-spend has to wait even a second, it has a huge disadvantage" [11]. We implement a doublespend-attack and investigate Satoshis claim in chapter 6.

There also exists a proposal for a doublespend-warning mechanism in the Bitcoin client software that sends doublespend-warning-messages [9]. We implement a doublespend-warning mechanism similar to the proposed one, which doesn't use a different message type in chapter 7.

Chapter 3

Background

In this chapter, we present some background knowledge about the Bitcoin system that is helpful to understand the following chapters. We begin by an overview of the Bitcoin system followed by a more detailed explanation of certain aspects of it.

Bitcoin is a digital payment scheme used for the transfer Bitcoins. Bitcoins themselves can be seen as a currency in the sense that they can be exchanged for other forms of money (e.g. USD) at exchanges [10] or for real goods [6]. However, it should be noted that there is legal concern [13] whether Bitcoin is electronic money.

The system was proposed in 2008 by Satoshi Nakamoto [12]. Bitcoin stands out as being a peer-to-peer system with no central authority, or bank, which keeps a record of all the account balances or registers all the transactions, and thus there is no central control point of the system which could potentially freeze certain accounts or transactions. Furthermore, the system is free to use and account creation does not require any registration.

Because of those facts, the number of users of Bitcoin is not easy to determine, but measurements [16] show about 60'000 connected nodes to the network, which can be viewed as an estimate for the number of users. The number of Bitcoins in the system is steadily growing by design, until it reaches a certain amount, and is at 8'509'000 Bitcoins [3] at the time of this writing. With an exchange rate of 4.90 USD per Bitcoin [4] at the time of this writing, this amounts to a total monetary volume of 41'694'100 USD.

As there is no central authority, the set of confirmed transactions is found via a consensus procedure and stored in a distributed manner. We will explain this procedure in detail later.

The Bitcoin network is randomly formed. A node finds (up to 125) peers in a randomly chosen IRC channel, which is chosen from a fixed set.

The equivalent of a bank account in Bitcoin is called a *(Bitcoin) address*. Each address is associated with a public/private key pair. This key pair is everything that a user needs to send and receive Bitcoins. The key pair is, along with user preferences etc. [21], stored in the file wallet.dat. The rest of the data relevant to Bitcoin is stored in the network, namely in a copy on each node.

Bitcoin stores all data (except keys etc., as mentioned) as *transactions*. Transactions, which we will explain in more detail later, contain all the in-

formation needed by the system, e.g. the balance of an account is the sum of incoming transactions minus the sum of outgoing transactions for this account. Bitcoins themselves have no explicit representation, their existence is implied by transactions.

When a new transaction is formed, it is sent from the creator to their peers in the Bitcoin network, where it is propagated until, eventually, all the nodes have received it. Later, it gets grouped into a new *block*, and thereby confirmed, by a process which we will explain later. A block, next to new transactions, also contains a reference the previous block, so the blocks form a linked list called the *blockchain*. The complete blockchain contains all the confirmed transactions in Bitcoin. Since the blockchain is stored in the network, the entire payment history of the Bitcoin system is public (and even browsable[1]).

3.1 Transactions

A transaction is an ownership transfer of Bitcoins from one address (payer) to another (payee). As a person can create several addresses, a transaction can also transfer Bitcoins from a person to themselves.

A transactions contains, for each payee:

- *references* to previous transactions [23]
- *a signature* of the hash of the previous transactions and the payees public key [12]
- *a value* stating how many Bitcoins should be transferred to this payee [23]

The references to previous transactions represent the Bitcoins to be transferred. As already stated, Bitcoins do not have a direct representation but are implied by transactions.

If a single previous transaction does not represent enough Bitcoins, e.g. when Alice wants to spend 5 Bitcoins, but only has 5 transactions transferring her 1 Bitcoin each, several references to previous transactions can be combined in one transaction.

The signature can be used by the payee to verify the "chain of ownership" [12].

The value states how much of the input should be transferred to the payee. A payer might not have the exact amount of Bitcoins to refer to, e.g. when Alice wants to spend 5 Bitcoins, but only has a transaction transferring her 6 Bitcoins.

A transaction, once referred to as a *previous transaction*, cannot be referred to as such again, and all the unspent Bitcoins are lost to the payer[2].

For a single transaction, there can be up to two recipients, so for a transaction there usually is a second recipient besides the payee, representing the payer (with a different address). The otherwise lost Bitcoins the payer sends to themselves is called *change* [23].

We extend the previous example: Alice wants to spend 5 Bitcoins to Bob, but only has one transaction $t_{toAlice}$, which transfers 6 Bitcoins to her, to refer

[1] `https://blockexplorer.com`

[2] they are not lost to the system, though

to. She creates a transaction t_{toBob} by referring $t_{toAlice}$ as a previous transaction and setting two recipients. One is Bob, who should receive 5 Bitcoins, and one is Alice, with a different address, generated just for this purpose, who should receive 1 Bitcoin.

It should be noted that Bitcoin allows the creation of complex transactions with a scripting language called Script[3]. These transactions can have complex redeeming scenarios such as the following: "It's [...] possible to require that an input be signed by ten different keys, or be redeemable with a password instead of a key." [23]. Within the scope of this thesis, we only use standard transactions.

3.2 Block creation / mining

Block creation is the consensus procedure used by the Bitcoin system. A transaction added to the blockchain can be considered confirmed by the system, i.e. consensus is reached that this transaction is valid and has happened. This description of confirmation is slightly simplified[4].

Blocks are found (or created) by nodes. Finding a block requires some computational work, and as the node solving the problem gains 50 Bitcoins, this process is called mining. This is the aspect of Bitcoin where differences between nodes exist, as a node with more computational power can mine better than a node with less power. The reward for finding a block even led to the formation of mining alliances (called mining pools) which are trying to find a block together and, if they are successful, they split the reward amongst themselves[5]. Mining is not required of a node, though. We didn't mine during all the experiments of this thesis.

To find a block, a miner has to solve a problem with adjustable difficulty. This allows the Bitcoin system to compensate for fluctuations in computational power in the network, e.g. due to better processors or a powerful node leaving. The problem difficulty is adjusted such that a block is expected to be found every 10 minutes. Note that this procedure implies that every 10 minutes, 50 new Bitcoins are created.

We now describe the problem a miner has to solve. A miner has to hash the header of the new block [22] such that the hash has a certain number of leading nibbles that are a zero[6]. The number of leading zeroes required is the difficulty which can be adjusted, whereby a requirement of more zeroes is more difficult. The blockheader contains a hash of all transactions that should be included in the block, so it depends on the transactions. It also contains a nonce which the miner can change in order to solve the problem.

A node has to try different nonces to find a solution, or they would break the hash function. This means the block finding process is probabilistic. We statistically analyse the time it took to find each block of the blockchain in chapter 6.

[3]Script is not Turing-complete. See `https://en.bitcoin.it/wiki/Script`

[4]for a more thorough description, see `https://en.bitcoin.it/wiki/Blockchain`

[5]like `https://mining.bitcoin.cz/`

[6]e.g. the hash of blockheader of block number 123456 is $000000000002917ed80650c6174aac8dfc46f5fe36480aaef682ff6cd83c3ca$ [1], with 12 leading zeroes

This procedure can be described as a voting system with "essentially one-CPU-one-vote" [12]. While an attacker might get lucky and find a hash to form one malicious block, trying to attack the blockchain by appending malicious blocks would need more than half of the computing power of the entire network to succeed with a high probability. This becomes even harder for them if they try to keep their attack up over several consecutive blocks, as "the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added." [12].

It is possible that two different new blocks are created almost simultaneously by two different nodes, which causes the blockchain to fork, so two linked lists with the same tail exist. Eventually, one list will get longer than the other and the nodes resolve this issue by accepting the longer list as the correct blockchain.

3.3 Doublespending

With real (paper, metal) money, it is easy to have a clear change of ownership. If Alice gives a coin to Bob, Alice no longer has the coin and Bob has the coin, as can be seen by anyone following the transaction, e.g. by seeing the coin in Bobs hand. With a digital coin of any form, this is not as simple, since creating a perfect copy of any digital object is easy.

Alice can create a copy of a digital coin before giving it to Bob, and later give the copy to Carol without either Bob or Carol being aware they are holding a coin that was copied, so both would agree to trade with Alice, e.g. Alice could buy a 1-coin chocolate bar from both Bob and Carol.

The process of using the same coin twice, like described, is called doublespending.

As previously mentioned, in Bitcoin, there is no direct representation of a coin, as its existence is only implied by transactions. A node that receives a transaction that transfers some Bitcoins and later receives another transaction transferring the same Bitcoins to someone else will classify the second transaction as invalid and ignore it. To create a copy of Bitcoins for later use is thus not a successful strategy for doublespending in Bitcoin.

What is possible, though, is to create two transactions simultaneously (we name them $t_{genuine}$ and t_{rogue}), which transfer the same Bitcoins to different payees, and to insert both transactions into the Bitcoin network at the same time. Since a node only accepts one of the two transactions, the network will split into two parts, as we will explain in more detail in chapter 4, with one part accepting $t_{genuine}$ and the other part accepting t_{rogue}.

Until consensus is reached in the network, i.e. a block is found including either $t_{genuine}$ or t_{rogue}, both transactions exist in the network. When a block is found, the transaction not included in the block will be dropped by all nodes and the one included in the block will be accepted by all nodes. This demonstrates how the consensus procedure protects against doublespending in Bitcoin.

As described, any mining node can find a block, which means that it is possible for both transactions to be confirmed. This implies that the described doublespend-attack is probabilistic, as an attacker without a significant part of the computational power of the network cannot reliably find a block with the desired transaction in it. It is possible to influence the outcome, though, by trying to affect the network spread of the transactions. We examine the effects

of different parameters on the outcome of this attack in chapter 6.

Since finding a block takes 10 minutes in expectation, a transaction takes 5 minutes in expectation to be confirmed. Before confirmation, a node has no way of knowing if there is a doublespend-attack happening with the current Bitcoin client software. Note that in particular, even when a node does receive both $t_{genuine}$ and t_{rogue}, the Bitcoin client software simply discards the transaction arriving later and does not warn the user.

While some businesses can afford to wait for these amounts of time to pass before a transaction is accepted, e.g. an online book store that doesn't start shipping the very moment someone orders a book, other businesses do not have those time frames available. Most brick and mortar businesses don't let their customers wait for a while before they are allowed to leave the premises, and some businesses even depend on a quick transaction of goods, e.g. a take-away restaurant. For a business like this, accepting Bitcoins is a constant risk they are taking and the current doublespend-protection procedure clearly isn't providing sufficient protection for them.

Chapter 4

The attack

In this chapter, we present our attack. We first explain conceptually how we attack the system and explain the detectability of our attack. We then present details about the implementation.

As described in chapter 3, Bitcoins are spent via a transaction, which references them. Doublespending a Bitcoin means issuing two transactions which reference the same Bitcoins, but differ in the recipient.

To attack, we create these two transactions and call them $t_{genuine}$ and t_{rogue}. $t_{genuine}$ contains the victim as a recipient and t_{rogue} contains some address controlled by the attacker as a recipient. We insert both transactions into the Bitcoin network simultaneously with two goals:

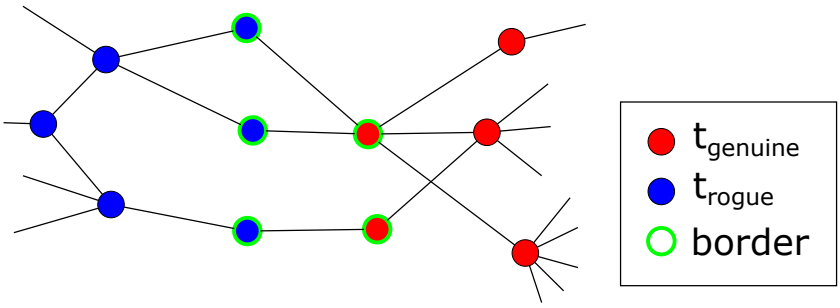
- The victim receives $t_{genuine}$
- t_{rogue} becomes part of the blockchain

The first goal states that the victim thinks they received the payment, as $t_{genuine}$ is a regular transaction from the attacker to the victim. Since the two transactions transfer the same Bitcoins, the second goal states that, in the network, consensus is reached that $t_{genuine}$ didn't happen, but t_{rogue} did happen.

The two transactions are in conflict with each other, so a node receiving one of them before the other one will classify the one later one as invalid and ignore it (see section 4.1). The goals can thus be reformulated as:

- The victim receives $t_{genuine}$ first
- t_{rogue} is received by a lot of nodes in the network first

The second goal is derived from the way the network forms consensus (described in chapter 3). Any miner trying to find a block tries to include its locally accepted transactions into the block. If more nodes have accepted t_{rogue} locally, then, likely, more miners have accepted t_{rogue} locally. Thus the higher the number of nodes receiving t_{rogue} first is, the higher the chances consensus will be formed with t_{rogue} are. We see here that the attack is probabilistic, so success cannot be guaranteed.



The red nodes have accepted $t_{genuine}$ locally and the blue nodes have accepted t_{rogue} locally.
The nodes with a green circle lie on the border of the two network parts. They received both $t_{genuine}$ and t_{rogue}, while nodes with no green circle only received one of them.

Figure 4.1: Network effects of the attack

4.1 Network effects

A node receiving a transaction will check it for well-formedness etc.. If these checks pass, the transaction is accepted by the node to the local transaction pool and relayed in the Bitcoin network. If, for some reason, a check fails, the transaction is discarded as invalid and, in consequence, not relayed. $t_{genuine}$ and t_{rogue} conflict with each other, so that a node, after having received one, the other will fail a check.

This means after receiving one transaction, the other one is discarded by the node. The two transactions thus are competing directly in the network for nodes, until each node either has accepted $t_{genuine}$ or t_{rogue} locally. The two transactions split the network in two parts. Since a node, having received $t_{genuine}$ first, will not relay t_{rogue} to its neighbours (and vice versa), a node with neighbours only in one part only sees only one of transactions in the network.

This creates 3 classes of nodes:

- Nodes that have only seen $t_{genuine}$
- Nodes that have seen both transactions, lying on the border of the two parts
- Nodes that have only seen t_{rogue}

See figure 4.1 for a graphic depiction of the three classes of nodes.

4.2 Detectability

A node receiving only one of the two transactions cannot detect our attack, since the transaction is indistinguishable from a regular transaction (to either the victim or someone else). For two classes of nodes mentioned in section 4.1, our attack is not detectable.

Nodes that do receive both transactions are able to detect the attack, which is what we use in chapter 7.

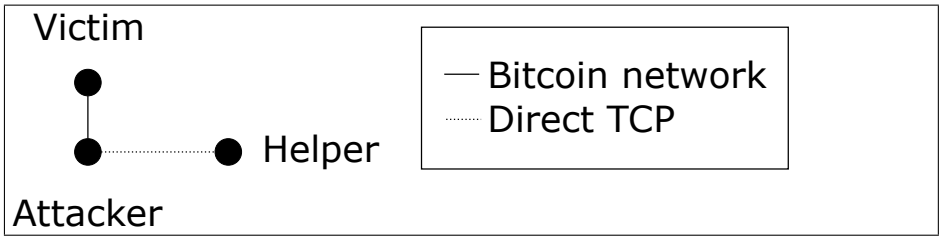


Figure 4.2: Attack setup

We can formulate another goal for the attack, namely that the attack should not be detectable for the victim. However, the original Bitcoin client software does not warn the user anyway, so this goal is not crucial to the success of our attack.

4.3 Design

A single node can only accept and relay either $t_{genuine}$ or t_{rogue} locally, so this attack requires a second attacker-controlled node in the Bitcoin network.

We call these two nodes the attacker and the helper. In order for the goals of the attack to be reached, the node inserting $t_{genuine}$ into the network should be connected directly to the victim. We decide that the attacker should be this node.

If the attacker is connected to nodes other than the victim, they help to spread $t_{genuine}$ in the network, which works directly against the second goal, so the attacker should only be connected to the victim.

The attack should also not be detectable on the victim, which means that the helper should be far away from the victim, in network hops. Since network formation is random (see chapter 3), the simple heuristic we use is to disconnect the helper from the victim, if they are connected.

See figure 4.2 for a graphic depiction of the attack setup.

After the network is set up as described, the attack continues as follows. The attacker creates the two transactions ($t_{genuine}$, t_{rogue}). They relay $t_{genuine}$ via Bitcoin network to the victim, and send t_{rogue} to the helper via a direct TCP connection. The helper then starts to relay t_{rogue} via Bitcoin network.

For experimental purposes, we add a parameter *delay* to the attack. Let the time when $t_{genuine}$ is relayed in the Bitcoin network (by the attacker) be $time_{genuine}$, and let the time when t_{rogue} is relayed in the Bitcoin network (by the helper) be $time_{rogue}$. *delay* is defined as the time difference between $time_{genuine}$ and $time_{rogue}$.

$$delay = time_{rogue} - time_{genuine}$$

Note that *delay* can be negative.

In addition to the delay, the attack requires multiple other parameters. The attack is initialised via command line, so several parameters are supplied as command line arguments, while others are supplied in textfiles which are read when needed.

Supplied via command line are the following parameters; the format of the command can be seen in figure 4.3.

Attacker:
`./bitcoind makedoublespend $address_1 $address_2 $amount $victim_ip $delay`
Where $address_1 and $address_2 are the Bitcoin addresses of the victim and the doublespend-target, in that order, $amount is a float representing the amount of Bitcoins to doublespend, $victim_ip is the IPv4 address of the victim in dot-decimal notation and $delay is an integer representing the delay to be used in the attack, in milliseconds. Note that $delay can be negative.
Helper:
`./bitcoind doublespendlisten $port $attacker_ip $victim_ip`
Where $port is the port the helper should listen on and $attacker_ip and $victim_ip are the IPv4 address of the attacker and the victim, respectively, in dot-decimal notation.

Figure 4.3: Attack commands

Attacker:

- Bitcoin address of the victim
- Doublespend address
- Amount of Bitcoins to doublespend
- IPv4 address of the victim
- Delay

Helper:

- TCP port to listen on
- IPv4 address of the attacker
- IPv4 address of the victim

Supplied via textfile are the following parameters (on the attacker only); the format of the textfile can be seen in figure 4.4.

- IPv4 addresses of the helpers
- TCP ports the helpers are listening on

Note that, since the helper listens for a connection, the helper command (figure 4.3) needs to be executed first on the helper, followed by the attacker command on the attacker.

4.4 Implementation

As described in section 4.3, the attack uses several machines. We implement the attack in the same software, i.e. the attacker and the helper nodes both run the same modified Bitcoin client software, but we present the behaviour of the two nodes separately.

Location of the file: $bitcoin/helpers.txt. Format of helpers.txt:

```
file = (line'\n')*line | ''
line = '$ip $port'
```

Where each line represents one helper node. $ip is the IPv4 address of a helper node in dot-decimal notation and $port is the port this helper is listening on.

Figure 4.4: Helpersfile

4.4.1 Attacker side

When given the attack command, the modified Bitcoin client software checks the user supplied parameters and stops the attack, if there is a problem. Otherwise, the attacker disconnects from all its peers in the Bitcoin network and prevents the creation of new connections. To this end, the function creating new connections, which runs in a separate thread, is modified.

The attacker then connects to the victim node in the Bitcoin network via its IPv4 address. If this fails, the attack is aborted.

After the network setup phase, the attacker creates the two two transactions ($t_{genuine}$, t_{rogue}). This is done like the creation of regular transactions, except that two transactions are created in parallel (sharing randomness etc.). The two transactions differ in the recipient address, and related fields, but otherwise are equal.

After both transactions are created, the attacker tests whether *delay* is positive or negative.

If $delay \geq 0$, $t_{genuine}$ is added to the local transaction pool and relayed via Bitcoin network. Additionally, a time measurement ($time_{start}$) is taken. If a problem arises while adding t_{geuine} to the local transaction pool, the attack is aborted.

For each helper node, a TCP connection is opened consecutively. Another time measurement is taken, and the adjusted delay is calculated by considering the time elapsed since t_{start}. The adjusted delay and t_{rogue} are then sent to the helper via TCP, and then the TCP connection to the helper is closed again.

If $delay < 0$, for each helper node, a TCP connection is opened consecutively. A *delay* of 0 and t_{rogue} are sent to the helper via TCP, and then the TCP connection to the helper is closed again.

The attacker then sleeps for $-delay$ (*delay* is negative), adds $t_{genuine}$ to the local transaction pool and relays it via Bitcoin network. If a problem arises while adding $t_{genuine}$ to the local transaction pool, the problem is reported, but t_{rogue} is already in the network and the attack cannot be aborted anymore.

The attacker part in algorithm notation can be seen in figure 4.5.

4.4.2 Helper side

When given the listen command, the helper disconnects from both the attacker and the victim in the Bitcoin network, if it is connected to them. They then listen for a TCP connection.

When a TCP connection is opened, the helper receives *delay* and t_{rogue} and then sleeps for *delay*.

```
error ← CHECK(parameters)
if error then
    return "Error"
end if
disconnect and prevent new connections
error ← BITCOINCONNECT(victim)
if error then
    return "Error"
end if
(t_genuine, t_rogue) ← CREATETRANSACTIONS(addresses, amount)
if delay ≥ 0 then
    error ← ADDTOLOCALPOOL(t_genuine)
    BITCOINRELAY(t_genuine)
    if error then
        return "Error"
    end if
    time_start ← GETTIMEOFDAY()
    for all helper ∈ helpers do
        TCPCONNECT(helper)
        time_now ← GETTIMEOFDAY()
        delay_adjusted ← delay − (time_now − time_start)
        delay_adjusted ← max(0, delay_adjusted)          ▷ so it is ≥ 0
        TCPSEND(t_rogue, delay_adjusted)
        TCPDISCONNECT(helper)
    end for
else                                                     ▷ delay < 0
    for all helper ∈ helpers do
        TCPCONNECT(helper)
        TCPSEND(t_rogue, 0)
        TCPDISCONNECT(helper)
    end for
    SLEEP(−delay)                                        ▷ delay is negative
    error ← ADDTOLOCALPOOL(t_genuine)
    BITCOINRELAY(t_genuine)
    if error then
        return "Error"                                   ▷ Attack is already on its way
    end if
end if
```

Figure 4.5: Attacker algorithm

```
if connected to attacker or victim then
    disconnect and prevent new connections to either one
end if
TCPListen(port)
(t_rogue, delay) ← TCPRead
Sleep(delay)
error ← AddToLocalPool(t_rogue)
if error then
    return "Error"
end if
BitcoinRelay(t_rogue)
```

Figure 4.6: Helper algorithm

The helper adds t_{rogue} to the local transaction pool and relays it via Bitcoin network. If a problem arises while adding t_{rogue} to the local transaction pool, the transaction is not relayed and the problem is reported. This can abort the attack, but if another helper doesn't encounter a problem, the attack continues without this helper participating.

The helper part in algorithm notation can be seen in figure 4.6.

Chapter 5

Experimental setup

In this chapter, we present how we set up our experiments, what data we measure and how we evaluate it.

We conduct several different experiments, so there are several setups. We present each setup independently, although there might be some overlap between them.

5.1 Required resources

All the experiments require an internet connection. Additionally, some software might be used to run an experiment:

- Scripts are written in Python version 2.7.2+
- The modified Bitcoin client software is based on the Bitcoin client software version 50000. For a list of required libraries for the Bitcoin client software, please consult the INSTALL file of the Bitcoin client software distribution.
- LaTeXis generated with pdfTeX version 3.1415926-1.40.10

5.1.1 Time measurements of the consensus procedure

For this experiment, a script running on a single local machine is used.

5.1.2 Connection measurements of a Bitcoin node

Several experiments are conducted. Each experiment is running a modified Bitcoin client software on a single machine. Each experiment is conducted in one of the following configurations:

- A local machine is used
- A rented machine in Virginia, USA is used

5.1.3 Attack measurements

Several experiments are conducted with many different configurations, and not all of them are listed here. For a detailed description of the configuration used in each experiment, please refer to chapter 6. All experiments share some characteristics, which are listed here:

An experiment involves running a modified Bitcoin client software on several machines.

The rented machines are at one of the following locations: Virginia, USA; Oregon, USA; Singapore; Tokyo, Japan; São Paulo, Brazil.

For each experiment the following setup is used:

- The attacker runs on a local machine
- One or more helpers run on a rented machine
- A victim runs on a rented machine, in a different location than the helpers
- Five observers, each on a rented machine, run on one machine in each location

5.2 Measurement methodology

5.2.1 Measured data

Time measurements of the consensus procedure

The data gathered for this experiment consists of the following:

- Timestamp of the creation of a block in the Bitcoin system

This data is saved in the blockchain and does not need to be measured. It is accessed via `https://blockexplorer.com`. A script is used to collect the data and analyse it (see appendix B).

Connection measurements of a Bitcoin node

The data gathered for this experiment consists of the following:

- Number of connections a Bitcoin node has, with a timestamp

This data is gathered by a modified Bitcoin client software (see appendix A). The data gathered this way is processed for easier plotting by a script (see appendix B).

Attack measurements

The data gathered for this experiment consists of the following:

- Location of the victim
- Location of the helpers
- Number of peers of the victim

- Number of helpers used
- Delay used
- Transaction id of a transaction sent through the Bitcoin network from the attacker, with a timestamp
- Transaction id of a transaction sent through TCP directly from the attacker, with a timestamp
- Transaction id of a transaction sent through the Bitcoin network from each of the helpers, with a timestamp
- Transaction id of a transaction received through the Bitcoin network on the victim, with a timestamp
- Transaction id of a transaction received through the Bitcoin network on each observer, with a timestamp
- Number of the newest block found in the blockchain before the start of the experiment
- Block data for each block from the starting block (data item above) until enough block data is acquired (see below what that means), namely a timestamp of the creation of each block and the transaction ids in each block

Some data can be chosen and has to be noted down, i.e. the location of the victim and the location of the helpers. Number of connections data can be chosen, but is also gathered by a modified Bitcoin client software (see appendix A).

The transaction id data with respective timestamps is gathered by a modified Bitcoin client software on the respective machine (see appendix A). The data gathered this way can be copied to the local machine for evaluation by a script (see appendix B).

Some data can be chosen and has to be noted down, but noting it in a certain way enables a script to parse it. Which data to note down in a certain way and how to note it down is shown here:

- Number of helpers used in 'helpers.txt'
- Number of the newest block found in 'blockcount.txt'
- Delay used (in ms) in 'delay.txt'

Transactions are grouped in pairs ($t_{genuine}$, t_{rogue}) and enough block data means that from each pair, one of the transactions is found in the block data. Block data is saved in the blockchain and does not need to be measured. This data is accessed via `https://blockexplorer.com`.

A script is used to collect the necessary block data, combine it with the other data (if stored in the suggested format) and analyse it (see appendix B).

5.2.2 Data evaluation

In this section, we describe how data was aggregated and combined.

Time measurements of the consensus procedure

The average, standard deviation, minimum and maximum of the timestamp data are calculated.

Connection measurements of a Bitcoin node

This data is used directly.

Attack measurements

Transaction id data from a helper is used to determine transaction ids that didn't encounter a problem (see section 5.2.3), namely the ones that are relayed on the helper. Based on this data, transaction id data from the attacker is used to determine the pair-relationship ($t_{genuine}$, t_{rogue}), for which t_{rogue} didn't encounter a problem on the helper. The relationship is formed by timestamp, e.g. a transaction relayed through the Bitcoin network and the next transaction relayed through TCP are paired.

Transaction id data from the victim is then used to determine the time difference between the arrival of the two paired transactions on the victim. Note that it could be that one transaction didn't arrive at all.

The average of the absolute of the time differences is calculated, as well as the number of times one of the transactions didn't arrive at all.

The same process we apply to the victim data is repeated for each observer. In addition, the number of observers that see t_{rogue} is determined. The fraction of observers receiving t_{rogue} is calculated.

Block data is used to determine which of the two paired transactions is confirmed by the Bitcoin network. The number of successful doublespend-attacks is determined, with successful being defined as when t_{rogue} gets confirmed.

Also extracted from the block data are the block number of confirmation for each pair, as well as the time delay between sending and confirming the transaction. The average of the absolute of the time delay is calculated.

5.2.3 Problems and solutions

t_{rogue} is not accepted on the helper node

After several transactions, starting the attack will result in t_{rogue} being rejected on the helper. The problem is solved by waiting for a new block to be found, which will clear some inconsistencies if downloaded. However, this can make measuring quite tedious. A way to reduce the chance of this problem occurring is to have a lot of small transactions sent to the attacker before starting the attack, e.g. instead of sending 5 Bitcoins to the attacker in preparation for the attack, one should send 50 times 0.1 Bitcoins.

Wallet of the attacker is corrupted

After executing the attack several times, the local data of the attacker can get corrupted. The problem can be solved by using a backup from a consistent state (suggested: a copy of the whole directory ~/.bitcoin). The backup should be updated from time to time, since the suggested directory also contains the blockchain data, which has to be downloaded if its not present.

Chapter 6

Experimental results

In this chapter, we present the results of our experiments introduced in chapter 5. We first present the pure data and follow up with an interpretation of the data.

6.1 Measurement results

6.1.1 Time measurements of the consensus procedure

We analyse the whole blockchain at the time of writing, consisting of 173563 blocks. The result can be found in table 6.1. Time refers to the difference in timestamps from a block to the next one. The data can be found in ./logs/block-analysis/.

6.1.2 Connection measurements of a Bitcoin node

We measure two data sets, one in Switzerland and one in Virginia. The result for the measurement in Switzerland can be seen in figure 6.1 and the result from the measurement in Virginia can be seen in figure 6.2. Note that the measurements are over time periods of different length. The data can be found in ./logs/connection/.

Field	Value
Analysed blocks	173563
Mean (time)	595 seconds (9 minutes, 55 seconds)
Standard deviation (time)	854.32 seconds (14 minutes, 14.32 seconds)
Minimum (time)	0 seconds
Maximum (time)	90532 seconds (25 hours, 8 minutes, 52 seconds)

Table 6.1: Time measurement of the consensus procedure

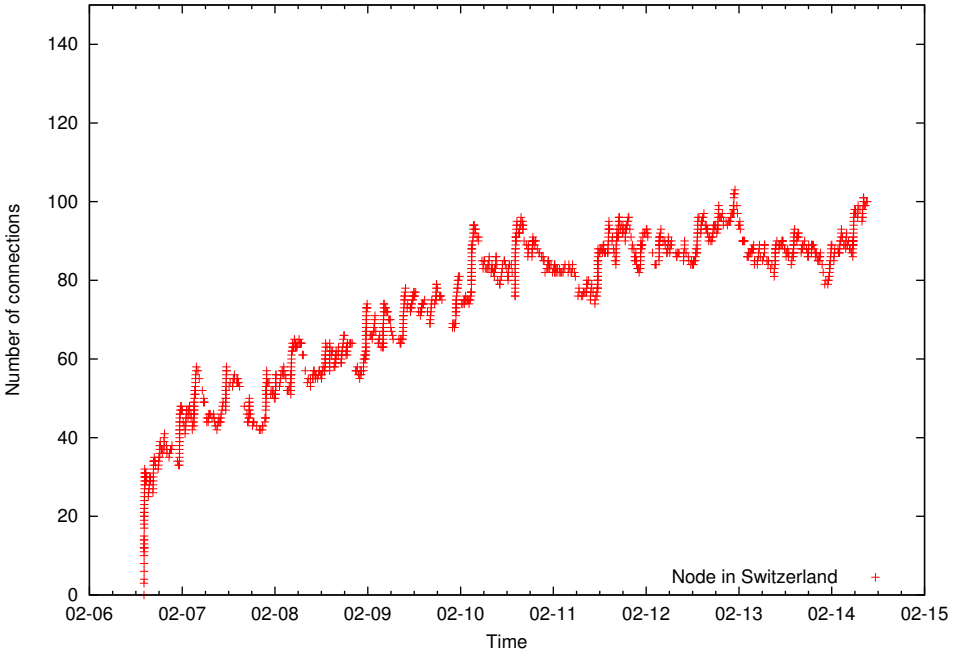


Figure 6.1: Number of connections in Switzerland

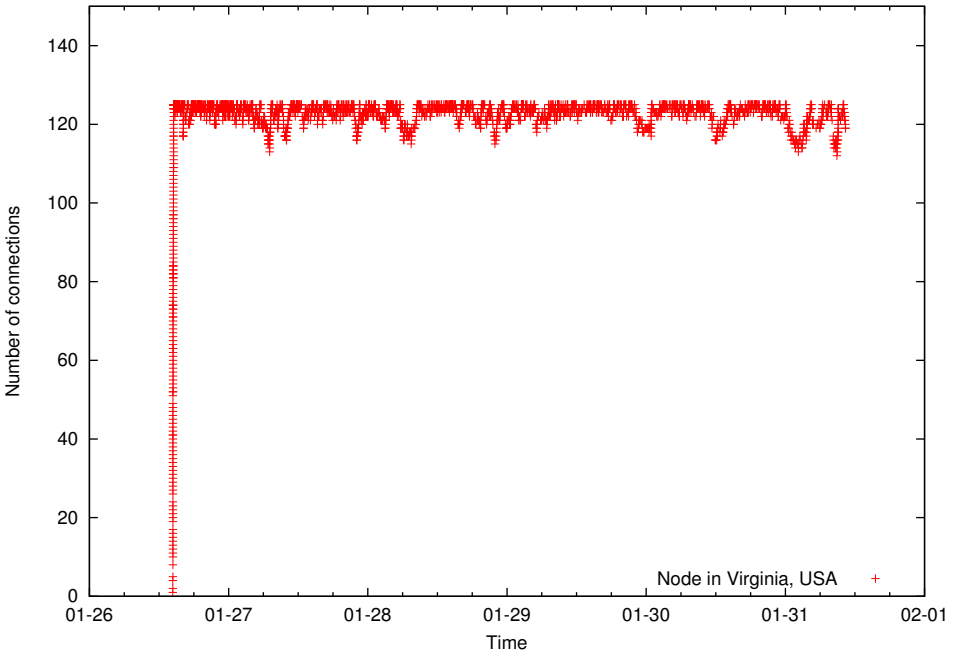


Figure 6.2: Number of connections in Virginia

6.1.3 Attack measurements

Mass measurements

For each measurement, the victim is on one of the rented servers, which one is denoted in the figures. Each of the helpers is on a different server than all the other helpers used in this experiment, if there is more than one helper.

A helper in a measurement is fully connected, they have 125 connections to other Bitcoin nodes. The number of connections of the victim is mentioned in the figures.

Each measurement consists of 10 attacks. We use one or two helpers for the measurement, how many is indicated in the figures.

Figures 6.3, 6.4, 6.5 and 6.6 show the number of successful attacks, defined as those attacks where t_{rogue} gets confirmed by the system.

Figures 6.7, 6.8, 6.9 and 6.10 show the number of times the pair ($t_{genuine}$,t_{rogue}) is *not* received on the victim, which corresponds to the attack not being detectable on the victim.

This only includes immediate receives. E.g. if t_{rogue} gets confirmed by the system, it will be distributed to the victim in a block. If the victim receives the transaction for the first time in a block, it will be counted as not received.

Figures 6.11, 6.12, 6.13 and 6.14 show the number of successful attacks in which the pair ($t_{genuine}$, t_{rogue}) is not received by the victim, which corresponds to a successful attack that is not detectable on the victim. Again, only direct receives are counted.

Figures 6.15, 6.16, 6.17 and 6.18 show the fraction of all t_{rogue} transactions that are received by all observers combined. Again, only direct receives are counted. Each t_{rogue} is counted individually, so with 10 attacks per measurement and 5 observers, one observed t_{rogue} on one observer evaluates to a fraction of 0.02.

The data can be found in ./logs/mass_measurement/.

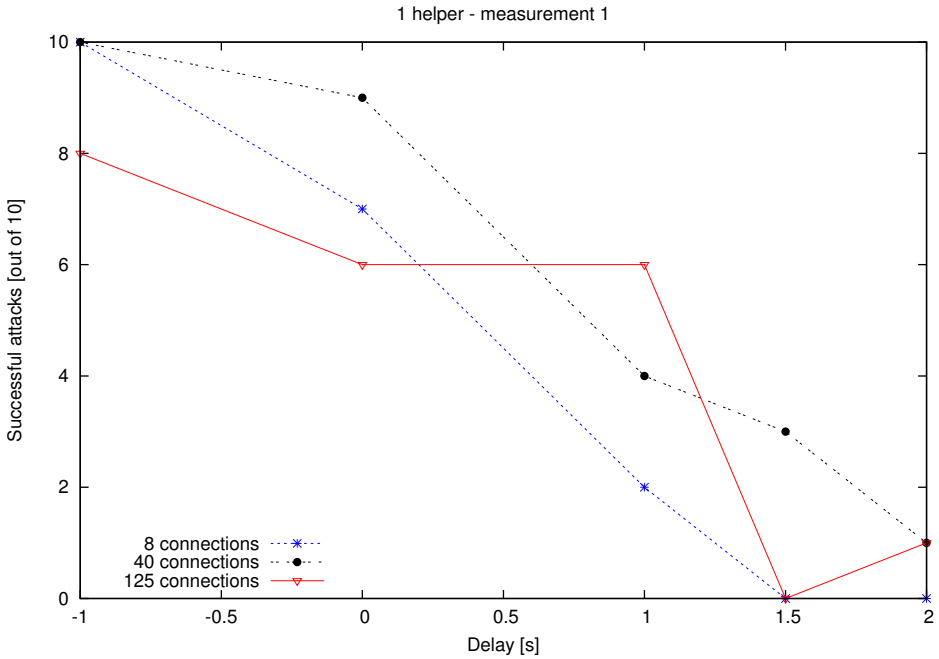


Figure 6.3: Successful attacks on victim in Virginia (1 helper)

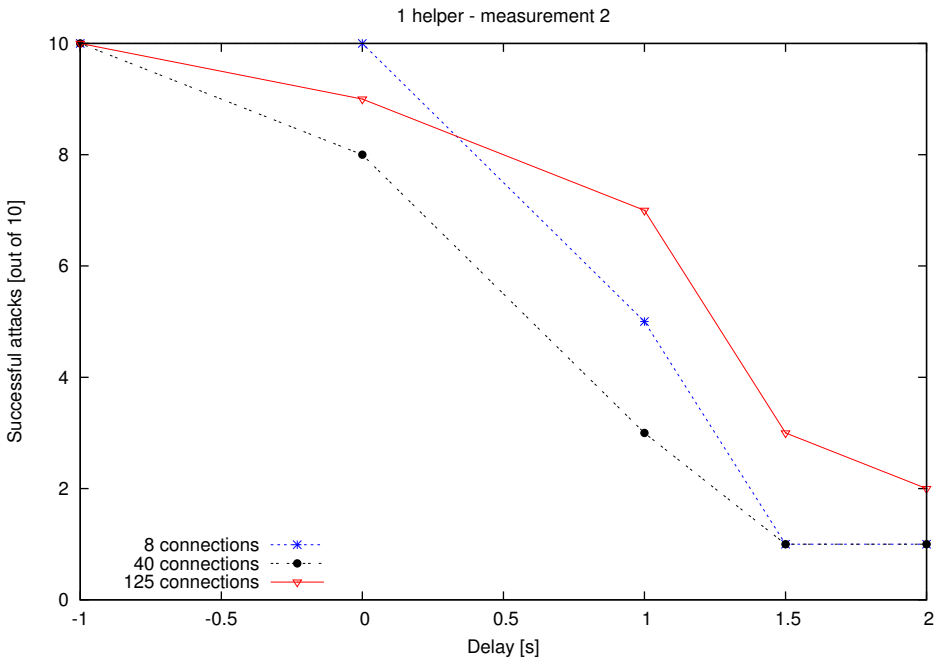


Figure 6.4: Successful attacks on victim in Oregon (1 helper)

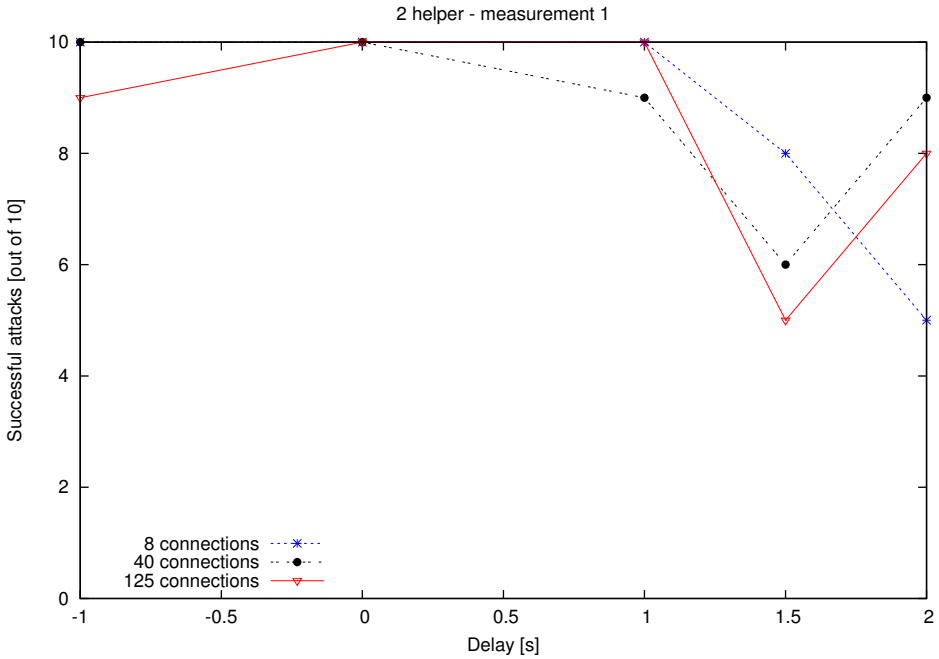


Figure 6.5: Successful attacks on victim in Singapore (2 helpers)

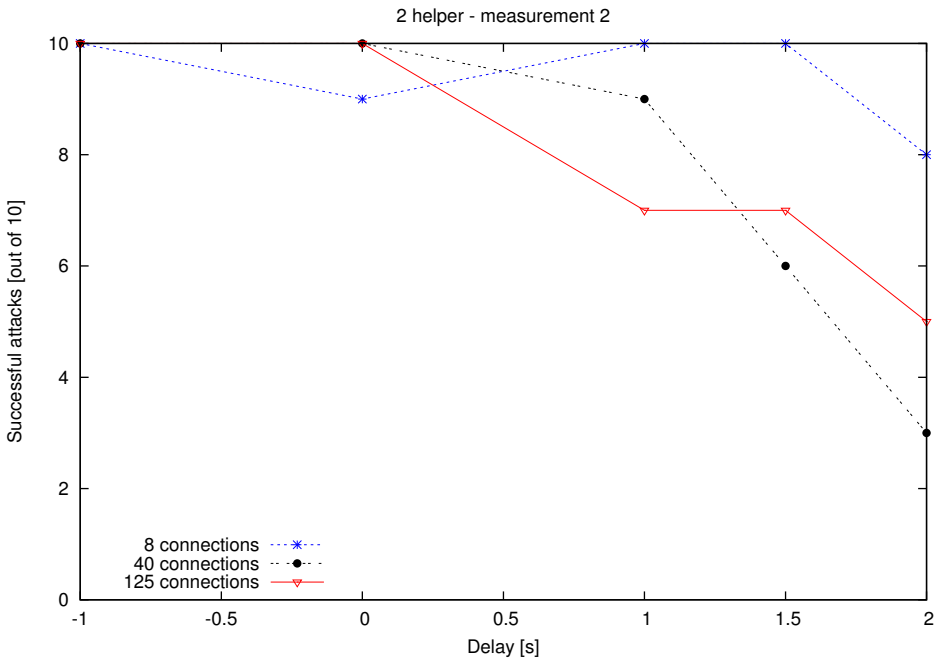


Figure 6.6: Successful attacks on victim in Tokyo (2 helpers)

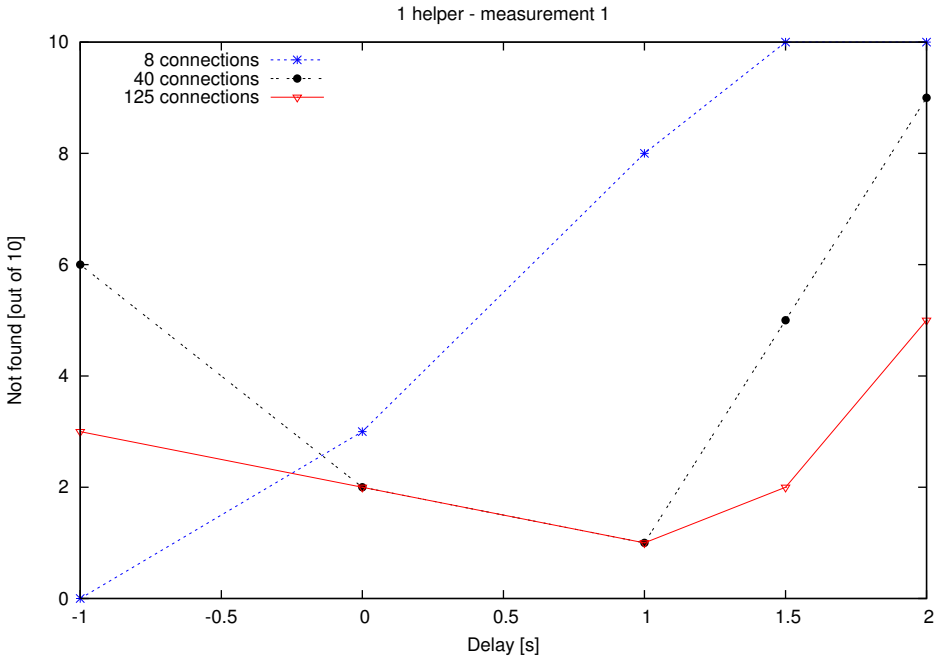


Figure 6.7: Not found on victim in Virginia (1 helper)

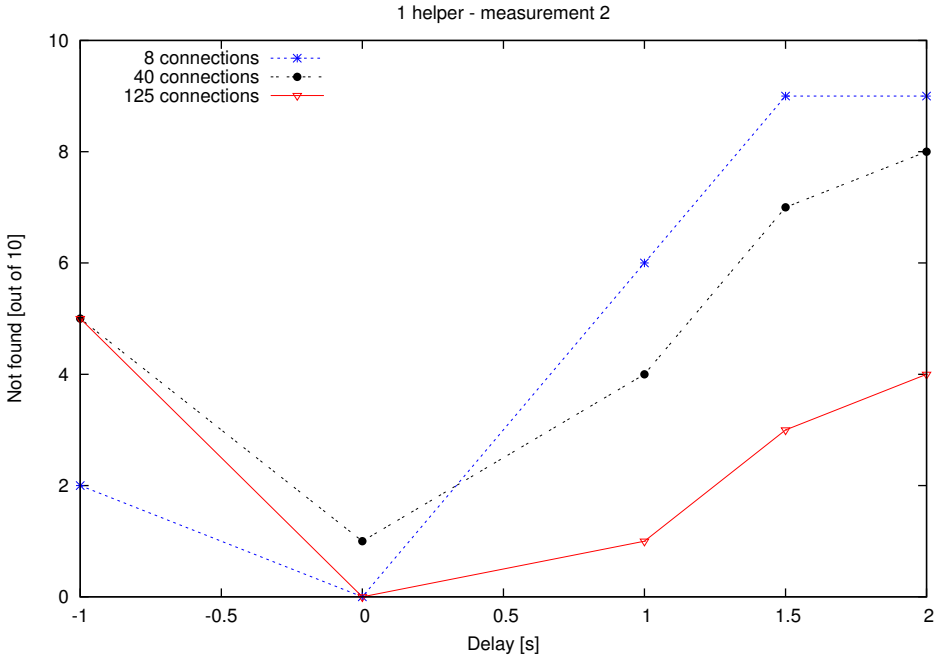


Figure 6.8: Not found on victim in Oregon (1 helper)

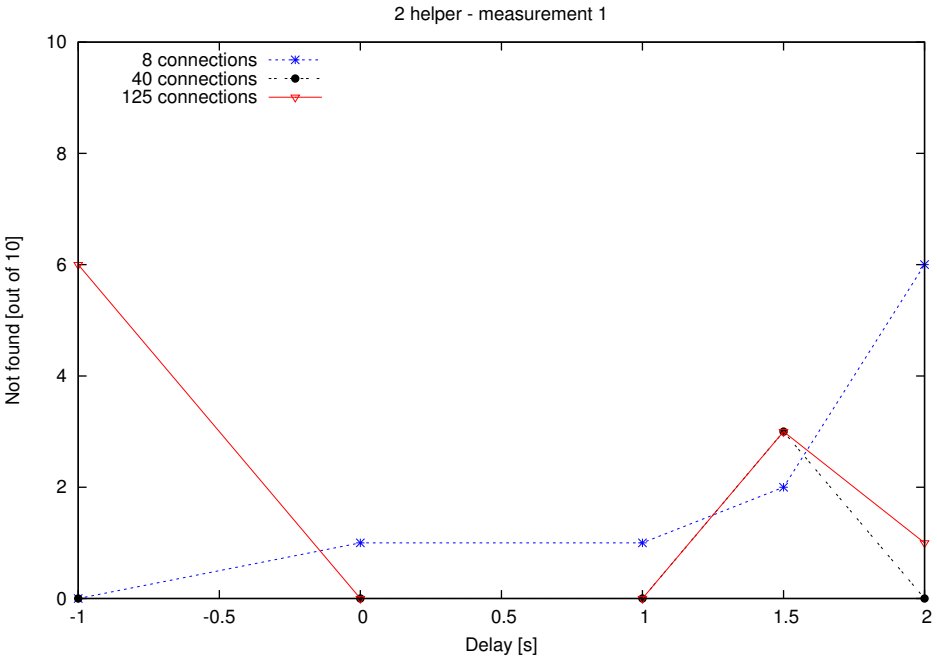


Figure 6.9: Not found on victim in Singapore (2 helpers)

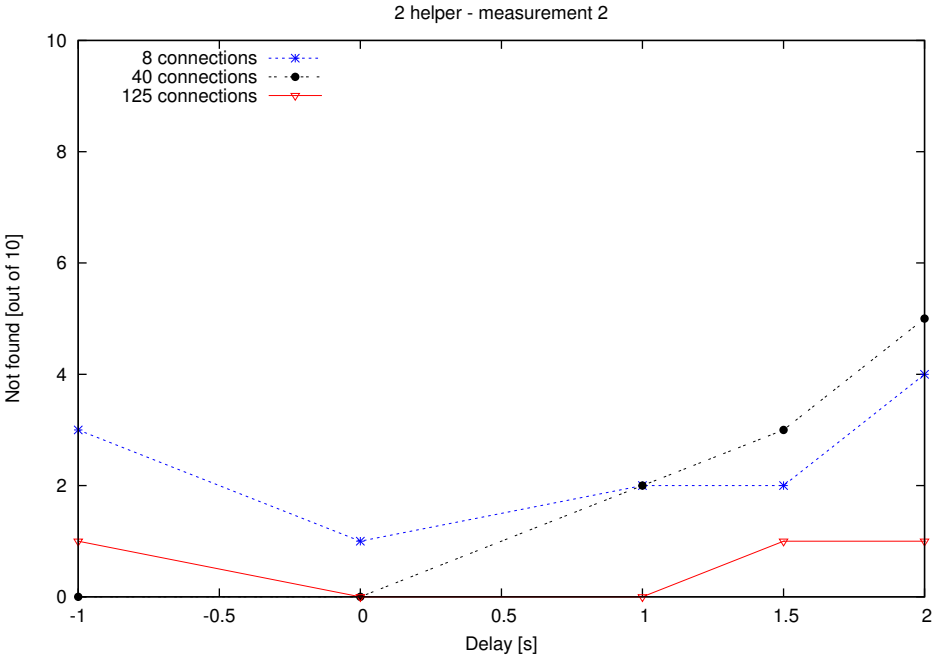


Figure 6.10: Not found on victim in Tokyo (2 helpers)

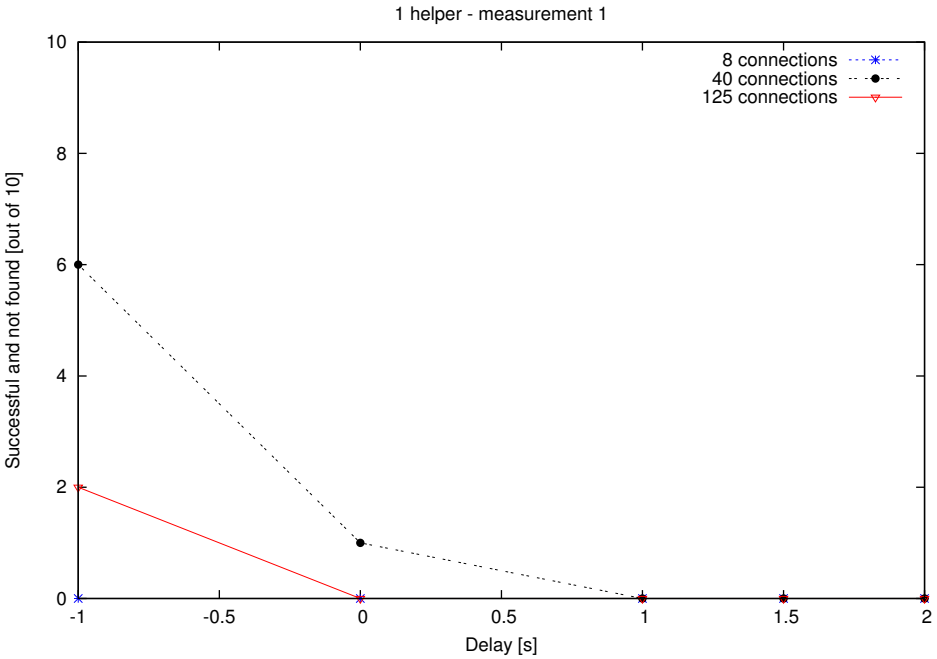


Figure 6.11: Successful and not found attacks on victim in Virginia (1 helper)

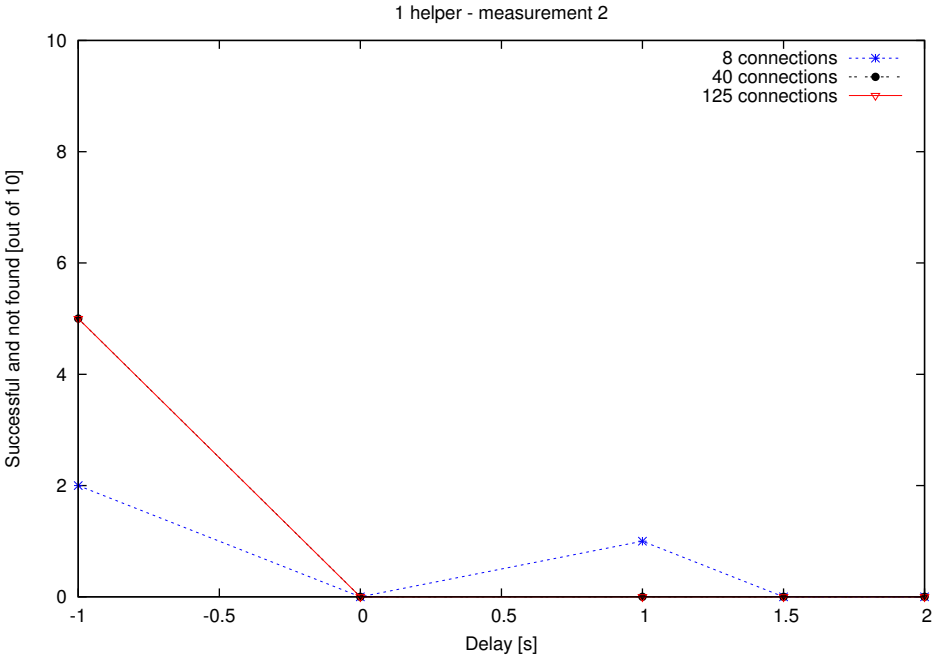


Figure 6.12: Successful and not found attacks on victim in Oregon (1 helper)

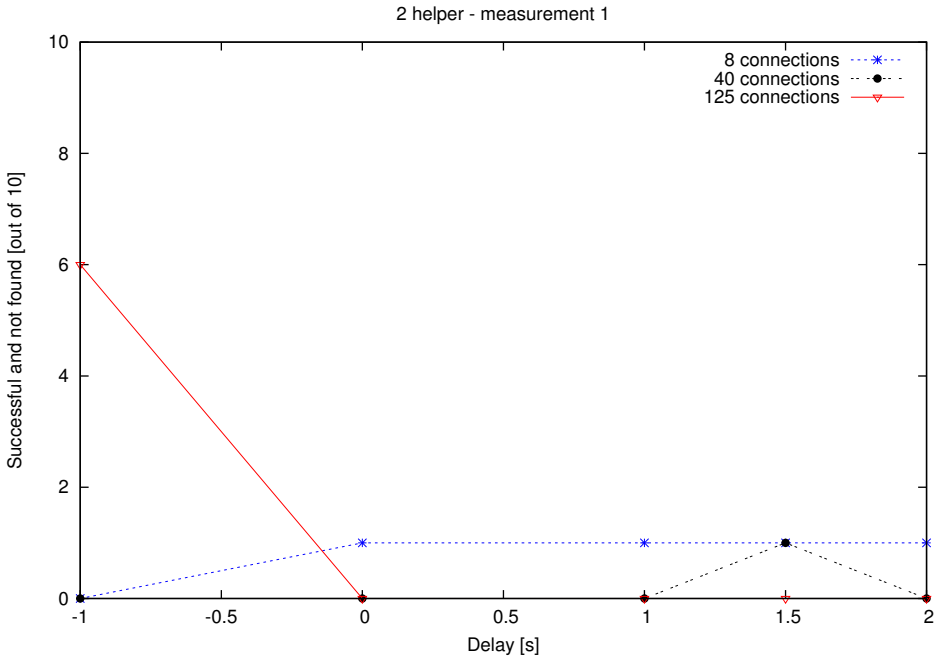


Figure 6.13: Successful and not found attacks on victim in Singapore (2 helpers)

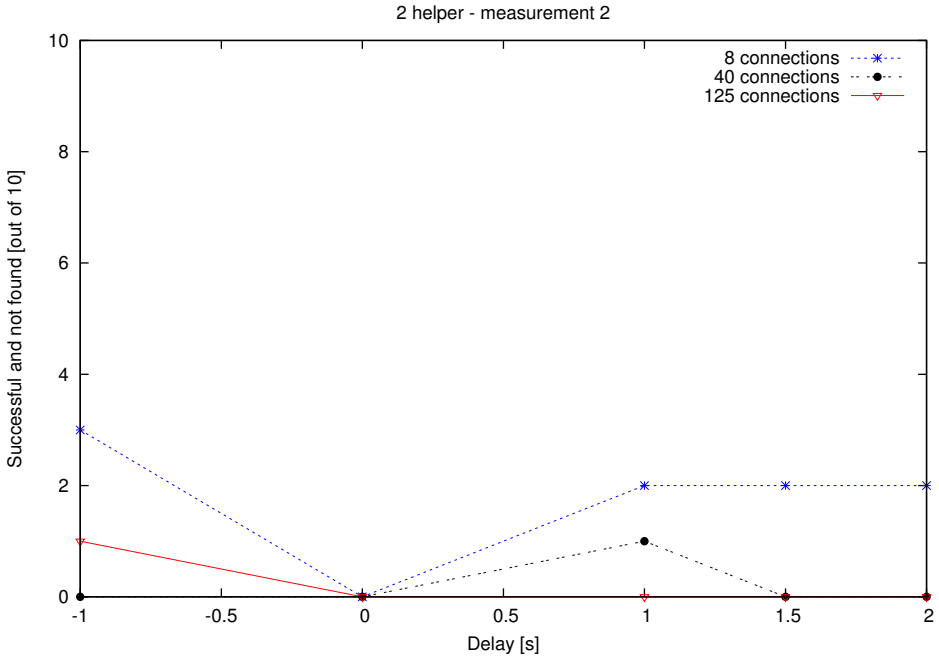


Figure 6.14: Successful and not found attacks on victim in Tokyo (2 helpers)

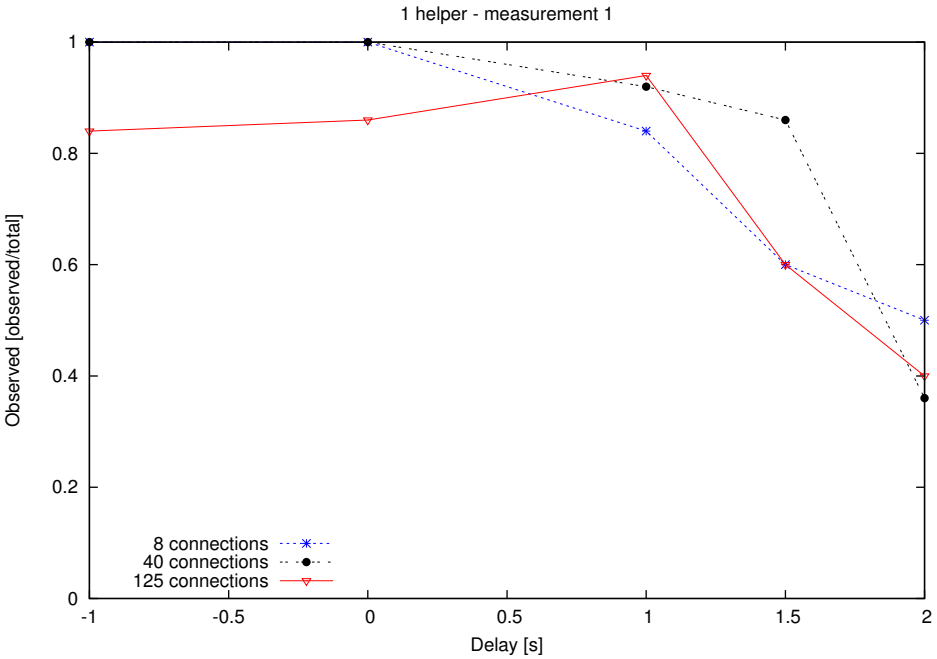


Figure 6.15: Observed attacks on victim in Virginia (1 helper)

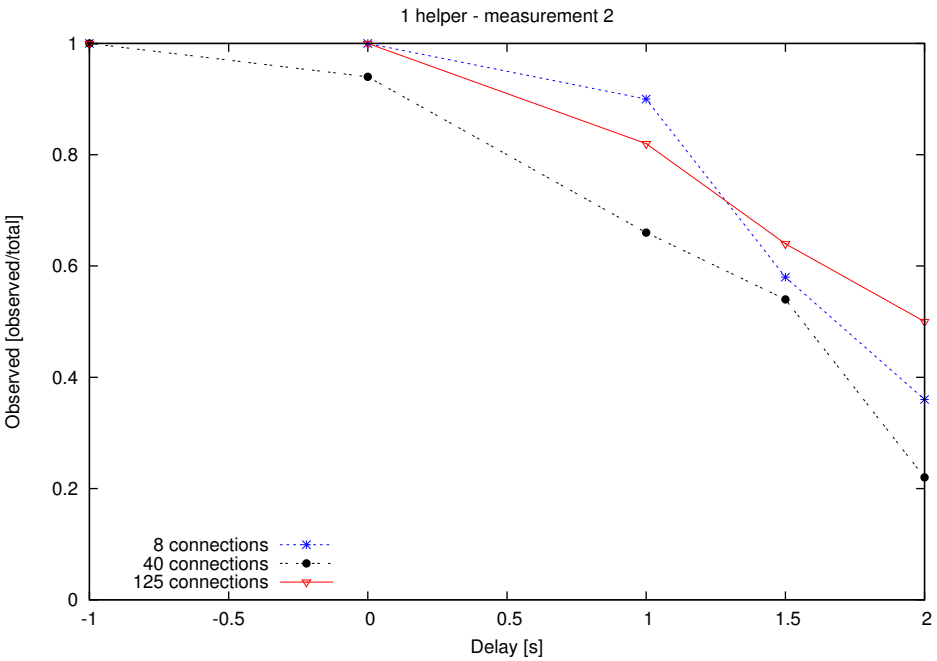


Figure 6.16: Observed attacks on victim in Oregon (1 helper)

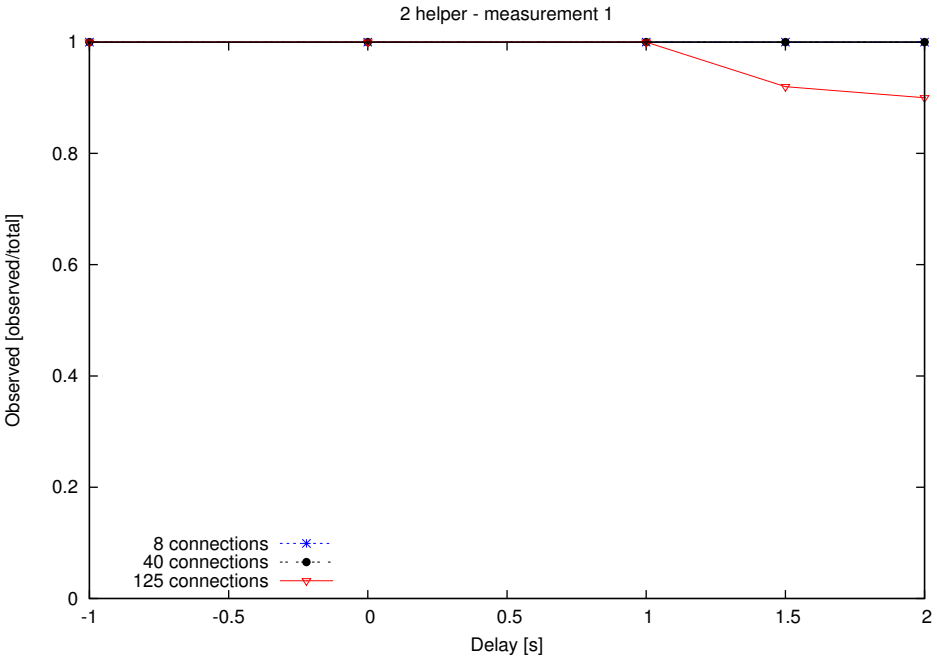


Figure 6.17: Observed attacks on victim in Singapore (2 helpers)

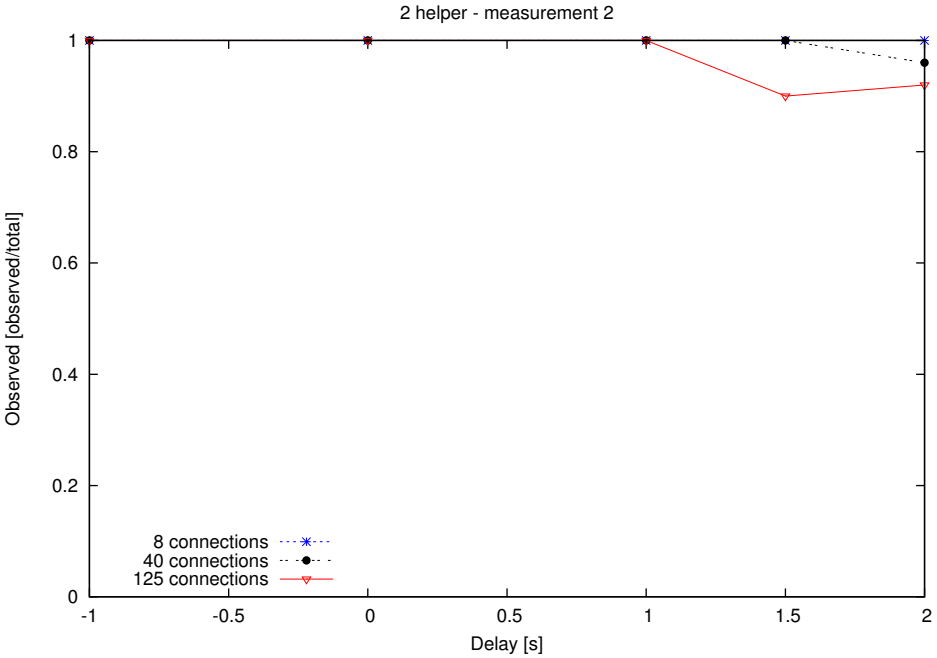


Figure 6.18: Observed attacks on victim in Tokyo (2 helpers)

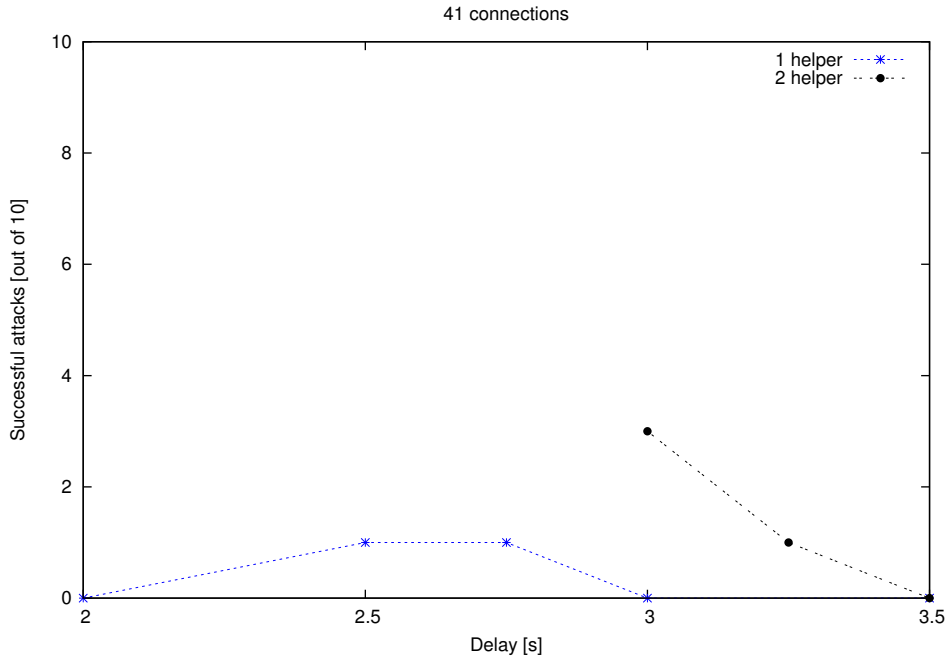


Figure 6.19: Successful attacks on victim with 41 connections

Exploring measurements

For these measurements, we chose 3 number of connections of the victim, derived from our connection measurement (figure 6.1). The 3 used values are:

125 connections full connectivity

83 connections the median of the measured number of connections

41 connections is as far from the median as 125 is ($41 = 83 - (125 - 83)$)

For each measurement, the victim is in Singapore, one of the helpers is in Tokyo, and the other helper, if present, is in Virginia.

A helper in a measurement is fully connected, they they have 125 connections to other Bitcoin nodes.

Each measurement consists of 10 attacks. We use one or two helpers for the experiments, how many is indicated in the figures.

Figures 6.19, 6.20, 6.22 and 6.21 show the results for a victim with 41 connections.

Figures 6.23, 6.24, 6.26 and 6.25 show the results for a victim with 83 connections.

Figures 6.27, 6.28, 6.30 and 6.29 show the results for a victim with 125 connections.

Note that the measurements use different delays.

The data can be found in ./logs/exploring/.

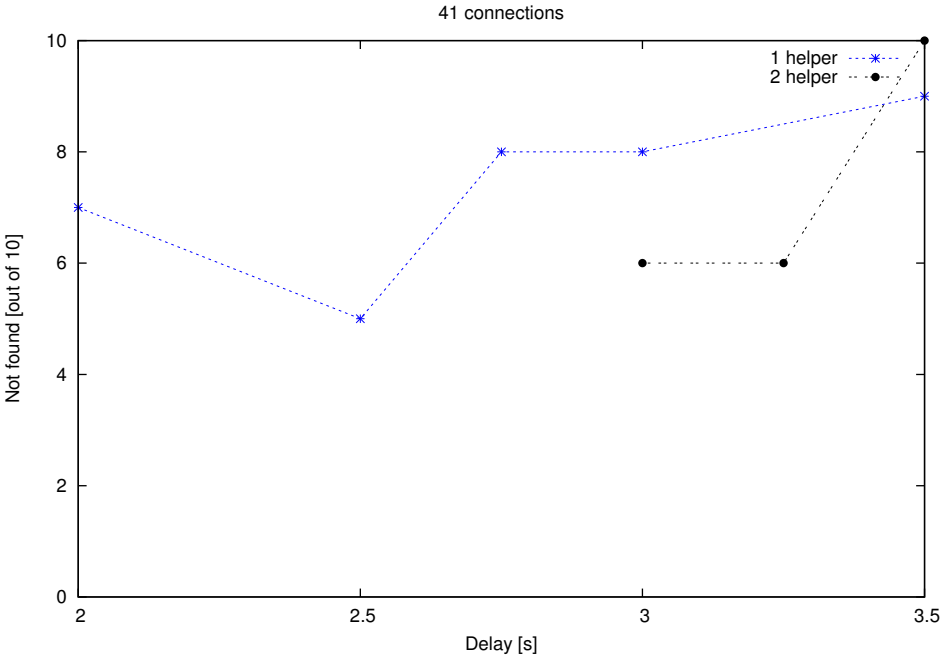


Figure 6.20: Not found on victim with 41 connections

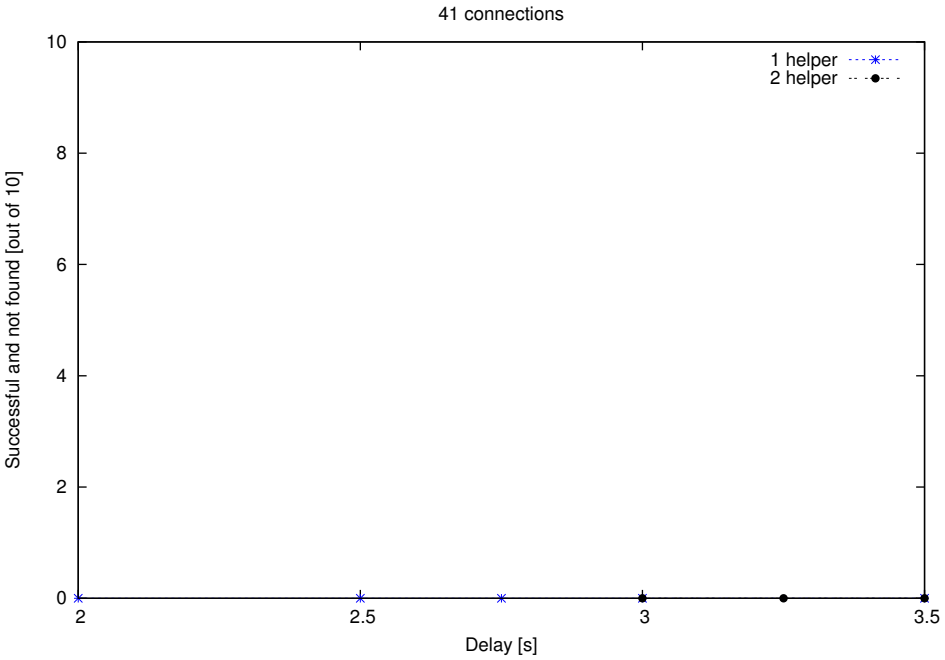


Figure 6.21: Successful and not found attacks on victim with 41 connections

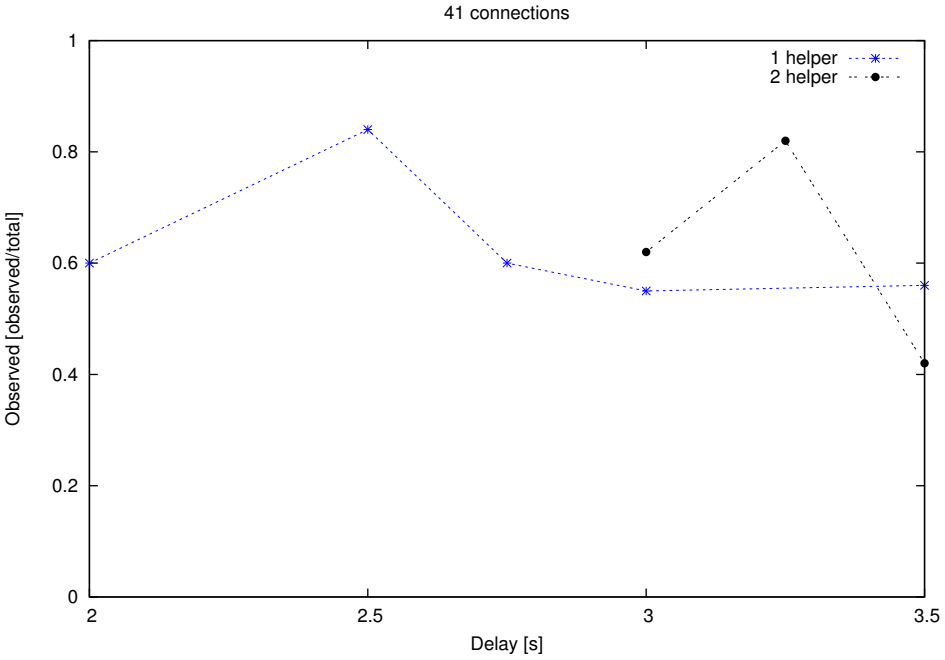


Figure 6.22: Observed attacks on victim with 41 connections

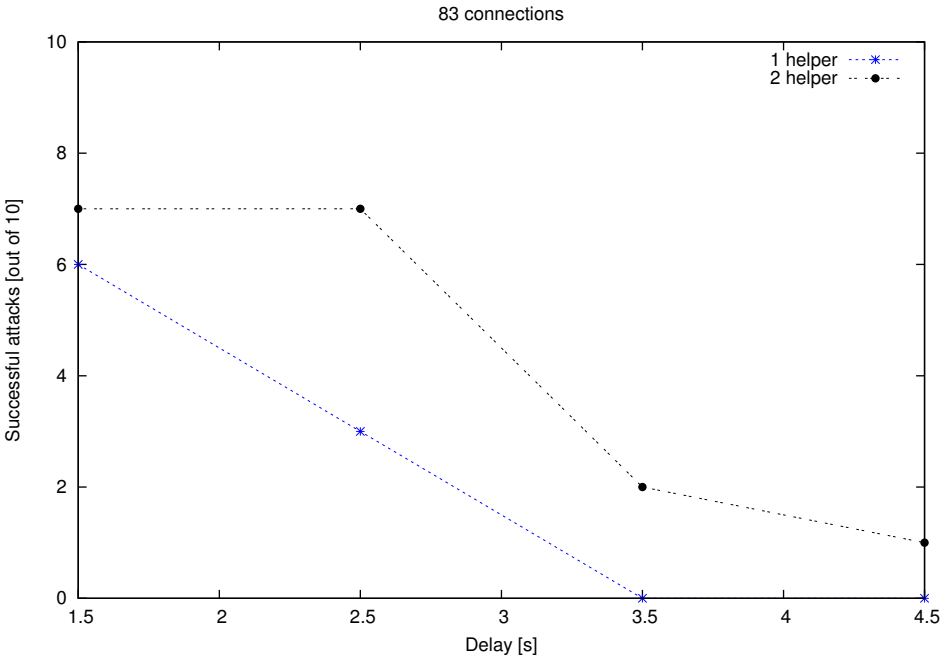


Figure 6.23: Successful attacks on victim with 83 connections

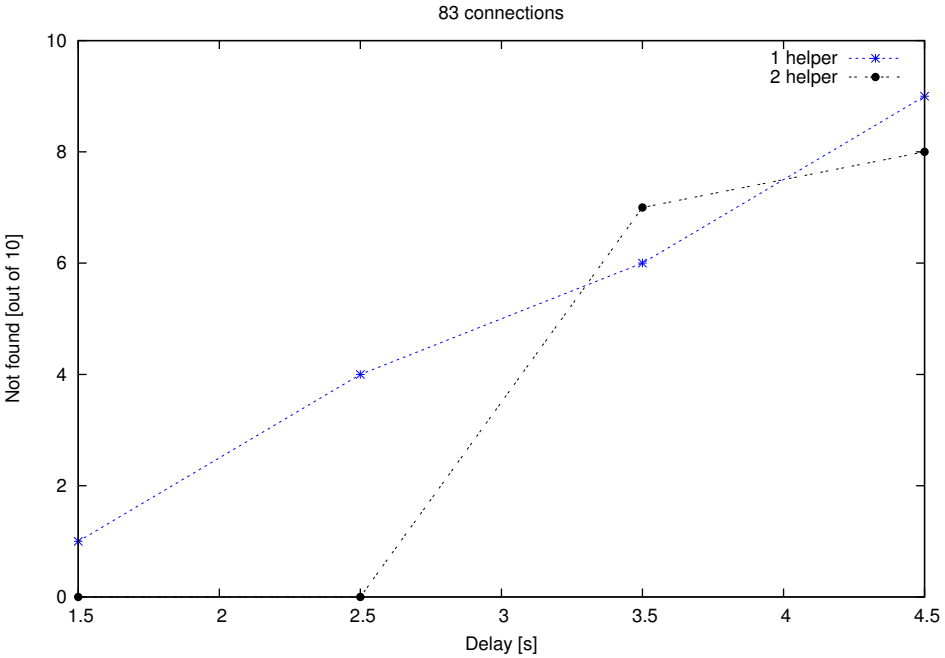


Figure 6.24: Not found on victim with 83 connections

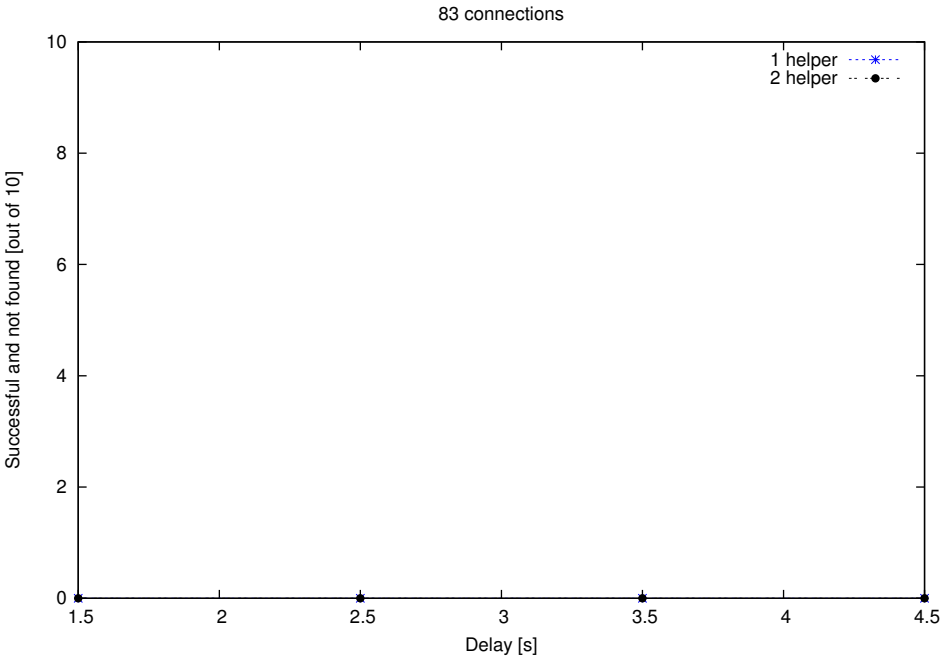


Figure 6.25: Successful and not found attacks on victim with 83 connections

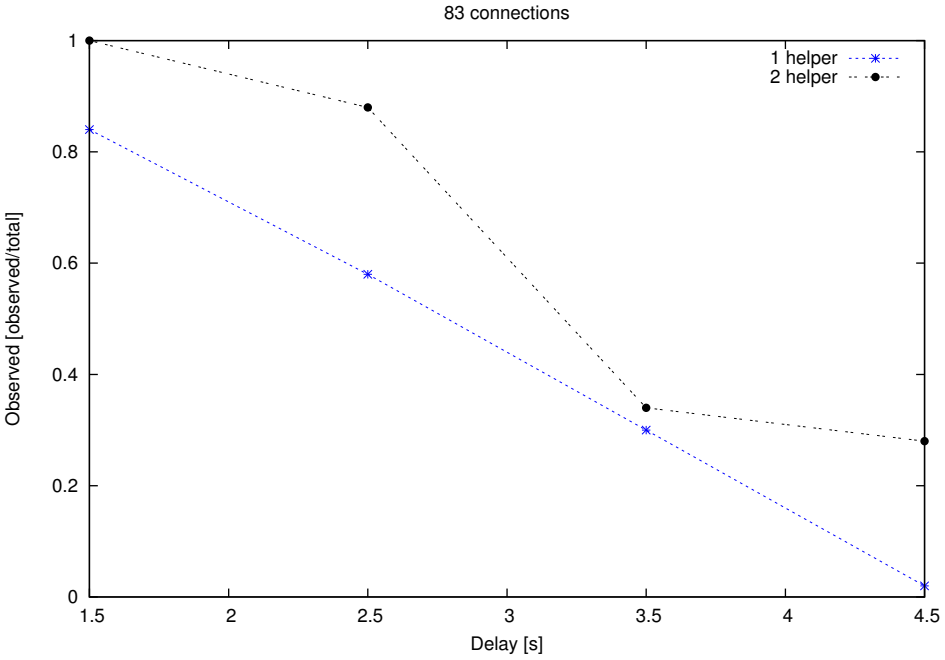


Figure 6.26: Observed attacks on victim with 83 connections

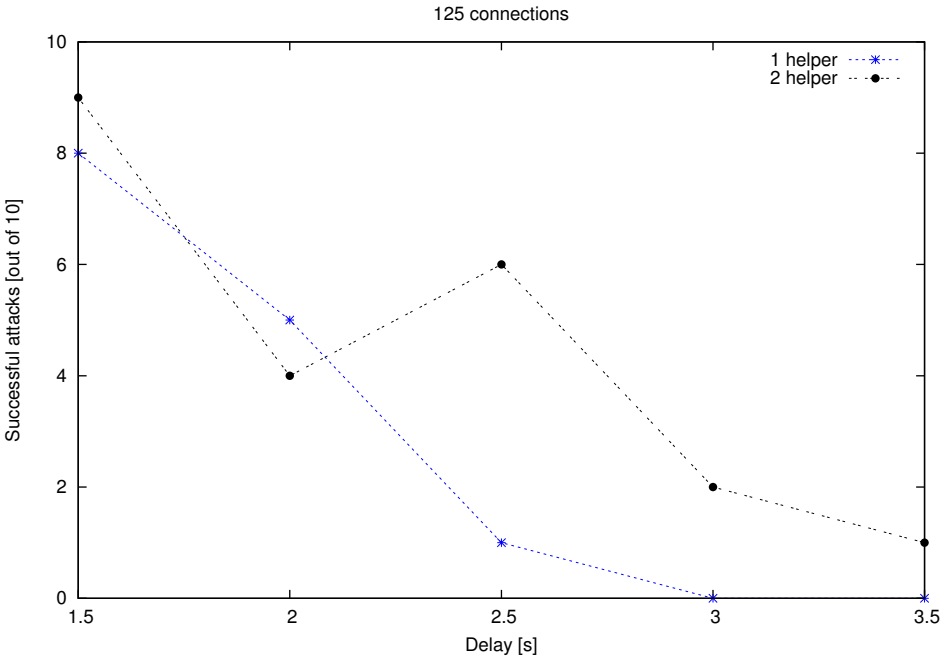


Figure 6.27: Successful attacks on victim with 125 connections

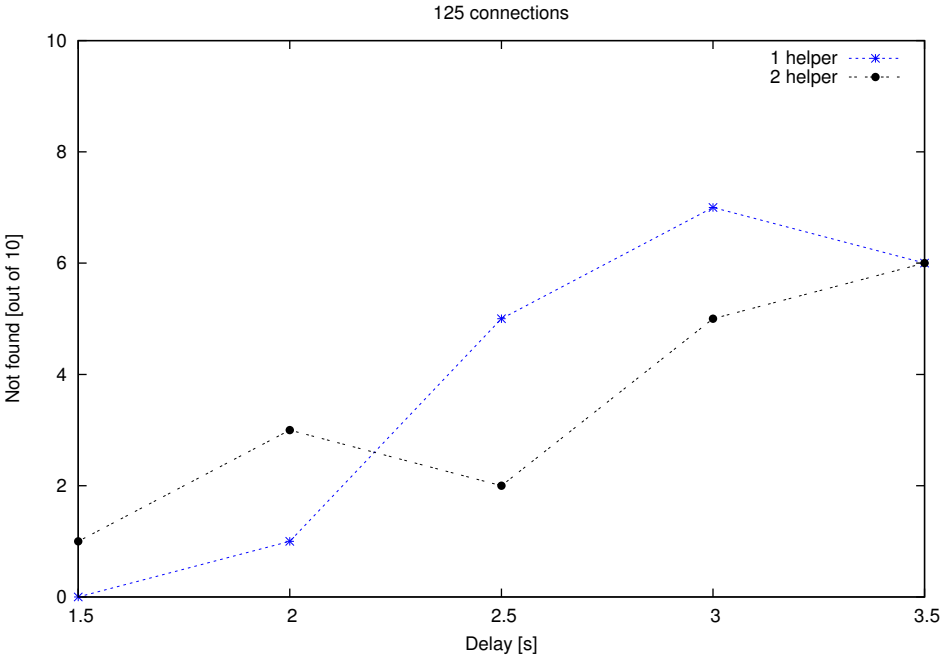


Figure 6.28: Not found on victim with 125 connections

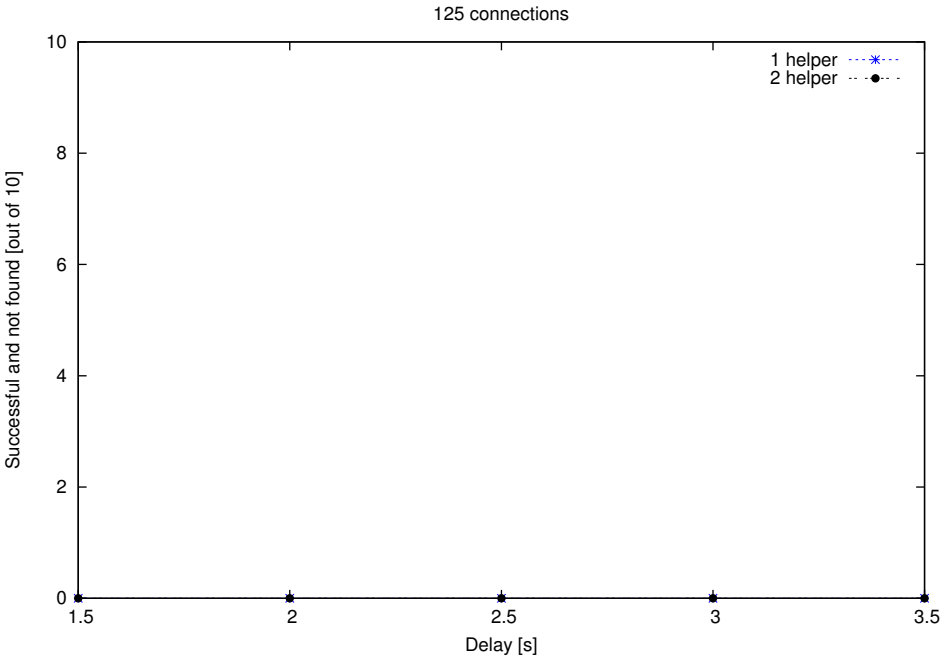


Figure 6.29: Successful and not found attacks on victim with 125 connections

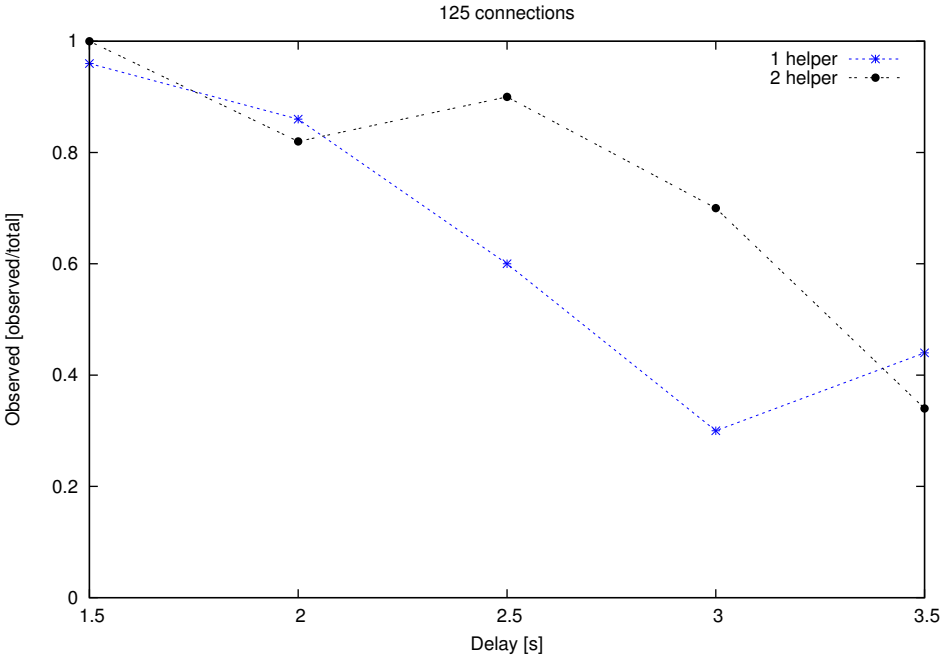


Figure 6.30: Observed attacks on victim with 125 connections

6.2 Analysis

6.2.1 Time measurements of the consensus procedure

The standard deviation is very high. What is more surprising about this data, though, is the maximum and, to a smaller extent, the minimum value. Finding two blocks immediately one after the other and finding no block for such a long time both seem very unlikely.

There are other possible explanations than pure chance for this data. A mechanism inside the consensus procedure adjusts the difficulty to find a new block based upon the past performance of the miners. If the Bitcoin network gains a lot of computation power and thus starts to find blocks fast, the adjustment mechanism will increase the difficulty to find a block.

The following might have happened: Blocks were getting found quickly, by chance or by a lot of computation power, so the difficulty was increased. Then a part of the Network with a lot of computation power stopped, so the total computation power of the Network was reduced by a large amount, and the rest of the network struggled for a long time with the very difficult problem. This is just speculation.

What the data shows us is that the desired mean block finding time of 10 minutes is achieved well. With a standard deviation of about 15 minutes, having longer waiting times for a new block is very probable. This falls in line with our experience during our measurements, as we encountered several waiting times of more than 30 minutes as well as very short ones of less than 10 seconds.

6.2.2 Connection measurements of a Bitcoin node

The limit of 125 connections observed in the Virginia dataset comes from the Bitcoin software as a standard value for the maximum number of connections.

Other than that we can only speculate as to why the node from Virginia gets so many connections so fast, and the node from Switzerland doesn't. The setup process for both nodes is the same, with no obvious differences in network-related parameters (network speed etc.), and some parameters are even in favour of the node in Switzerland (processing power, RAM).

Since the node in Virginia runs on a rented server, there could be a lot of nodes close to it, since other people might have also rented servers there with Bitcoin running on them. Those nodes could prefer to connect to it.

Another effect we observed (but this effect is not in the data, so this is speculation) is that a node which was connected before has a smaller growth rate of connections. It could be that the network setup process recognises IP addresses it has seen before, and gives the respective nodes less priority.

What the data tells is is that a node, even when running for several days, might not reach 100 connections. This has implications with regard to the doublespend-attack which we discuss in section 6.2.3.

The Switzerland scenario can be considered more realistic, since some businesses might shut down their operation overnight, which results in exactly this scenario. Even if the reason for the lower growth rate is different from what we speculate, a node cannot be expected to always have the standard maximal connectivity of 125 peers.

6.2.3 Attack measurements

Successful attacks

As explained in chapter 3, the results of these measurements are probabilistic. Several observations have to be attributed to this fact (especially counterintuitive ones).

We observe a decline in the number of successful attacks when the delay is increased, most prominently displayed in figure 6.4. This meets our expectations, as with a low delay $t_{genuine}$ and t_{rogue} are spread in the network at the same time. With a higher delay $t_{genuine}$ has more time to spread uncontested in the network, so the probability that $t_{genuine}$ gets confirmed is higher, which corresponds to a not successful attack.

A low (or even negative) delay favours t_{rogue}. We think the cause for the fact that a delay of 0 does show a high number of successful attacks, and not 5 out of 10, is the Bitcoin transaction-relay mechanism. The relay-mechanism doesn't immediately relay a transaction, but waits a random amount of time before doing so. This happens twice for $t_{genuine}$, once when the attacker relays it to the victim and once when the victim relays it to its peers, but only once to t_{rogue}, when the helper relays it to its peers. This might be enough to give t_{rogue} a crucial advantage.

What we did not expect, and probably happens because of the randomness involved, is that a higher connectivity of the victim doesn't seem to lead to a lower number of successful attacks. In some measurements, the scenario with a victim with only 8 connections has the highest number of successful attacks, which is the expected result, but in other measurements this is not the case. It is also not monotone, e.g. in figure 6.3 the victim with 40 connections has the highest number of successful attacks. This indicates that the results are this way due to randomness.

With a delay of 2 seconds and one helper node, the chance of an attack being successful is very low (figures 6.3, 6.4). When we use two helper nodes, the number of successful attacks stays much higher, even for a delay of 2 seconds (figures 6.5, 6.6). An additional helper increases the number of nodes which receive t_{rogue} in the first hop significantly so it to spreads in the network much faster, so we expect this result.

Not found

As explained in chapter 3, the results of these measurements are probabilistic. Several observations have to be attributed to this fact (especially counterintuitive ones).

The number of times when not both transactions are received by the victim is named not found.

Not found is higher for victims with a lower connection count (most prominently in figure 6.7). This meets our expectations, since for a victim to not receive t_{rogue}, all of its neighbours must have received $t_{genuine}$ before t_{rogue}. The victim spreads $t_{genuine}$ to its neighbours, and if there are fewer neighbours, the chance that all get $t_{genuine}$ before t_{rogue} is higher.

What seems counterintuitive is the fact that not found falls first, and then raises again (see figure 6.7). We expected not found to raise monotonically with the delay. This might be caused by the following: t_{rogue} arrives at the attacker

(via Bitcoin network) when they still sleep. If the attacker then tries to add $t_{genuine}$ to its local transaction pool, they fail and $t_{genuine}$ is discarded by the attacker, and as a consequence, not relayed. This is not a doublespend-attack anymore, since one of the transactions never entered the network. Note that the attack can also fail this way if not the attacker, but just the victim receives t_{rogue} first. However, in this case, the victim receives both transactions .

With the previous statement in mind, we expect not found to increase with a higher delay. With a higher delay, the victim has more time to spread $t_{genuine}$ to all its neighbours, which lowers the chance of one of them receiving t_{rogue} first, and relaying it to the victim. In some measurements (figures 6.7, 6.8) we almost reach the point where all neighbours receive $t_{genuine}$ first reliably in the 8-connections-scenario.

When we add a helper node, we expect not found to be lower (see figures 6.9, 6.10), since t_{rogue} is spread better in the network and thus more likely to be seen first by one of the neighbours of the victim.

Successful and not found

As explained in chapter 3, the results of these measurements are probabilistic. Several observations have to be attributed to this fact (especially counter-intuitive ones).

Since this data is just a combination of the previous two data sets, many effects are already explained in their respective sections.

As explained before, with a higher delay we observe a lower number of successful attacks. We do, however, expect that t_{rogue} is less likely to be received by the victim. The low number of successful and not found attacks is probably due to the fact that, for an attack to be not found, *each* neighbour of the victim needs to have seen $t_{genuine}$ first, which means that each neighbour spreads $t_{genuine}$ in the network. This scenario likely results in a very wide spread of $t_{genuine}$ in the network, which means t_{rogue} has a low chance of getting confirmed.

While there are some successful and not found attacks, they are very unlikely. In figure 6.14 there are some for the 8-connections measurement, but there are still only 2 out of 10. Even when the victim has only 8 neighbours which have to receive $t_{genuine}$ first, and the attacker has two helpers, both with 125 connections, it is not enough to tip the favour in the network towards t_{rogue} reliably.

Observed

As explained in chapter 3, the results of these measurements are probabilistic. Several observations have to be attributed to this fact (especially counter-intuitive ones).

The observers measure the presence of t_{rogue} in the network.

We expect the measurement to decrease with a higher delay, since $t_{genuine}$ has a larger amount of time to spread in the network uncontested, resulting in a smaller spread of t_{rogue}, and thus in a smaller presence.

We also expect that the number of connections of victim has very little influence on the measurement, since t_{rogue} is spread almost independently of the connection count of the victim.

When we look at the measurements with two helpers (figures 6.17, 6.18), we see that even with a delay of 2 seconds, almost all observers saw t_{rogue}. This falls in line with our previous interpretation that a second helper increases the network presence of t_{rogue} significantly.

What we have to keep in mind is that these measurements only show the presence of t_{rogue} in the network, and not if the observers have accepted t_{rogue} locally. For an observer, a single neighbour that has accepted t_{rogue} locally is sufficient to measure the transaction as present, which is a very small requirement, considering that each observer has 125 neighbours.

Even with that in mind, we see a decline of the network presence of t_{rogue} at a delay of only 2 seconds (more prominent in the 1-helper scenarios, figures 6.15, 6.16).

What we can see is that even with high delays, the chance that any observer sees t_{rogue} is high. With a high probability, at least one of the 5 observers lies on the border of the two network parts (described in chapter 4). It is possible that an observer simply lies inside the part where t_{rogue} is accepted locally, but that is highly unlikely, since that part is probably small, and an observer has 125 neighbours. In either case, an observer sees t_{rogue}

This suggests that a doublespend-detection mechanism based on a few sentinel nodes, like the 5 observers, in the network might be possible. The sentinels would be special-purpose Bitcoin nodes with a very high peer count a node could connect to. The sentinels could then forward all transactions they receive, even ones they would discard as a normal Bitcoin client, to that node. This would very likely make an attack detectable on the node, as with very high probability, at least one observer receives and forwards t_{rogue}.

Something similar is implemented in [14], but this service runs in parallel to Bitcoin, i.e. there is no direct Bitcoin interface. The service does not provide sentinels for other nodes, but uses them for themselves, to update a website according to the observed transactions.

We decide on another approach to detect doublespending.

Exploring measurements

As explained in chapter 3, the results of these measurements are probabilistic. Several observations have to be attributed to this fact (especially counter-intuitive ones).

In these measurements, the goal is not to gather data in a systematic way and gain insight into the influence of parameters on the attack, but to find a parameter setting which produces a desired result. The desired result is a high number of successful and not found attacks. This corresponds to successful attacks which are not detectable on the victim.

The process we use to find the desired setting is to increase the delay until the transactions are not found with a high probability, and to see if any attacks are still successful.

In both the 83 and the 125 connection scenario (figures 6.24, 6.28), the delay needed for a high number of not found is very high, beyond the used values of 4.5 and 3.5 seconds, respectively.

With a delay this high, the attack fails most of the time (figures 6.23, 6.27). As we see in figures 6.26 and 6.30, t_{rogue} is almost not present in the network anymore.

There are 0 successful and not found attacks in both settings (see figures 6.25, 6.29). Even adding another helper node does not change these results much, but for completeness, a measurement for all the delay settings is included.

In the 41 connection setting, the delay needed to have a high number of not found is smaller, and some attacks are still successful with this delay (see figure 6.20, 6.19). Still, there are 0 successful and not found attacks (see figure 6.21).

With the addition of a second helper, more attacks are successful, even with a high delay. In figure 6.22 we see that even with a delay of 3.25 seconds, t_{rogue} is present in the network, but at the same time the number of transactions not found decreases again. When we increase the delay to adjust for this, we cause the number of successful attacks to decrease, and so there are still 0 successful and not found attacks (see figure 6.21).

Our exploration didn't yield a parameter setting with the desired properties, but the parameter space is large and we only explored a small part of it. We don't want to exclude the possibility of such a parameter setting existing.

What we see is that, with sufficiently many connections on the victim, it is very unlikely to find such a setting. We thus decide that the number of connections is a security parameter, which is reflected in our detection mechanism in chapter 7.

Chapter 7

Detection mechanism

In this chapter, we present our detection mechanism. At first we explain it conceptually and then provide insight into the implementation. We conclude this chapter with an evaluation of our new mechanism.

7.1 Design

The detection mechanism we present consists of two parts. One part is the actual detection part which runs just on the local node, and the other part increases detectability in the network by sending additional messages.

As described in chapter 4, our attack works by creating two transactions ($t_{genuine}$,t_{rogue}) which are almost identical, except for the recipient. In particular, they reference the same previous transactions.

Also described in said chapter is the concept of detectability, summarised as the statement that only nodes which receive both transactions can detect our attack. Those nodes are not necessarily the entire network.

7.1.1 Detection

Transactions that are received on a node via the Bitcoin network and are accepted locally, but are not yet confirmed, i.e. they do not appear in a block yet, shall be named open transactions.

When a transaction is received, we add a test to the end of the reception procedure. If the checks (for well-formedness etc.) the Bitcoin client software performs pass, we do nothing. The transaction, even if it were part of an attack, is the first one to arrive, and thus is indistinguishable from a regular transaction.

If one of the checks fails, we compare the previous transactions field of this transaction with the same field of all open transactions. We confirm that we are not comparing a transaction to its duplicate, which can happen when two different nodes send us the same transaction at different times. If we find two different transactions which refer to the same previous transactions, we have found a doublespend-attack. This procedure is sufficient to detect our doublespend-attack on a node, provided that the attack is detectable for this node.

We implement a mechanism that gives positive or negative feedback to the user. When a transaction arrives for a user, i.e. the user is the recipient,

the mechanism starts to monitor this transaction. If no doublespend-attack is detected for this transaction after a certain amount of time, the mechanism will tell the user that this transaction is probably fine. If there is a doublespend-attack detected for this transaction, the mechanism will warn the user. Based on the observations in section 6.2 we monitor a transaction for 10 seconds.

As an additional security feature we also inform the user if their Bitcoin client is connected to less than 100 peers, as we identified the number of connections as a security parameter.

7.1.2 Detectability

When a doublespend-attack is is detectable on a node, it only relays one of the two transactions. We change this behaviour, so the node also relays the second transaction it receives. This relay happens after the relay of the first transaction, so we don't change the reception order on any neighbour nodes.

A node lying on the border between the two network parts (described in section 4.1) behaving as described changes the shape of the border. It will push the border towards (at least) one of its neighbours that was not on the border before, and thus increase the number of nodes for which the attack is detectable. Note that in this scenario, the term border doesn't really fit the situation any longer.

If sufficiently many nodes (note: not necessarily all of them) are modified to behave as described, the attack becomes fully detectable in the network.

See figure 7.1 for a graphic depiction of the effects the detection mechanism has on the network.

7.2 Implementation

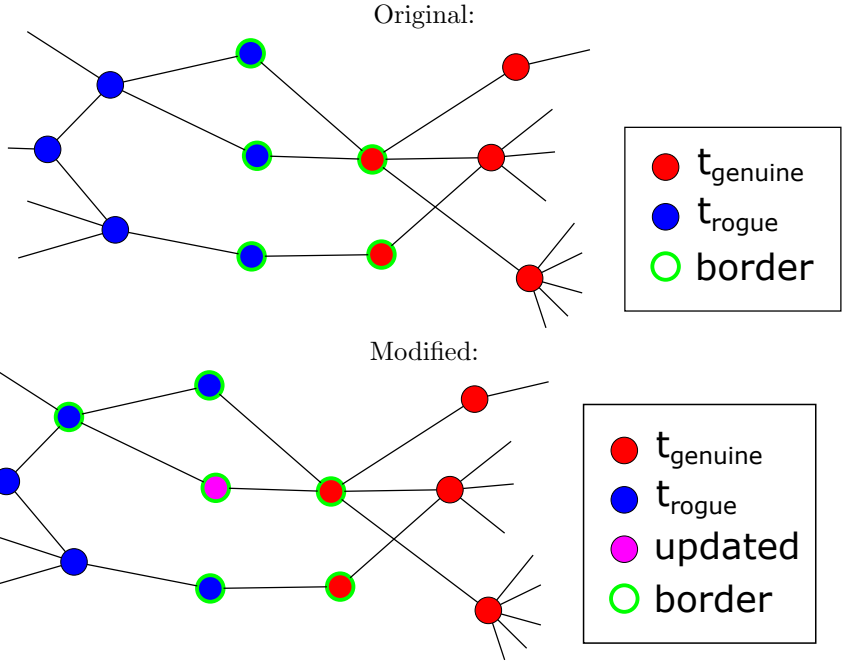
The implementation of the two parts of the detection mechanism are interleaved, so they we don't present them separately.

The Bitcoin client software already stores open transactions. When a new transaction arrives it is checked by the original Bitcoin client. If these checks pass, we inspect if the transaction is for the user, and if so, we monitor the transaction (explained later).

If these checks fail for some reason, we check the transactions against all open transactions and compare their previous transactions field. If they are equal, the hashes of the transactions are compared, and if the hashes are equal, the two colliding transactions are duplicates. In that case, no further action is taken. If the hashes differ, a doublespend-attack is detected.

In this case we put the conflicting transaction into a data structure (a map with the hash of the transaction as key and a boolean indicating the double-spending status as value), or update the data structure if the transaction is already present. A warning is logged. The format of the warning can be seen in figure 7.2. The conflicting transaction is then relayed in the Bitcoin network.

Monitoring a transaction consists of putting the transaction into the mentioned data structure and starting a thread. This thread waits for 10 seconds and then check the status of the transaction in the data structure. Depending on the status of the transaction, positive or negative feedback is output to the console (see figure 7.3).



The red nodes have accepted $t_{genuine}$ locally and the blue nodes have accepted t_{rogue} locally.
The pink node runs our modified Bitcoin client software. It still has t_{rogue} accepted locally.
The nodes with a green circle receive both $t_{genuine}$ and t_{rogue}, while nodes with no green circle only receive one of them. Note that in the modified version the number of green circles increases.

Figure 7.1: Effects of the detection mechanism

Location of the file: $bitcoin/doublespendwarnings.txt. Format of a warning:

```
$timestamp registered doublespending, conflicting transactions:
    $hash_1
    $hash_2
```

Where $timestamp is a Unix timestamp with millisecond resolution and $hash_1 and $hash_2 are the hashes of transaction 1 and 2, respectively.

Figure 7.2: Doublespend-warning

Positive feedback format:

```
transaction $hash seems okay
```

Where $hash is the hash of the transaction.
Negative feedback format:

```
a doublespend-attack was detected for transaction $hash
```

Where $hash is the hash of the transaction.
Connection warning format:

```
    you currently have $number connections
    it is recommended to wait for 100 connections
```

Where $number is the number of peers of the node.

Figure 7.3: User feedback

As an additional security feature, if there is no doublespend-attack detected for a transaction and the number of peers of the node is below 100, a warning is output (see figure 7.3).

The detection mechanism in algorithm notation can be seen in figure 7.4.

7.3 Evaluation

For testing purposes, we run the implementation of the detection mechanism on five nodes in the Bitcoin network for 4 days.

No problems, such as unwanted interaction with regular transactions or crashes of the Bitcoin client software, appeared during this test.

To test the detectability mechanism, we create a small private Bitcoin network by forcing 4 nodes to connect in a certain manner. The resulting network consists of the following 4 nodes: the attacker, helper, victim and messenger node. When the messenger node behaves like the original Bitcoin client software, i.e. dropping either $t_{genuine}$ or t_{rogue} upon reception, the attack is not detectable on the victim and positive feedback is output. With the mechanism we implemented active on the messenger node, the attack becomes detectable on the victim and negative feedback is output.

See figure 7.5 for a graphic depiction of the test network.

There is an additional network load cost of the detectability mechanism, but it is minor. The cost is less or equal than the cost of sending one additional

```
transactionerror ← BitcoinClientTests(transaction)
if transactionerror = true then                          ▷ There was an error
    (doublespend, conflict) ← TestDoublespend(transaction)
    if doublespend ∧ (hash(transaction) ≠ hash(conflict)) then
        doublespendmap[Hash(conflict)] ← true
        Log warning(transaction, conflict)
        Relay(transaction)
    end if
else                                                     ▷ There was no error
    if IsForMe(transaction) then
        doublespendmap[Hash(transaction)] ← false
        Monitor(transaction)
    end if
end if
function TestDoublespend(transaction)
    for all tx ∈ OpenTransactions do
        if tx.previous = transaction.previous then
            return (true, tx)
        end if
    end for
    return (false, NULL)
end function
function Monitor(transaction)                ▷ Is run in a separate thread
    Sleep(10 seconds)
    check ← doublespendmap[Hash(transaction)]
    if ¬check then                           ▷ No doublespend-attack detected
        Output positive feedback
        if connectioncount < 100 then
            Output connection warning
        end if
    else                                     ▷ doublespend-attack detected
        Output negative feedback
    end if
end function
```

Figure 7.4: Detection algorithm

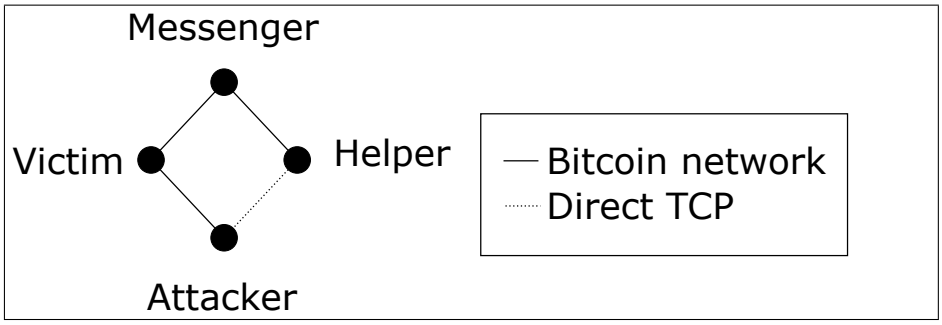


Figure 7.5: Test network for the detectability improvements

Bitcoin transaction in the network, as all upgraded nodes (not necessarily the whole network) relay one additional transaction.

Our tests show that the mechanism works properly, as described.

Chapter 8

Conclusion

8.1 This thesis

We evaluated the doublespend-protection procedure in Bitcoin. While the procedure is not vulnerable, it imposes a long transaction confirmation time of (in expectation) 5 minutes on the Bitcoin system. We mentioned businesses for which this waiting time is infeasible, and we claimed that they are putting themselves at risk of being the victim of a doublespend-attack.

We continued to implement such a doublespend-attack on the Bitcoin system and run several experiments with it. We systematically run one set of measurements to determine the impact attack parameters have on the attack, and we run another set of measurements to achieve certain attack properties by finding the right attack parameters.

While we were able to grasp and explain the effects of the parameters on the attack, we were not successful in discovering a parameter setting which lead to the desired results. We only explored a small part of the parameter space, though, and we think that there might exist a parameter setting which results in the desired properties. Even without achieving all the desired attack properties, we still consider the attack as successful.

We continued to implement a detection mechanism which improves the current doublespend-protection procedure in Bitcoin by warning a user if a doublespend-attack is detected. The mechanism also improves the detectability of the attack in the network at a low network load cost. If our upgrade is deployed wide enough in the network, it will make the attack fully detectable in the network.

Summarising, we have shown a vulnerability for a certain usage type of the Bitcoin system and implemented a mechanism which greatly reduces the risk businesses with this usage type have to take with Bitcoin.

8.2 Future work

There might be a parameter setting which results in the desired, but not achieved attack properties of our attack. Further exploration of the parameter space would be needed to find such a setting. This setting might not exist at all if the detection mechanism we developed is deployed widely in the network.

We implemented a basic version of a doublespend-attack, but attacks can be more sophisticated. We currently doublespend all the Bitcoins of a single transaction, and we detect a doublespend-attack under this assumption. A more sophisticated attack could, for example, doublespend only some of the Bitcoins of a single transaction which our detection mechanism would not recognise as a doublespend-attack. Describing such attacks might lead to more sophisticated detection mechanisms, or to the conclusion that no detection mechanism could ever be sufficiently complex, so the risk of a doublespend-attack cannot be mitigated in the time before transactions are confirmed.

Appendix A

Bitcoin logging mechanisms

In this appendix, we present a logging mechanism we implemented to collect data for the experiments mentioned in chapter 5. We do not explain any details about the design or implementation, but just give an overview.

We implement the logging mechanism in the Bitcoin client software. Some behaviour of the mechanism can be modified, as we will show.

A.1 Console logging

By default, console log output is disabled. It can be enabled with the command `./bitcoind setlog 1` and disabled with the command `./bitcoind setlog 0`. Enabling console logging will cause the Bitcoin client software to output log messages to the console, namely:

- additional output for detected doublespend-attacks, in particular for transactions not concerning the user
- transaction log messages

A.2 Transaction-logging

Transaction-logging logs all transactions involving user-specified addresses. Transaction-logging uses a configuration file that is loaded at program start. The format of the configuration file can be seen in figure A.1.

Which events are logged is fixed in the software, but templates for other types of events exist in the code as well. See the source file $bitcoin/src/main.cpp and the function **void** writelog(**int** type, CTransaction tx) for details. The following events are logged:

- reception of a transaction via Bitcoin network
- relay of a transaction via a direct TCP connection
- relay of a transaction via Bitcoin network

Note that "relay of a transaction via Bitcoin network" also includes detected doublespend-transactions. The format of the log can be seen in figure A.2.

Location of the file: $bitcoin/addresses.txt. Format of addresses.txt:

```
file = (line'\n')*line | ''
line = '$address'
```

Where each line represents one address. $address is the Bitcoin address to be logged.

Figure A.1: Addressfile

Location of the file: $bitcoin/log.txt. Format of a log entry:

```
$timestamp $message $hash (address: $address)
```

Where $timestamp is a Unix timestamp with millisecond resolution, $message is text depending on the type of event logged, $hash is the hash of the transaction and $address is the Bitcoin address involved.

Figure A.2: Transaction log

A.3 Connections-logging

We log the number of connections a node has. This information is never output to the console, but is written to a file. The format of the connectionslog can be seen in figure A.3.

A.4 Doublespend-warnings

Doublespend-warnings are logged, as has been already covered in chapter 7. The format of the warnings can be seen in figure 7.2.

Location of the file: $bitcoin/connectionslog.txt. Format of connectionslog.txt:

```
$timestamp $incoming incoming connections ($total total)
```

Where $timestamp is a Unix timestamp with millisecond resolution, $incoming is the number of incoming connections and $total is the total number of connections.

Figure A.3: Connectionslog

Appendix B

Scripts

In this appendix, we present several scripts we create to facilitate different aspects of the work related to this thesis.

We just present the scripts briefly.For a more thorough description, please consult the respective README file.

B.1 Analysis of the blockchain

We write a script that accesses data about the Bitcoin blockchain via `https://blockexplorer.com/`. It collects the timestamps when the blocks were created and evaluates the block finding times (the time from a timestamp to the next) statistically. It can be found in ./scripts/blockanalysis/.

B.2 Connectionslog post processing

We write a script that processes the connectionslog and generates a file which can be plotted directly. It can be found in ./scripts/plot/.

B.3 Collection of log files

We write a script that collects various log files (connectionslog, transaction log) from multiple sources which have been set up according to our experiment setup and stores them in a way such that the script presented in appendix B.4 can evaluate them. It can be found in ./scripts/downloadlogs/.

B.4 Evaluation of log files

We write two script that evaluates logs stored in a certain format (described in section 5.2) and creates a PDF report. The one script, for $delay \geq 0$, can be found in ./scripts/evaluatelog/ and the other script, for $delay < 0$, can be found in ./scripts/negevaluatelog/.

Bibliography

[1] Blockexplorer. https://blockexplorer.com/b/123456. Block 123456, looked at on 2012-03-13.

[2] Xavier Boyen and Elaine Shi. Bitter to better - how to make bitcoin a better currency. In *Financial cryptography and data security 2012*, February 2012.

[3] Bitcoin charts. http://bitcoincharts.com/bitcoin/. A bitcoin economic status analysis, looked at on 2012-03-08.

[4] Bitcoin charts. http://bitcoincharts.com/markets/currencies/. A bitcoin currency value analysis, looked at on 2012-03-08.

[5] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *Advances in Cryptology CRYPTO 88*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer Berlin / Heidelberg, 1990.

[6] CNN. Bitcoins uncertain future as currency. http://money.cnn.com/video/technology/2011/07/18/t_bitcoin_currency.cnnmoney/, July 2011. A newscast.

[7] Frerk-Malte Feller. Paypal globales zahlungssystem mit kompetenz fr lokale zahlungsmrkte. In Thomas Lammer, editor, *Handbuch E-Money, E-Payment & M-Payment*, pages 237–247. Physica-Verlag HD, 2006.

[8] Hal Finney. https://bitcointalk.org/index.php?topic=3441.msg48384#msg48384. Finney attack, looked at on 2012-04-18.

[9] jevonx. https://github.com/bitcoin/bitcoin/issues/1034. Double-spend alerts for fast transactions and added security, looked at on 2012-04-19.

[10] Mt.Gox K.K. https://mtgox.com/. A bitcoin exchange, looked at on 2012-03-08.

[11] Satoshi Nakamoto. https://bitcointalk.org/index.php?topic=423.msg3819#msg3819. Reply to "Bitcoin snack machine (fast transaction problem)", looked at on 2012-04-18.

[12] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://www.bitcoin.org/bitcoin.pdf, October 2008.

[13] Peio Popov. Electronic money: The road to bitcoin and a glimpse forward. http://mirror.fem-net.de/CCC/28C3/mp4-h264-HQ/28c3-4668-en-electronic_money_h264.mp4, December 2011. A talk.

[14] Transaction radar. http://transactionradar.com/. A Bitcoin network observing service, looked at on 2012-04-19.

[15] F. Reid and M. Harrigan. An analysis of anonymity in the bitcoin system. In *Privacy, security, risk and trust (passat), 2011 ieee third international conference on and 2011 ieee third international conference on social computing (socialcom)*, pages 1318–1326, October 2011.

[16] RowIT. http://bitcoinstatus.rowit.co.uk/. A bitcoin network status analysis, looked at on 2012-03-08.

[17] O. Goldreich S. Even and Y. Yacobi. Electronic wallet. In Kevin McCurley and Claus Ziegler, editors, *Advances in Cryptology 1981 - 1997*, volume 1440 of *Lecture Notes in Computer Science*, pages 383–386. Springer Berlin / Heidelberg, 1999.

[18] Bitcoin wiki. https://en.bitcoin.it/wiki/Confirmation. Confirmation, looked at on 2012-04-18.

[19] Bitcoin wiki. https://en.bitcoin.it/wiki/Myths#Point_of_sale_with_bitcoins_isn.27t_possible_because_of_the_10_minute_wait_for_confirmation. Myths, looked at on 2012-04-18.

[20] Bitcoin wiki. https://en.bitcoin.it/wiki/Green_address. Green address, looked at on 2012-04-18.

[21] Bitcoin wiki. https://en.bitcoin.it/wiki/Wallet. Wallet, looked at on 2012-03-12.

[22] Bitcoin wiki. https://en.bitcoin.it/wiki/Block_hashing_algorithm. Block hashing algorithm, looked at on 2012-03-12.

[23] Bitcoin wiki. https://en.bitcoin.it/wiki/Transaction. Transaction, looked at on 2012-03-13.