Safe acceptance of zero-confirmation transactions in Bitcoin

Renlord Yang

Computing & Information Systems

University of Melbourne

Supervisors:
Dr. Christian Decker
Dr. Udaya Parampalli, University of Melbourne

A thesis submitted for the subject of

COMP60003 - Computer Science Research Project

03 June 2016

Acknowledgements

First and foremost, I would like to thank Dr. Christian Decker for the weekly meetings we have had during my exchange semester in Zurich. The meetings we have had were crucial to the proposed insurance scheme and my understanding of Bitcoin. Secondly, I would like to thank Dr. Udaya Parampalli for his guidance as a mentor and supervisor. Your advice on both life and academia were essential to my experience as a Masters student in the University of Melbourne.

Finally, I would like to extend my gratitude to my friends and family for taking out time for critiqueing the ideas I have presented in my thesis and for the moral support you have kindly shown me throughout the writing of my thesis.

Declaration

I certify that

- this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material reviously published or written by another person where due reference is not made in the text.
- where necessary I have received clearance for this research from the Ethics Committee of the University and have submitted all required data to the Department.
- the thesis is 14130 words in length (excluding text in images, table, bibliographies and appendices).

Abstract

Acceptance of zero confirmation transactions in Bitcoin is inherently unsafe due to the lack of consistency in states between nodes in the network. As a consequence of this, Bitcoin users must endure a mean wait time of 10 minutes to accept confirmed transactions. Even so, due to the possibility of forks in the Blockchain, users who may want to avoid invalidation risks completely may have to wait up to 6 confirmations, which in turn results in a 60 minute mean wait time. This is untenable and remains a deterrent to the utility of Bitcoin as a payment method for merchants.

Our work seeks to address this problem by introducing a novel insurance scheme to guarantee a deterministic outcome for transaction recipients. The proposed insurance scheme utilizes standard Bitcoin scripts and transactions to produce inter-dependent transactions which will be triggered or invalidated based on the occurance of potential doublespend attacks. A library to setup the insurance scheme and a test suite was implemented for anyone who may be interested in using this scheme to setup a fully anonymous and trustless insurance scheme. Based on our test in Testnet, our insurance scheme was successful at defending against 10 out of 10 doublespend attacks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Bitcoin is the world's first peer to peer digital currency with a market capitalisation of 7B USD.[1] Its distributed system supports and facillitates up to 270,000 transactions per day or 3 transactions per second. Bitcoin transactions are verified and enforced in a distributed manner with no central authority to mediate disputes. As a consequence, each peer must download and synchronise its own copy of the Bitcoin distributed ledger, commonly known as the Blockchain. Given the distributed nature of Bitcoin, it is vulnerable to third party malicious attacks and more often than not, users of the Bitcoin network who are not educated of the technical intricacies of Bitcoin are often victims transactional exploits such as doublespend attacks. Currently, there are no known methods to mitigate such attacks, other than to wait for transaction confirmations, which is time consuming as the expected mean wait time is 10 minutes for a single transaction confirmation, but may take considerably longer. Thus, Bitcoin as it is in its current state is not a feasible payment system for minimal-contact brick and mortar businesses such as supermarkets, clinics, newsagents and restaurants as it is not reasonable to expect a customer to wait around a checkout counter until a Bitcoin transaction is confirmed.

[1] as of 9th May 2016 from http://blockchain.info

1.1 Motivation

Bitcoin is a distributed network which follows an eventual consistency model for the state of transactions. Therefore, there is non-determinism when it comes to when a transaction actually gets confirmed. In the current state of Bitcoin, it is possible for transactions to either get stuck or fail to get committed to the Blockchain for a wide variety of reasons. Broadcasted transactions may fail to be relayed by peers due to insufficient transaction fees. Reasons include overwhelming traffing, thus dynamically increasing the fee requirement for confirmation or even accidental failures to relay said transcation. Alternatively, Bitcoin is also vulnerable to *doublespend attacks* where a malicious user attempts to execute two transaction with one transaction spent to an anticipating receiver, while another transaction which spends the same coins are spent back to himself. Thus, leading to the failure of verification of one of transaction by peers in the Bitcoin network. These inconsistencies more often than not are difficult to observe in the Bitcoin network as conventional peers are programmed to only connect to 8 outbound peers. As a consequence, it takes time for a node to be able to synchronize its state with the state of the Bitcoin network and it is possible for a node to be fed malicious information which enables the attack to be successful.

Uptake in Bitcoin is also significantly impaired as it is difficult for merchants to confirm Bitcoin transactions quickly. While the Bitcoin protocol has set the mean block generation time to approximately 10 minutes, the block generation time suffers from a high variance, thus it is often difficult to estimate how long it will take for a transaction to be included in a block. Depending on transaction volume in the Bitcoin network, it is also possible that should there be a surge in transaction volumes, transactions may not be included in the next available minted block as each block in the Blockchain are limited in size. Such shortcomings are one of the core reasons why Bitcoin has yet been adopted as the payment of choice for brick and mortar merchants as it would be infeasible to expect a customer to wait at checkout until a transaction has been confirmed.

Due to recent consensus rule changes, specifically Opt-In Replace By Fee [7], there are concerns within the Bitcoin community that doublespending will be easier as it is now possible to replace transactions. As the selection process for transactions to be included in blocks cannot be enforced by consensus rule, it is possible for profit-maximizing clients to ignore recommended transaction selection policy and accept doublespend transaction as long as they maximise a miner's profit. Therefore, despite efforts by developers to reduce doublespend capabilities in client distributions, it is still possible for advanced users who are aware of the consensus rules of Bitcoin to augment their clients to accept or relay doublespend transactions.

1.2 Goals

The purposes of this research is to reduce non-determinism in the commitment of Bitcoin transaction in the Blockchain by utilising a robust zero-confirmation transaction insurance scheme and a detection mechanism for participating insurers to react to an unfavourable outcome. Robustness may be achieved by guaranteeing the succesful receipt of insured sum by the designated recipient of the insured transaction, at the same time allowing timely intervention for an insurer to attempt to mitigate the need to reimburse an insured recipient by performing counter-actions within the Bitcoin network to encourage the confirmation of the insured transaction as soon as possible. At the same time, another main objective is to also figure out the proportion of Bitcoin peers in the network that are accepting a new type of transaction which allows replacement by incrementing its transaction fee.

1.3 Contributions

The contribution of this thesis includes:

- Proposal of a novel zero confirmation Bitcoin transaction insurance which guarantees a deterministic outcome regardless of different state outcomes.

- Implementation of an insurance scheme and system which facilitates the negotiation, setup and execution. As part of our commitment to the requirements of the thesis, an evaluation and analysis of the system is also presented as part of the thesis.
- Comparison of different contract transaction methods to execute insurance scheme and its associated monetary cost.
- A survey on the proportion of Bitcoin peers accepting Replace-By-Fee (RBF) transactions and a measure on the rate of propogation of a RBF transaction compared to a standard transaction.
- An implementation of a Bitcoin monitoring software which enables arbitrary transactions to be tracked. Detailed information such as proportion of nodes reached within the network will be recorded and the propogation speed will be measured.

1.4 Structure

In chapter 1, we will provide a thorough introduction of Bitcoin, providing the necessary details for the reader to understand the underlying contructs of the proposed insurance scheme. In chapter 2, we present our findings and discussion on the acceptance of RBF-enabled transaction in both the Mainnet and Testnet Bitcoin networks. In chapter 3, we present our insurance contract designs, followed by a detailed description of the implementation and simulation of these contracts in chapter 4. Finally, we end our thesis with a conclusion.

1.5 Related Works

Bitcoin [14] was intially proposed and implemented based on a proof of work scheme similar to Hashcash [3] and the idea of an untraceable digital currency [6]. Following widespread adoption and substantial review, Bitcoin improvement proposals were introduced to improve the usability and extend the functionality of Bitcoin [1][7]. Segregated witnesses [12] and normalized TXIDs [8] were proposed to eliminate the potential of transaction mallaebility altogether by removing signature data from transactions. The effects of information propogation delay on the scability of Bitcoin was examined by Decker et. al. and numerous suggestions were made to reduce the delay time [9]. Due to the inherent capacity limitations of Bitcoin, off-chain scalability such as sidechains [2], lightning network [16] and duplex micropayment channels [11] were also introduced to increase the number of transactions processable by the network at any given point in time. On-chain scalability which involves augmenting the Bitcoin mining protocol which enables the inclusion of orphaned blocks with an "uncle" relationship was also proposed [18].

Numerous studies contributed to the wide variety of attack vectors and model against the Bitcoin network, specifically transactions Rosenfield's work on Hashrate based double-spending proposed a probability model for the likelihood of success of double spend attacks [17]. Bamert et. al. studied the viability of accepting fast payments [4] (i.e. zero confirmation transactions) and presented numerous suggestions on the methods to safely determine if a zero confirmation transaction was safe to accept. Lei et. al. [13] presented a doublespend attack vector which exploits temporal inconsistencies in transactional states between peers. Transaction malleability [10] as a means to perform doublespend attack and its effects on big businesses were also explored by Decker et. al.

Chapter 2

Bitcoin

Bitcoin [14] is a peer to peer distributed payment system on a global scale without a centralised intermediary. Coins may be minted and payments can be made between two peers by means of broadcasting a transaction to the peer to peer network. All of these are made possible through the enforcement of consensus rules within the distributed network. To participate in the Bitcoin network, each peer are free to deviate from the reference client, but its implementation of custom policies must adhere to the Bitcoin protocol. Implementation of policies with regards to transaction verification, block verification, creation of blocks must adhere to protocol followed by the network. Failure to conform to the Bitcoin protocol may lead to inconsistent states between peers, rejection of malformed transactions and rejection of block announcements.

In this chapter, the technical details of Bitcoin will be presented to the reader as a guide for the subsequent parts of the thesis. Firstly, we will talk about how Bitcoin maintains its state. Secondly, we talk about how state changes (ie. transactions) are formularised, broadcasted and verified by the Bitcoin network. Finally, we present some critical consensus rules that are crucial to the work of thesis.

2.1 Blockchain

The state of the currency used by the Bitcoin network, also known as Bitcoin (BTC) is maintained with the use of a distributed data structure known as the *Blockchain*. The Blockchain follows the structure of a directed tree and has no central authority to govern read and write permissions to the data structure. It is designed to be replicated by every other peer in the network and maintained distributed by all peers. All future writes to the data structure are independent verified based on a set of consensus rules agreed by the consensus network. Therefore, the Blockchain data structure is Byzantine fault tolerant. The root node of the Blockchain is known as the *Genesis Block*. It is the first block created to initialise the Blockchain. A path to the Genesis Block can be found for all subsequent blocks that are appended to the Blockchain. Each block in the Blockchain contains an intial transaction known as the *Generation Transaction*. This transaction is created to reward the miner that has discovered the proof of work for the founded block. An exception is applied to the generation transaction in the genesis block where the outputs of the transaction cannot be spent. The amount of reward paid by the Generation Transaction is enforced by Bitcoin consensus rules, therefore should a miner attempt to produce a Coinbase transaction which pays more than the reward allowable by consensus, the newly discovered block will be rejected by the Bitcoin network.

Block Suppose we denote B_0 as the *Genesis Block*. We can formally denote all descendent blocks of the Genesis Block as $B_i, i > 0$. Each block in Bitcoin contains a blockheader which stores the value of the merkle root hash of all transactions included in the block, the hash of the previous block header, a nonce and etc.. We may formally express the dependency between each block as $\texttt{Hash(Header of } B_{i-1}\texttt{)} = B_i\texttt{.BlockHeader.prevBlockHash}$. Figure 2.1 illustrates this dependency requirement enforcable by consenses rules. During the process of mining, transactions are accepted and propogated continuously throughout the network. As they cannot be immediately committed to the Blockchain, they are stored in a data structure stored in non-persistent memory, known as the *mempool* until they are included in a block by a miner.

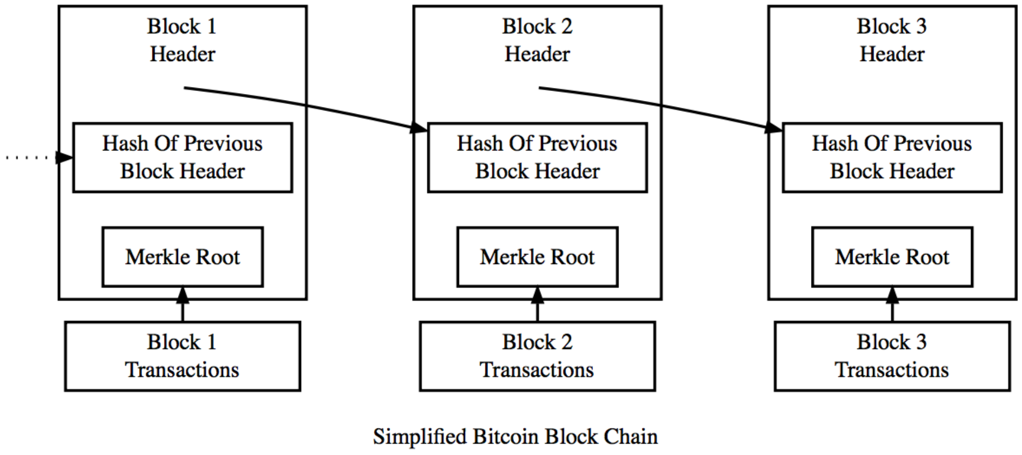

Figure 2.1: Blockchain Structure

2.1.1 Mining

The *proof of work* scheme used in Bitcoin is modelled after the Hashcash proof of work scheme, where the proof of work is a function which consumes a tuple of a service string, a nonce and a counter as an input and produces a hash of the input as an output. Similarly, in Bitcoin the service string is represented by the information provided in the latest accepted block in the Blockchain. As the 4 byte nonce included in a header may be exhausted very quickly by a miner with a hashing capacity of more than 4GH/s, an extra nonce field which is included in the *Coinbase* to increase the total number of hashes made possible with a set of blockheader information. The *Coinbase* is a special field which replaces the signature script in a Generation Transaction. Miners use this transaction to claim all *mining fees*, which by definition is the net difference between the sum of output values and the sum of input values and a block reward which decreases exponentially over time. As each Coinbase is unique between miners, we may assume that each merkle root hash of a new block will be different. Therefore, for every new transaction that is received, this merkle root hash is ever changing. As a result of this, the block minting process features a *memoryless* property and is considered a Poisson process.

Discovering a solution for a proof of work to create a block is known as *mining* and the process is inherently difficult as the blockchain is designed to be an immutable data structure, therefore write privilleges to the blockchain is controlled through the means of solving for a proof of work to gain write access. The proof of work is computed by hashing the merkle root hash, the hash of the previous block and a nonce until a hash with a certain number of leading zeroes has been found. The requirement of the number of leading zeroes is determined by consensus and is formally known as the *network difficulty.* The network difficulty is re-set each time 2016 blocks are minted. The current targeted minting duration for 2016 blocks is set to 14 days, therefore should the network be able to mine 2016 blocks in less than 14 days, the network difficulty will increase by the network. Otherwise, the network difficulty will be lowered instead. Participation in mining is voluntary for users of Bitcoin, however it is expected of users to maintain their own copy of the blockchain to execute their own verification in order to enforce consensus rules.

2.1.2 Forks

Occasionally, the Bitcoin may experience a network fork. A *network fork* is essentially the splitting of the Bitcoin network as a whole into the total number of *forks* that has occured in the blockchain. As minting may be modelled as a Poisson process, it is possible for two or more blocks to be discovered concurrently. As both blocks get propogated to the network, nodes within the network will either accept one block or the other depending on how effective either blocks are propogated. The Blockchain only contains blocks that are successful producing the longest chain of blocks. Blocks that are unsuccessful in producing a descendent to maintain the status of being the longest chain become *orphan blocks.* Transactions included in *orphan blocks* are thus invalidated as they are no longer considered the sequence of events which has occured as recorded by the Blockchain. As transactions in Bitcoin are mutually dependent, more often than not, the invalidation of one transaction due to a Fork may need to a cascading invalidation of subsequent future transactions. Figure 2.2 illustrations the occurance of a fork in the Blockchain and how forks are eventually resolved.

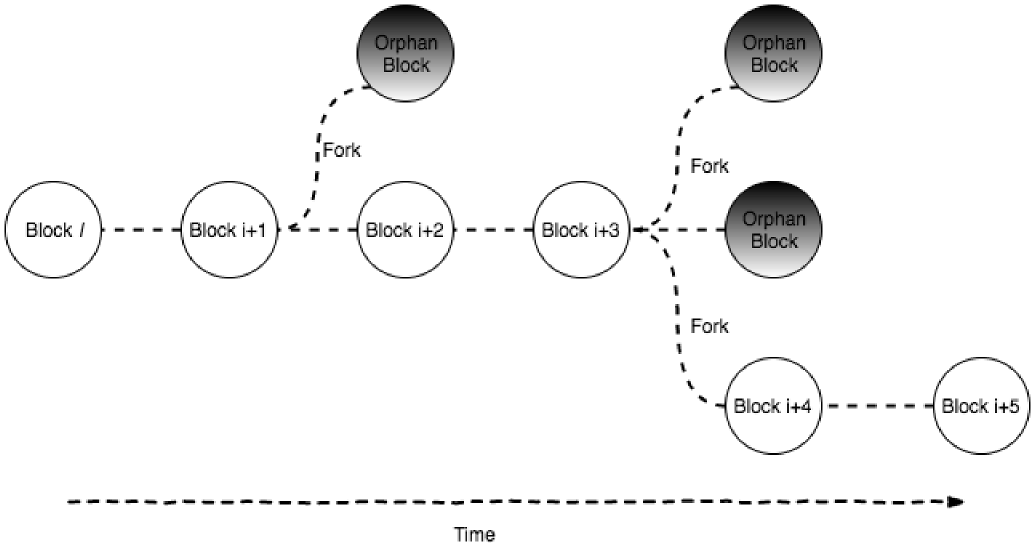

Figure 2.2: Blockchain Fork

2.2 Transaction

While the maintenance of the state of Bitcoin is executed through the use of Blockchain, where each block in the Blockchain represents some state, S_i of the state of ownership of coins. A *transaction* is used to denote a state transition from S_i to S_{i+1} within Bitcoin, when executed by the Bitcoin network, applies a state change. A transaction in a literal sense consumes a list of unspent outputs from previous transactions and produces a new list of outputs. The execution environment of Bitcoin may be interpreted as a higher order function which consumes a state transiton function $g(x)$, where $g(x) \in Transactions$. A transaction can then be represented as a function $f(x)$ which takes an input $x \in UTXO_{S_i}$ and outputs $y \in UTXO_{S_{i+1}}$. *Unspent transaction outputs* (UTXOs) are outputs from transactions that have not been used in transactions as inputs. The details of the function itself is then defined by the execution steps expressed in the script included in the transaction. Received transactions by peers in the network are independently verified by consensus rules; whereas the selection policy to choose the set of transactions to be included into a newly found block cannot be enforced by consensus rules. We now define the components necessary to produce a valid transaction for Bitcoin.

Output An *output* in a transaction represents an ownership change for the coins funded with the corresponding list of inputs within the same transaction. An output consists of public key script (*pkScript*) and the value of coins transfer to the designated recipient of the *pkScript*. An unspent output (UTXO) from a transaction may only be used once as an input to another transaction as once an unspent output is consumed by a transaction, its state is no longer maintained by the Blockchain.

Input An *input* is a reference to an unspent output from another Bitcoin transaction. Each input consists of a signature script (scriptSig), a double hash of a previous transaction, an index that points to the n-th output in the previous transaction and a sequence number. ScriptSigs contain the claiming condition of an output of a transaction and the signatures required to validate the claiming conditions as speicified in the script component of the scriptSig. The SHA256 hash of a previous transaction is also used as a transaction identifier and it is used in outputs when referencing a previous transaction. The previous transaction that is referenced may be a confirmed transaction or it may be an unconfirmed (zero-confirmation) transaction. A *zero-confirmation* transaction is a transaction which has not been included in the Blockchain, but has been accepted to the Mempool of the peers in the Bitcoin network.

Timelock A *timelock-ed transaction* in Bitcoin includes a time lock which prevents a transaction to be committed into a block until the specified lock time has elapsed. When a transaction is broadcasted prior to the lock time, said transaction will be rejected by the network. Such a policy was introduced to eliminate spam transactions that may exploit the use of timelocks in transactions to overwhelm the Mempool of peers within the Bitcoin network.

Fees The difference between the sum of values of outputs and the sum of UTXOs contributed by inputs is the *fee* to be paid to miners. A negative difference would result in a malformed transaction as there would be insufficient funds to execute the transaction and a net difference less than the *dust fee* would also result in the rejection of a transaction. In Bitcoin, the *dust fee* represents the minimum transaction fee for each kilobytes of data used by a transaction. The intrinsic value of a transaction fee is computed relative to the complexity of the transaction itself. Transaction fees are evaluated on the basis of the combined value of a transaction chain. As an unconfirmed transaction tx_i may reference another unconfirmed transaction tx_{i-1} which is broadcasted earlier to the Bitcoin network, the transaction fee is computed on a per transaction chain basis, as oppose to a per transaction basis, so as long as every transaction along the transaction chain pays the minimum fee required to be accepted as it is a base requirement to pay the minimum fee to qualify for acceptance.

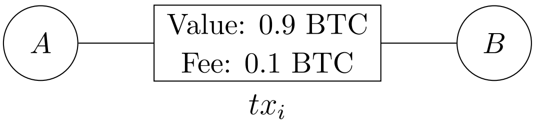

Figure 2.3: Standard Transaction of user X paying user Y 1 BTC

Standard Transaction To reduce the incidence of malformed transactions which inevitably consume network bandwidth, the network enforces the acceptance of standard transactions and rejects non-standard transactions. The *standard transaction* consensus rule requires that each broadcasted transaction must contain one of two types of Public Key Scripts in its output. The two types of Public Key Scripts are known as *Pay-to-Pubkey-Hash (P2PKH)* and *Pay-to-Script-Hash (P2SH)* [1]. P2PKH scripts contain the hash of the public key of the designated recipient of the output. P2SH Public Key Scripts allow a receiver to provide a redeem script of its preference to a payer of a transaction. The P2SH is calculated by hashing the redeem script produced by the receiver and is provided to the payer for payment. P2SH allows a receiver to conveniently redeem the outputs of the transaction by simply reproducing the same redeem script that was used to produce the P2SH.

2.2.1 Child Pays For Parent (CPFP)

Child-pays-for-parent is a method at which a dependent transaction is used to pay the transactions fees for both itself and a parent transaction it references.

CPFP is often used when the liability of transaction fee is shifted from the payer to the receiver, thus the receiver may consume the output contributed by the parent transaction to fund the receiving transaction and the transaction which pays the transaction fee. Consider the situation where user A wishes to pay user B 1 BTC, but is unwilling to pay any transaction fee. A can create a transaction tx_i which pays exactly 1 BTC to user B where net difference in value between the outputs and inputs is 0. For tx_i to be accepted by the Bitcoin network, tx_i must be funded. Thus, user B may use CPFP to increase the value of payable transaction fees to miners by consuming the output of tx_i to create transaction tx_{i+1} which then pays user B the net difference of the transaction fees and the receiving value.

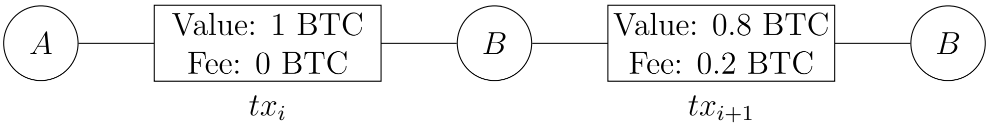

Figure 2.4: Child Pays for Parent Transaction

2.2.2 Replace By Fee (RBF)

Replace by fee (RBF)[15] is a transaction acceptance policy which allows transactions that have already been accepted to be replaced by transactions that pay higher fees regardless of its changes in inputs or outputs. A RBF transaction in Bitcoin is created by setting the `nsequence` of the inputs of the transaction to less than `0xffffffff`. For each subsequent replacement, the payer may increment the `nsequence` and pay a higher fee to replace a previous broadcasted transaction. Figure 2.5 illustrates a simple RBF transaction. The input referenced by both tx_0 and tx_{RBF} refers to the same output from a previous transaction as shown in the diagram.

2.2.3 Transaction Signing

Bitcoin transactions allow a spender to control the sections of transactions to be signed. Signed components of a transaction are immutable, as any mutation

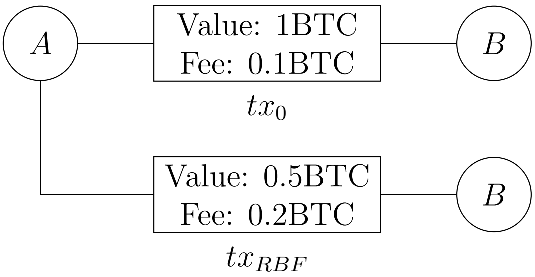

Figure 2.5: Replace by Fee Transaction

attempt will result in the change of the transaction hash. The different level of control over components to be signed is determined by the `SIGHASH` type provided to the signing function. A signed transaction includes its signatures in its scriptSig and a corresponding SHA256 Hash is produced and may be used as a reference for future transactions. However, a signature cannot sign the contents of any scriptSig from any inputs as it is not possible for a signature to sign itself.

Signature Hash Types

- `SIGHASH_NONE` when used will allow the spender to sign none of the outputs, thus allowing Public Key Script and value of each output to be mutable. However, all inputs are signed.
- `SIGHASH_SINGLE` when used allows the spender to sign all inputs and a single output where the index of the output to be signed must be equal to the index of the input contributed by the spender.
- `SIGHASH_ALL` when used allows the spender to sign all inputs and outputs, putting the transaction in an immutable state. This is the default `SIGHASH` operation used for most standard transactions.
- `SIGHASH_ANYONE_CAN_PAY` when concatenated with any of the `SIGHASH` types above allows the spender to only sign the input it contributes to a transaction instead of including all inputs into the signing of the signature, allowing additional inputs to be included in a transaction.

2.3 Script

Bitcoin transactions typically require the spender of a transaction to fulfill a set of conditions which is encoded in a sequence of bytes within a transaction described in a language known as Script. The scripting language, *Script* is a purposefully non-Turing complete, stack-based language that uses single byte opcodes. Typically, when a spender pays a recipient some coins, the spender will require a recipient to produce a redeem script which defines the set of conditions the spender must fulfill in order to spend in future transactions. The redeem script is then hashed to a *Pay-to-Script-Hash (P2SH)* to be included in the output of the transaction. Should the recipient of the transaction elect to spend the output received in said transaction, she would have to re-produce the script originally used to produce the P2SH and provide the corresponding signatures required. The signature and the script is then included in the input of the spending transaction and sent to the Bitcoin Network for validation. Should validation be successful, the spending transaction will be successful, otherwise, it will not be possible for the recipient to spend the received coins.

Complex Redemption Conditions Script also allows users to set up complex claiming conditions such as writing a transaction which pays to a shared account held by multiple signatories. We call these transactions a *M of N Multisig transaction*, where M is the number of signatories required to sign a transaction and N is the total number of registered signatories. For example, if we had a transaction that has paid 2 BTC to a 2 of 4 Multisig transaction. In future, when spending the output, only 2 of the 4 signatories will be required to sign the transaction that will spend the output of the referenced transaction. Multisig transactions are typically useful in cases where trustless and fair protocols are implemented. They are used in micropayment channels, duplex payment channels, escrow-based transactions and etc.

2.4 Transaction Finality

Bitcoin transactions do not get transaction finality as it is possible for a transaction to be invalidated due the lack of strong consistency between peers in Bitcoin. In this section, we explore how the lack of transaction finality is exploited by malicious actors in the network and the state of the art methods to address the lack of strong consistency.

2.4.1 Doublespend Attack

A doublespend attack is a deliberate attack by a malicious user to invalidate a transaction that consumes the same inputs as the doublespent transaction. A doublespend attack is often mounted to redirect the outputs of a broadcasted transaction to the spender, which essentially is a chargeback against the recipient of the transaction. Firstly, a malicious user may mount a doublespend attack by producing a fork in the Blockchain with a block that includes the doublespend transaction. The attacker must then mine on the branch which contains the doublespent transaction and compete with the network to produce a descendent block which will eventually allow the working branch to succeed as the main branch. Secondly, an attacker may mount a doublespend attack by means of information eclipsing [13] and exploiting the topology of the Bitcoin network. Such an attack exploits the inconsistency in the state of Bitcoin attributed by information propogation delay within the network. For example, suppose we have a transaction tx_i which spends the same output as tx_j and both transactions are broadcasted at the same time by two distinct peers A and B. Finally, transactions in Bitcoin are inherent replaceable as described earlier in section 2.2.2. A transaction tx_i can be replaced with a subsequent transaction tx_{i+1} if tx_{i+1} pays a higher fee than tx_i assuming miners in the network are incentivised only by the value of transaction fees obtainable from accepted transactions.

Figure 2.6 illustrates a RBF doublespend attack that is executed after tx_0 is broadcasted to the Bitcoin network and prior to the commitment of tx_0 in the Blockchain. The doublespend attack enables user A to be able to doublespend the unspent output to invalidate the original transaction and redirect the unspent

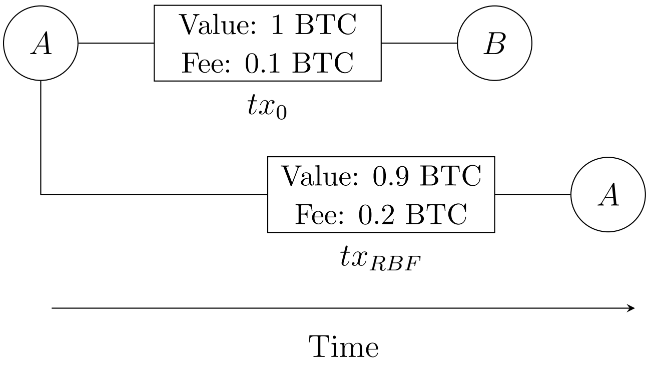

Figure 2.6: Doublespend attack by RBF

output back to herself.

2.4.2 Malleability Attack

A mallaebitility attack is executed by exploiting the malleability of transactions in Bitcoin. As signatures in Bitcoin do not sign the contents of scriptSigs, it is possible for a malicious actor to modify the contents of the scriptSig of a transaction without invalidating a transaction. Such an attack is effective against transactions which form a series of chained transactions. For example, for a sequence of transactions tx_{i+1} that are dependent on some transaction tx_i. Suppose the input script of tx_i manipulated and does cause an invalidation, we denote the manipulated transaction as tx_j. If tx_j was accepted instead of tx_i, this would result in the invalidation of all subsequent transactions that are dependent on the acceptance of tx_i as tx_j has successfully doublespend the UTXOs referenced by tx_i. As a consequence of transaction malleability, it is often not recommended to accept zero confirmation transactions. *Segregated Witnesses* is a recent proposal [12] proposed to remove the claiming script and signatures from all future Bitcoin transactions, thus eliminating the potential of transaction Malleability. However, the proposed scheme will only work for wallet softwares that implement the proposed scheme.

Chapter 3

Exploring RBF Acceptance

In this chapter, we will present our findings on the acceptance of RBF transactions within the Bitcoin network. The high acceptance rate of RBF transactions enables us to deploy a pure RBF-enabled insurance scheme which would enable significant savings when setting up the insurance scheme.

3.1 Hypothesis

It is well-known that only clients implementing protocol version v0.12 and above will be supporting the acceptance of RBF transactions and this is already implemented in the reference client. However, the acceptance of RBF transaction is still left as a policy decision for the owner of nodes to decide whether or not to accept RBF transactions. Furthermore, there exist various clients that deviate from the reference client and it is unclear whether some of these clients support the acceptance of RBF transactions.

Based on figure 4.1, assuming that every node which implements protocol version 0.12 and above accept replacement transaction, we expect to see that at least 56.4% of nodes to accept RBF Transactions.

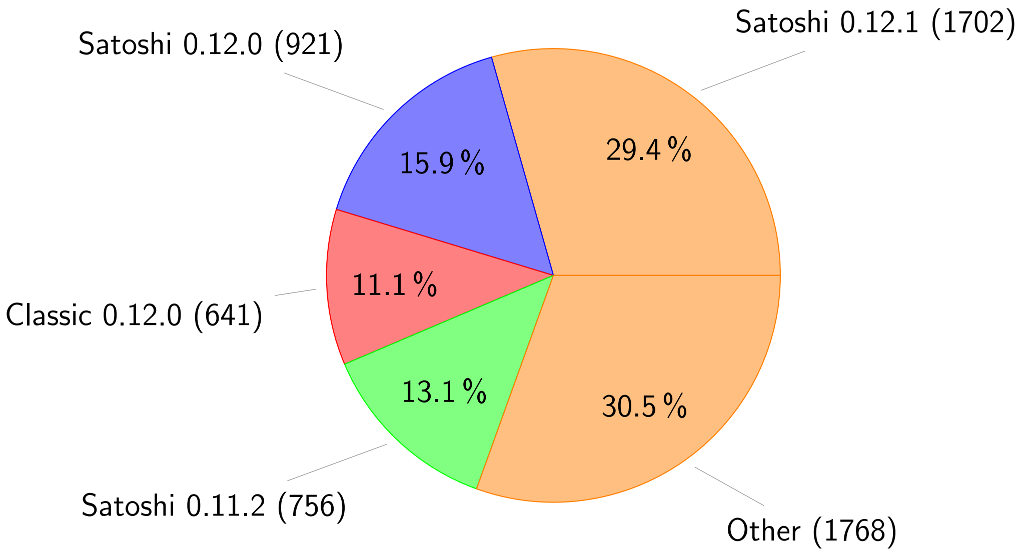

Figure 3.1: User Agents in the Bitcoin Network[1]

3.2 Method

The creation, signage and broadcasting of transaction requires a client to be running at all times. The Bitcoin reference client implements a daemon which is connected to the network and responsible for all protocol level communication pertaining to transactions and blocks. The reference implementation is also coupled with a wallet implementation which keeps a set of private keys used to sign and authorise outgoing payments. The wallet is also used to generate a Bitcoin address which enables the wallet owner to receive funds. As the reference client is intended to be run as a background process, the reference client exposes a set of JSON-RPC interfaces which enables third-party clients to execute RPC commands. The third-party clients may be executed locally or remotely. The reference client also provides a default implementation of the RPC client, known as `bitcoin-cli`, it implements the full set of RPC interfaces exposed by the daemon.

To create a transaction in the reference client, the client wallet must be funded with some Bitcoins (i.e. there must be one or more UTXOs that are spendable by the set of keys held in the client wallet). The ingredients required to set up a transaction are a set of UTXOs, and a set of payment addresses and the output values. To obtain the set of UTXOs spendable, the `getunspentoutputs` RPC command can be invoked. Once the set of UTXOs have been found, we can create simple non-replacable transactions with the unspent outputs tracked by the wallet. Once a transaction is created, the UTXO consumed in the newly created transaction will be marked as spent and it will no longer be possible to create new transactions with the UTXOs. The reference RPC client implementation features a set of convenience functions which allow a wallet user to create many types of simple transactions. To create transactions with complex redeem conditions, it will be necessary to create a raw transaction from scratch by manually specifying creating inputs with the correct signature hash type and redeem scripts.

Currently, the creation of replacable transactions is not supported by the reference client although the reference client has begun supporting mempool replacement policies in the client daemon. As a consequence of this, to create a replacable transaction, it is necessary to create a raw transaction. A raw transaction is created by specifying the list of inputs manually to be included in a transaction and the list of outputs to be included in a transaction. To signal replacability of a transaction, the `nsequence` field of an input simply has to be set to an integer less than `0xffffffff`. Once the transaction is created, it can be signed with the `signrawtransaction` command and then broadcasted to the network with the daemon by invoking the `sendrawtransaction` command. As the creation of transaction manually does not account for fees, it is necessary for the creator to invoke the `estimatesmartfee` command to get a rough indication of the fee required to fund the transaction.

When transactions are broadcasted by the reference clients, only rejected transactions will be accompanied with a reject message from the node rejecting said transaction. Otherwise, if there is no response, it is assumed that the transaction has been accepted by the node and will be relayed by the node to other connected nodes. Transactions are relayed with the use of `inv` messages containing sets of transaction hashes and block hashes. Once a node received an `inv` message, it will check each entry in the message to check if any of the transaction or block is unknown to it. Unknown transactions or blocks may be queried against nodes that contain information pertaining to them using the `getdata` message. The response to `getdata` messages are `tx` and `block` messages. On receipt of `tx` messages, the node will process the transaction to check if there exist a conflicting set of transactions will invalidate any transactions previously accepted. If there is, the node will respond with a `reject` message to indicate that the transaction is rejected by the node with the rejection reason. In the case of an accepted transaction, the transaction will be stored in the mempool and relayed to other connected peer in subsequent `inv` messages.

To create subsequent replacement transactions, due to the lack of support for replacement transaction in the current client, it is necessary for the user to keep track of the list of UTXOs used in the previous replacement transaction in the event a newer replacement trasaction is to be created. This track set enables the user to recreate a new replacement transaction which spends the same set of UTXOs. Prior to creating a new replacement transaction to replace the older replacable transaction, it is necessary for the user to explicitly invalidate the previous transaction to be replaced by invoking the `abandontransaction` command. Once this is completed, the UTXOs used in the previous transaction will be available for use again in the subsequent replacement transaction. Therefore, to effectively measure the acceptance of replacable transaction, we must first create, sign and broadcast an initial transaction which is marked as replacable. Subsequently, to check if a node actually supports replacement transactions, we then create, sign and broadcast the second transaction which spends the same UTXOs as the first transaction. If no `reject` message is received, we can assume that the node that has received the replacement transaction has effectively accepted the RBF transaction.

3.3 Challenges

Measuring the acceptance of RBF transactions is difficult as there is no concrete way of knowing that a RBF transaction has been accepted or rejected. While we have discussed that it is possible to know that a transaction is rejected if a reject message was received in response to the sending of a tx message. The option to reply with a reject message is a policy and not necessarily respected by other client implementations. Another definitive method to is to query the memory pool of connected nodes directly. But doing so periodically is not a viable option as each mempool request to a reference client node will incur an increase in banscore as a preventive measure against potential Denial of Service attacks as the operation to query the memory pool is an expensive computational operation. Furthermore, it is also left as a policy for clients to implement the response to a mempool message. A client could easily respond with a subset of the transactions in its mempool.

As Bitcoin is a fully distributed peer to peer system, it is impossible to be able to reach every single node on the network as some nodes may have implemented whitelisting and other networking policies. Nodes connecting behind a router may also be impossible to reach due to network address translation as the route to the host will not be possible to be worked out. Furthermore, as a consequence of these reasons beyond our control, it is also difficult to obtain a reliable count of current online nodes in the network.

The only way to test acceptance of RBF transactions is to send transactions directly to nodes and see what is the response from the sending of the tx message. As the set of transactions in Mempool is a policy, transactions that are rejected due to conflicts may not be communicated to the sending peer. To actually know for sure if a transaction was formally accepted, it will be necessary to send a direct mempool message which will increase the querying node's ban score. Transaction acceptance in Bitcoin must meet a minimum fee requirement, therefore it will be necessary to spend Bitcoins to test the acceptance of RBF transactions. The cumulative total for this part of the work is limited to 0.0271 BTC, which is equivalent to AUD 10[1]. Given that the minimum transaction fee on Mainnet is currently 0.00011558 BTC per KB[2], and the size of each RBF transaction is approxmately less than 1 KB, but for simplicity sake, we estimate

[1] effective 31st May 2016

[2] obtained using the estimatefee command from bitcoin-cli

our transactions to be at least 1KB. We can only effectively test for 39 replacement transactions, accouting for the necessity to raise fees for each subsequent replacement transaction.

False positive measurements may also be made in the case where there is no `reject` message response to a rejected transaction. Currently, for every `tx` message that is sent and if there is no `reject` message which follows before a block is generated and the RBF transaction is minted into the block, we assume that the RBF transaction has been accepted by the node. However, this may not be the case for the reasons we have discussed earlier and there is no way for us to check for sure if a transaction has been rejected without an explicit `reject` message sent from the target node. To counteract this problem, we can make a `mempool` query request each time a RBF transaction is sent and there is no `reject` response. However, there is still the case where the response to the `mempool` message may not necessarily represent the complete set of transaction in the mempool of the target node.

3.4 Implementation

Usage of the reference client to perform measurements is inadequate. Firstly, the reference client is not designed to support the establishment of large number of outbound connections, therefore it takes a long period of time to set up outbound connections. On inspection of the source code, this is attributed to the fact that outbound connections are executed in a single thread sequentially. Secondly, the reference client produces a lot of noise as it is responsible for a responding to the complete set of protocol messages. This makes analysis and data collection difficult which led us to develop our own client, Bitwatch which implements a subset of the Bitcoin protocol.

Bitwatch is a minimalistic client which implements a subset of the Bitcoin protocol. It only responds to *critical* protocol messages which may trigger a connection teardown if left unattended. Bitwatch also does not implement transaction verification as there is no implementation of the blockchain in the client.

Therefore, Bitwatch must still run in parallel to a full node. Bitwatch can establish connections faster than the reference client as it simultaneously connects to all known nodes concurrently. To reduce the noise attributed by exchanges of protocol messages, Bitwatch does not relay blocks or transactions and it does not send any messages unless absolutely required to maintain connectivity. As Bitwatch does not maintain a blockchain, it is also unopinionated when it comes to whichever transaction to accept or reject. Every transaction relayed to Bitwatch will be stored in its mempool and the our customised mempool implementation also supports the storage of conflict transactions to facilitate double spend monitoring.

For the purposes of the thesis, Bitwatch was also customised to enable the running of our RBF probing experiments. First, Bitwatch is enabled to connect up to N peers. After N connections have been established, the initial transaction is then broadcasted to all peers, followed by the RBF transaction. Between the transmission of both transactions, there is a 10 second window to enable transactions to propogate throughout the network before a RBF transaction is broadcasted. After the broadcasting of the RBF transaction, a `mempool` message is sent to probe the state of the memory pool on each connected peer. On return of the `inv` messages, we check either the initial RBF-enabled transaction or the RBF replacement transaction has been successfully accepted into the mempool of the peer.

3.5 Results

Testnet As testnet connections were harder to obtain, we lowered the number of connections for testnet to 80 connections. In our experiment, we ran two trials and we have connected up to 118 unique peers running 18 different variants of Bitcoin clients. The summary of our findings is detailed below:

Our findings in Testnet were inline with our intial hypothesis, where we expected most reference clients implementing protocol version v0.12 and above to accept RBF replacement transactions. On top of that, all RBF doublespend

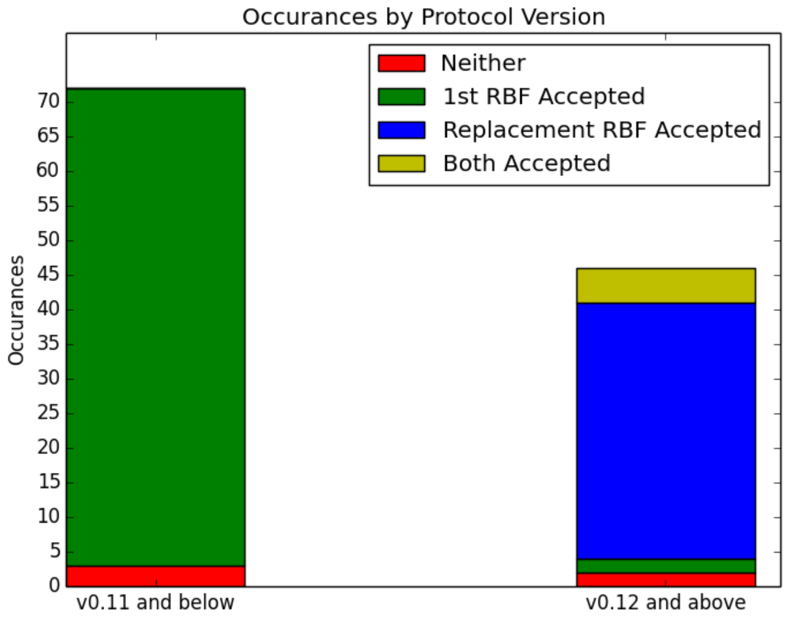

Figure 3.2: RBF Transaction Acceptance in Testnet

transactions were eventually committed to the Testnet Blockchain, which worked out just as expected. Most, if not all reference clients running on protocol version 0.12 and above accepted the RBF transaction which pays a higher fee than the initial opt-in RBF transaction. On instances where transactions that did not signal opt-in RBF were broadcasted, none of the RBF replacement transactions were accepted, thus confirming that all nodes we have surveyed implemented the correct behaviorial response for non opt-in RBF transactions. All nodes which run on older protocol versions (i.e. v0.11 and below) only accepted the initial RBF-enabled transaction and rejected subsequent RBF replacement transaction which pays a higher fee just as expected. On specific instances, v0.12+ nodes indicated that both initial and replacement transactions were accepted into the mempool, which is an unexpected behaviour.

Mainnet For our mainnet measurements, we have configured Bitwatch to accept up to 200 connections for each trial. In our experiment, we ran two trials. From both trials combined, we connected up to 374 unique peers running 28

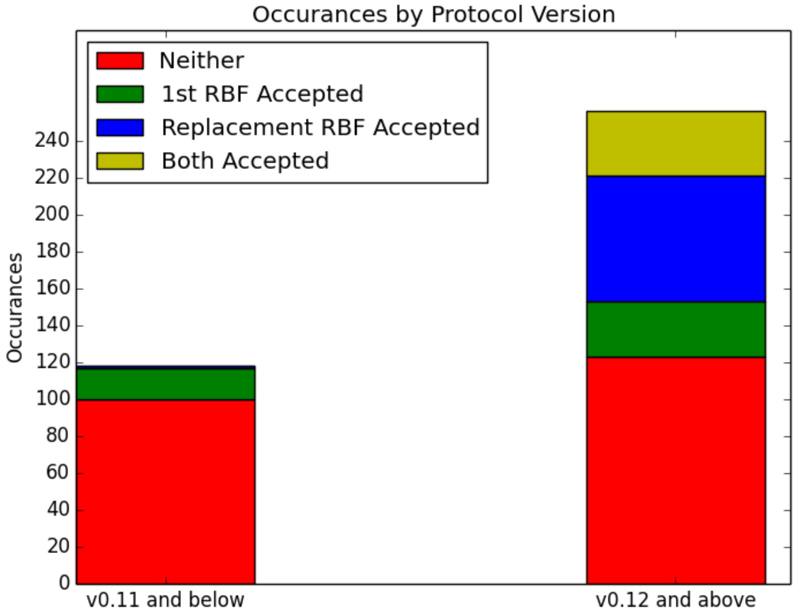

Figure 3.3: Mainnet RBF Acceptance with Classic Nodes

different variants of Bitcoin clients. The summary of our findings is detailed below:

Our findings in Mainnet were also inline with our expectation despite a signifcant number of nodes rejected Opt-In RBF transactions or simply did not respond to the `mempool` message we have sent manually to probe for the full list of transactions accepted. On our Mainnet trials, we were also able to connect to a significantly larger proportion of v0.12+ nodes which is above the representative proportion as show in figure 3.1 indicating that v0.12+ nodes are signficiantly more reachable than older nodes within the network. Removing all double rejection nodes, we can see that despite we have a large number of connections to v0.12+ nodes, it is apparent that at least 20% of v0.12+ nodes disabled mempool transaction replacement, thus in effect ignores higher paying RBF replacement transactions. This alludes to an apparent deviation from expected transaction acceptance policies on Mainnet and is the main cause of uncertainty of the eventual commitment of RBF transactions in the Blockchain as higher paying RBF

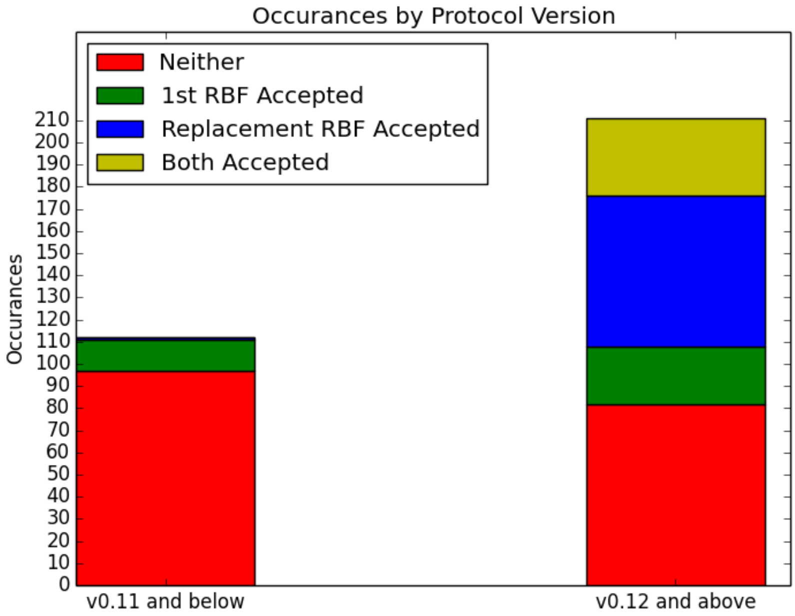

Figure 3.4: Mainnet RBF Acceptance without Classic Nodes

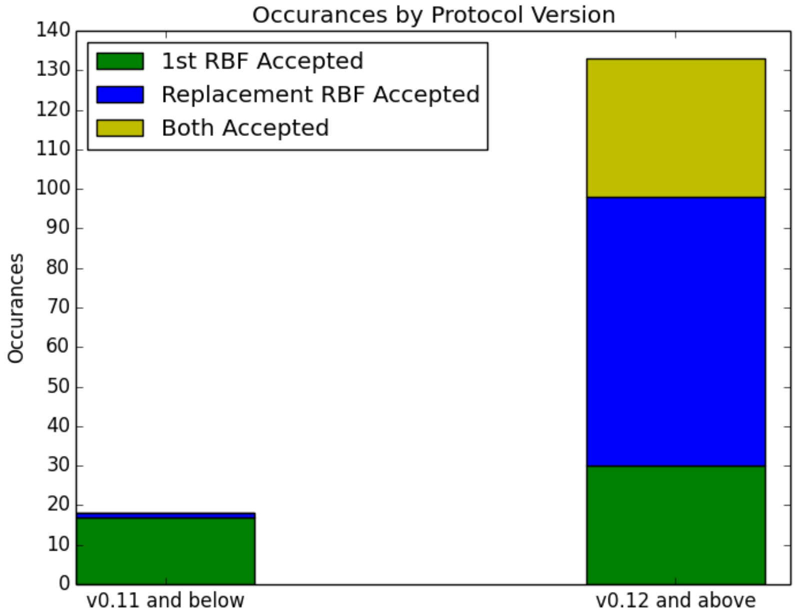

Figure 3.5: Mainnet RBF Acceptance without Opt-In RBF Rejection Nodes

transaction is not guaranteed to be selected despite paying a higher fee. Out of 10 RBF Doublespend attempts we have conducted, we were only successful at doublespending 1 out of 10 attempts which does not correlate to the proportion of nodes who accepts RBF replacement transactions in our findings. The only instance where a Doublespend RBF Transaction was successful was when the transaction was minted by the Elligius Mining Pool. Therefore, we can infer that major mining pools (i.e. BTCC, Antpool and F2Pool) do not mine RBF replacement transactions and still operate based on the old First-Seen-Safe (FSS) rules. The behaviour of v0.11 and below nodes were also inline with our expectations and it is apparent that some nodes have applied a custom patch which enables older nodes to accept RBF replacement transactions. A large proportion of mainnet nodes have also indicated that they accepted both the initial RBF enabled transaction and the RBF replacement transaction, thus it is difficult to infer eventually which transaction will likely be committed to the Blockchain.

3.6 Discussion

Due to time constraints, we were not able to optimize the codebase for Bitwatch to facillitate a much higher number of connections and it was not possible to implement the complete set of RPC interfaces which would enable experiments to be run at any time whilst Bitwatch was running. Despite using a batch connection method, with each batch connection initiating 50 connections in parallel everytime, most outgoing connection to nodes timedout despite set with a high timeout duration of 5000ms. This problem was apparent in both Mainnet and Testnet environments and we assumed that it was attributed to the most nodes maxing out their maximum connection limit or were too busy to respond to our connection requests.

We expected that all peers which utilize v0.12 (i.e. Satoshi v0.12++) and above of the reference client or any peer which implements protocol version v0.12+ to accept RBF transactions by default. Furthermore, we also expected that all peers below v0.12 would reject RBF transactions. Based on our findings, it is clear that not all nodes running protocol version v0.12+ in Mainnet supports mempool transaction replacement for opt-in RBF transactions as shown in figure 3.2, which led to a significant non-determinism in the commitment of RBF transaction in the Blockchain. We also intentionally excluded Classic nodes from our analysis as the proportion of Classic blocks in the Blockchain is signficantly less than the number of blocks minted with the reference client (i.e. Satoshi nodes)[1].

Our method relies on the correct response of nodes to `mempool` messages, where it is expected that nodes receving the message will respond with an `inv` message containing the complete set of transaction hashes that is currently in its mempool. As mempool information is meant to be kept private, it is left to the discretion of the node owner to ensure the expected behavior was implemented for the `mempool` message. From an outsider's perspective, there is no way that we can verify if the node has implemented the response we expect correctly, therefore we expect that the results may deviate from the actual mempool content of the

[1] Proportion of Bitcoin Classic Blocks effective 1st June 2016 from http://xtnodes.com/

probed nodes. Furthermore, it is unclear why there are incidence where nodes returned that both conflicting transactions were accepted to its memory pool. This could be attributed to an incorrect response implementation or the mempool implementation has deviated from the reference client, where it is expected that the mempool should only contain transactions it has decided to select to commit to the Blockchain.

Despite the signficant variations in our measurements in Mainnet, we anticipate that as node owners upgrade to Satoshi v0.12+, the acceptance of RBF transactions will gradually increase over time as observed in our Testnet findings. We can also conclude that right now, it is unsafe for payees to accept zero confirmation transactions in Mainnet due to inconsistent transaction acceptance policies upheld by different node owners and mining pools, which further highlights the need for an insurance scheme for zero-confirmation transactions to provide a deterministic outcome guarantee for payees relying on the acceptance of zero-confirmation transactions.

Chapter 4

Bitsurance Contracts

In this chapter, we present *Bitsurance*, a novel approach to insure zero confirmation Bitcoin transactions. The insurance scheme is designed to increase the probability of a definitive outcome for a receiver of an insured transaction regardless of the commitment outcome of said transaction in the Blockchain. Firstly, we present the two designs for the insurance contracts, namely the RBF-Enabled variant, currently only support by nodes running Bitcoin protocol v0.12 and the RBF-Disabled variant which are supported by older nodes running protocol version v0.11 and below which are currently used in the Bitcoin network by independent participants. Finally we present an analysis of potential attack scenarios and the game theory which justifies the effectiveness of the insurance scheme.

A standard transaction in Bitcoin typically involves a payer and a payee. As Bitcoin transactions typically pay to an address which does not expose the identity of a recipient nor exposes the identity of a payer. Payers are incentivised to doublespend transactions as there are no repercussions for doing so. In an ecosystem where Miners are incentivised to include transactions which pays the highest fees, it is always possible for a payer to doublespend a payment transaction by doublespending the UTXO used to fund the payment transaction with a conflicting transaction. To incentivse miners to select the conflicting transaction in favour of the original payment transaction, a doublespender can simply pay higher transaction fees. Ultimately, a rational and profit-maximising miner will select the conflicting transaction to include into a minted block in favour of the original payment transaction. As a consequence of this, it is inherently unsafe for a payee to accept zero-confirmation transactions in Bitcoin. To address this problem, we propose an insurance scheme which insures the payment transaction and guarantees a payout for the payee regardless of the eventual outcome of a doublespend attack against the insured transaction.

4.1 Insurance Scheme

Our proposed insurance scheme, known as *Bitsurance* utilises Bitcoin transactions to produce *contracts* in a trustless manner. A contract is a transaction which is dependent on the eventual commitment of a set of transactions in the Blockchain. Depending on the type of contract within the insurance scheme, some contracts may share a mutual exlusive relationship where the commitment of one contract would lead to the invalidation of other transactions. The insurance scheme is composed of four main contracts and two transactions, known as a *Commitment Contract*, *Staking Contract*, *Insure Contract*, *Claim Contract*, zero or more optional *Fee Bump Transactions* for the RBF-Disabled variant and finally a *Payment Transaction.*

4.1.1 Definitions

For all contracts and transactions described in the Insurance Scheme the following notations apply:

- P denotes the Insured Sum. All our cases assume that the entire output of the insured payment transaction is insured. However, this is not a strict requirement of the scheme.

- Pr denotes the *Premium* payable to establish the insurance scheme. For most of the contract transactions, the successful inclusion of the input funding the Premium is a requirement for any of the contracts to be successful. Therefore, it is impossible for a payee to doublespend the Premium and benefit from the insurance scheme concurrently.

- F denotes the current transaction fee dynamically set by the Bitcoin network. Depending on the amount of pending transactions that are waiting to be confirmed, the transaction fee for new transactions is dynamic. For simplicity reasons, we assume F is the same for all contracts regardless of its redeem conditions. But in actual fact, the actual transaction fees would be rise proportional to the complexity of the redeem condition.

4.1.2 Payment Transaction

By design, the insurance scheme requires that the input used to fund an insured payment transaction to be referenced from a confirmed transaction. This is to prevent a series of cascading invalidations that would result in the invalidation of a payment transaction which will trigger a compensation payable by the insurer. This requirement can be waived if the unconfirmed transaction referenced is insured by a similar insurance scheme, however an analysis of this is beyond the scope of this thesis. The insurance scheme also prohibits the usage of RBF transactions for the Payment Transaction, as each subsequent replacement would result in change in the hash of the payment transaction (TxID). Changes in the hash of the payment transaction would result in cascading invalidations of linked transactions in the insurance scheme. While it is not strict to avoid RBF transactions for the payment transaction, any replacement will be perceived as a doublespend by the Insurer and will prompt countermeasures to be deployed to incentivise miners to accept the original insured transaction over subsequent valid replacement transactions.

4.1.3 Commitment Contract

The commitment contract is the first transaction that is created by an insuree to initiate the insurance creation process. It is used for an insuree or recipient of the payment transaction to prove that she is a recipient of a payment transaction through the spending of the unspent output of the payment transaction. This disallows either party to gamble on someone else's transaction. The commitment contract once created should be broadcasted to the network and may be independently verified by an anonymous third-party insurer running a Full Node in the network.

Inputs The contract must contain one or more UTXO from payment transactions to be insured and one or more UTXOs from confirmed transactions to fund the premium and associated transaction fees to set up subsequent contracts. The signature hash type for inputs in the commitment contract must be `SIGHASH_ALL` to ensure immutability. The `nsequence` field of the commitment contract should also be set to `0xffffffff` to opt-out of replacements. The combined sum of the inputs should be $P + 2F + Pr$ where P is the insured sum, F is the current transaction fee for each transaction and Pr is the premium payable to the insurer.

Output The output of the contract should be payable to a multisig address controlled by both the insurer and insuree. Essentially, by doing so, this guarantees that the payee can no longer defraud the insurer by double-spending against the insurer. The contract should contain exactly one output which pays $P + F + Pr$ to said multisig address.

Redeem Condition To cater for situations where an insurer may renege on continuing with the insurance setup process, we can setup a complex redeem condition which allows the insuree to spend the UTXO of the commitment contract after the commitment contract is minted into the blockchain. If no subsequent contract is minted, then it is fair to assume that the insurance scheme has yet come into effect, therefore an insuree may withdraw from the insurance scheme. To do so, a redeem condition allowing the insuree to spend the UTXO of the commitment contract after N confirmations is added to the UTXO script. The specific script to implement this redemption condition may be found in the appendix.

Issuing Party The commitment contract should be created and signed by the insuree from information provided by an Insurer, then broadcasted to the Bitcoin network. This contract should also be sent directly to an insurer, so she may be able to set up subsequent contracts to complete the setup of the insurance scheme.

4.1.4 Staking Contract

Upon receipt of a commitment contract from an Insuree. The Insurer is obliged to create a staking contract which reserves the insured sum for compensation in the event of a successful double spend attack. The staking contract serves two purposes within the insurance scheme. Firstly, it guarantees that the Insurer has sufficient funds to compensate an Insuree in the event of a successful double spend attack. Secondly, it prevents an Insurer from double spending the committed insured sum against an Insuree. Thus, allowing the scheme to be a completely trustless scheme.

Inputs The staking contract must contain one or more UTXOs from confirmed transactions minted in the Blockchain. It is left as a policy decision for the insurer to decide the confirmation threshold for the UTXOs referenced. The combined sum of the referenced UTXOs must at least be greater than or equal to $P + F$. The inputs of the staking contract must be signed with signature hash type `SIGHASH_ALL` with `nsequence` fields set to `0xffffffff`.

Outputs The value of the output payable to the same multisig address as in the commitment contract must be P with a complex redeem script. Any change from the Input may be refunded back to the Insurer with a standard script.

Redeem Condition In the event no compensation is required to be made to the Insuree, a complex redeem condition must be setup to enable the insurer to spend the UTXO of the staking contract after the insurance contract is minted into the blockchain. In the event that the insurance contract is not minted into the blockchain as a consequence of a successful doublespend attack, the UTXO of the staking contract will be used to fund the *claim contract* which is described later. Therefore the redeem script should allow a 2 of 2 multisig spend during zero confirmation state, so that the UTXO of the staking contract may be used to fund the input of the claim contract. However, should the claim contract be invalidated as a result of the successful inclusion of the insurance contract into the blockchain, then the UTXO of the staking contract should be spendable by the Insurer after N confirmation(s).

Issuing Party This contract should be created and signed by the insurer and broadcasted to the network immediately. As a precaution, the contract should be sent back to the Insuree as a proof of commitment by the Insurer.

4.1.5 Claim Contract

The claim contract is a transaction used by the Insuree to claim a compensation in the event of a successful double spend attack. The claim contract requires a locktime that is bound by the block height at `CurrentBlockHeight + k + 1`, where k is the block at which the insure transaction or successful doublespend transaction is commited. The claim contract may only be used after it is definitely known that the insurance contract is invalidated.

Inputs The claim contract must contain the same referenced inputs used to fund $Pr + 2F$ in the commitment contract. This is a strict requirement as we need to use these inputs to invalidate either contracts depending on the outcome of the insurance scheme. The claim contract must also contain the UTXO from the staking contract, this UTXO is used to fund P, which is the compensation amount. All inputs must be signed with the signature hash type `SIGHASH_ALL` with `nsequence` field set to `0xffffffff`.

Outputs This contract must contain an output which pays P to the Insuree and an output which pays $Pr + F$ to the Insurer.

Redeem Script There is no need for complex redeem conditions for the UTXOs of this contract. A simple scriptPubKey is sufficient.

Issuing Party The Insurer is responsible for creating this contract. Upon creation, the contract should be partially signed and returned to the Insuree along with the staking contract and the insurance contract. Upon receipt by the Insuree, the Insuree can then sign this contract which enables the spending of the UTXO from the staking contract as the redeem condition for the UTXO requires 2 of 2 signatures before the script time lock expires.

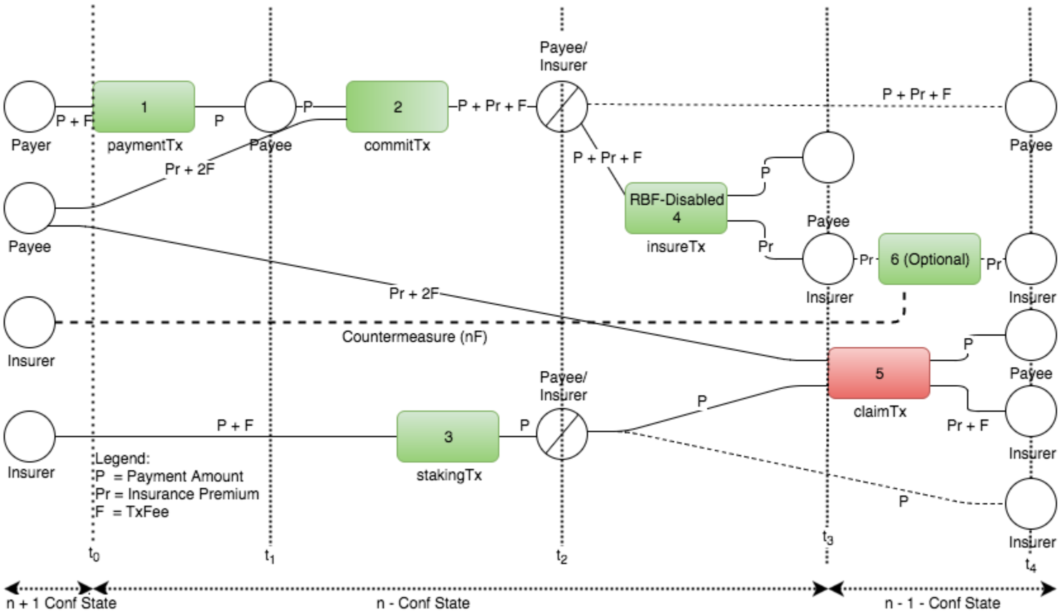

Figure 4.1: CPFP Insurance Contracts

4.1.6 Insurance Contract

An insurance contract is the transaction designated to pay the insured sum to an insuree and the premium amount to the Insurer. This contract is also utilized for the insurer to perform countermeasures against a potential doublespend attack. By design, an insurer is incentivised to ensure that this contract is eventually minted into the Blockchain to be best of her effort as this contract will invalidate the claim contract once minted into the blockchain. The tradeoff of selecting the CPFP design is that subsequent fee bump transactions must include a much higher transaction fee which accounts for the fee bump transaction itself and the increase in fee value of preceding transactions along the chain of referenced transactions.

Inputs During creation, the insurance contract must contain the exact UTXO from the commitment contract. For the RBF-Enabled solution, Inputs must be signed with the signature hash type `SIGHASH_SINGLE | SIGHASH_ANYONE_CAN_PAY` with the `nsequence` field set to a number less than `0xffffffff` for future transaction replacement with will be used to increase transaction fees payable to miners.

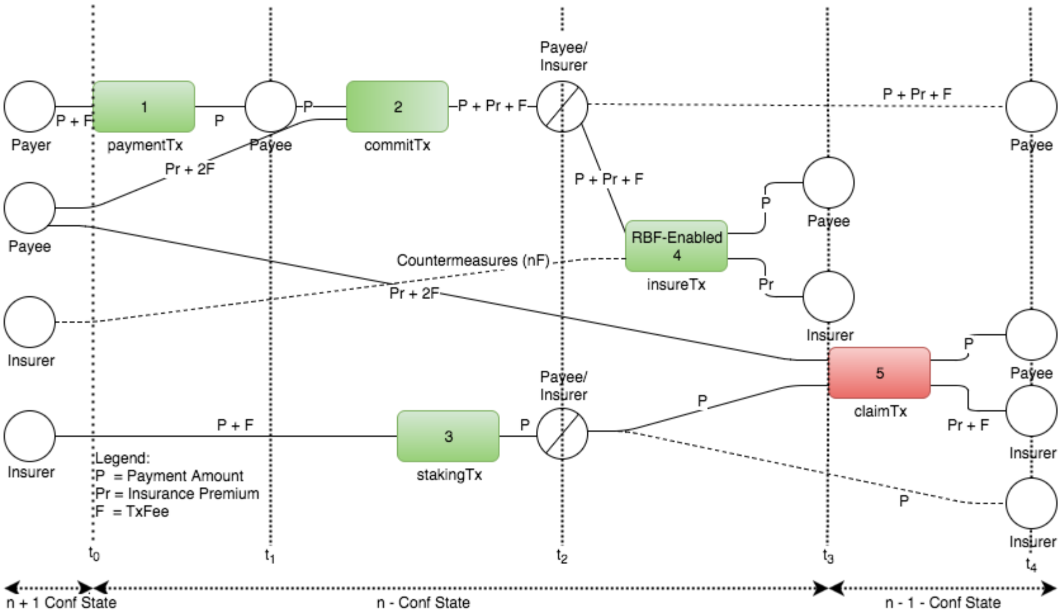

Figure 4.2: RBF Enabled Insurance Contracts

SIGHASH_SINGLE was specially chosen for the RBF-Enabled solution so that it is possible for the insurer to include additional inputs in the event of an ongoing doublespend attack, which in effect increases the transaction fees for the insurance contract without invalidating the insuree's signature. For the RBF-Disabled variant, Inputs must be signed with only SIGHASH_ALL.

Outputs This contract must contain exactly two outputs which pays P to the insuree and Pr to the insurer. These outputs may be paid to a simple scriptPubKey.

Redeem Condition No complex redeem conditions are required for any outputs in the insurance contract. A standard scriptPubKey is sufficient.

Countermeasures In the RBF variant of the insurance scheme, countermeasures are deployed by replacing this contract with a newer contract which pays a higher transaction fee to miners. In the CPFP variant, countermeasures are performed by appending fee bump transactions to the output which pays the Insurer. For each CPFP depth increase, the fee included for each CPFP transaction must also increase by a factor of N, where N is number of fee bump transactions chained to the insurance contract.

Issuing Party The insurance contract should be created and partially signed by the Insurer. Once this is completed, this insurance contract should be sent to the Insuree concurrently with the signed staking contract and the signed claim contract. Once the Insuree has confirmed that the staking contract has been broadcasted and accepted by the network and the claim contract is valid, the Insuree may sign the Insurance Contract and return the insurance contract to the Insurer. Simultaneously, the Insuree may also broadcast the Insurance Contract to the network.

4.2 Contract Lifecycle

The lifecycle of a contract is dependent on the eventual commitment of the dependent transactions referenced by each contract. Formally, we may write that for each of the input referenced in each contract transaction, the commitment outcome of each contract is conditional on the successful commitment of the transactions used to the fund said contract. We express this conditional requirements as $Commit(\alpha) \rightarrow \beta$, where $Commit(\alpha)$ is the minting of a set of transaction, α in a newly appended block to the Blockchain. α is the set of transactions containing the unspent outputs used to fund the inputs required in a contract transaction, β. Therefore, for each contract, the validity of each contract may be expressed as $Commit(\alpha) \rightarrow \beta$ where $Commit(\alpha) \rightarrow \beta$ evaluates to the validity of the contract β. For each transaction, the set α is as follow:

- Commitment Contract - Let A be the set of payment transactions to be insured and $Commit(A)$ is the event at which the set of payment transactions is committed to the Blockchain. Let B be the set of transactions used by the payee to fund the Premium and transaction fees of the insurance scheme. The insurance scheme asserts that B must be a set of confirmed transactions, therefore $Commit(B)$ yields true at all times.

- Staking Contract - Let C be the set of transactions used to fund the Insured Sum and the transaction fee for the staking contract. Again, the Insurance Scheme asserts that C must be a set of confirmed transactions, therefore $Commit(C)$ yields true at all times.

- Claim Contract - Transitively, the claim contract requires the set B and C.

- Insurance Contract - Again, by transitivity, the insurance contract only requires the set A and B.

Figure 4.3 and 4.4 illustrates a scenario where a doublespend attack succeeds and another scenario where an ongoing doublespend attack is successfully stopped by the insurance scheme. Green coloured transactions and UTXOs indicates

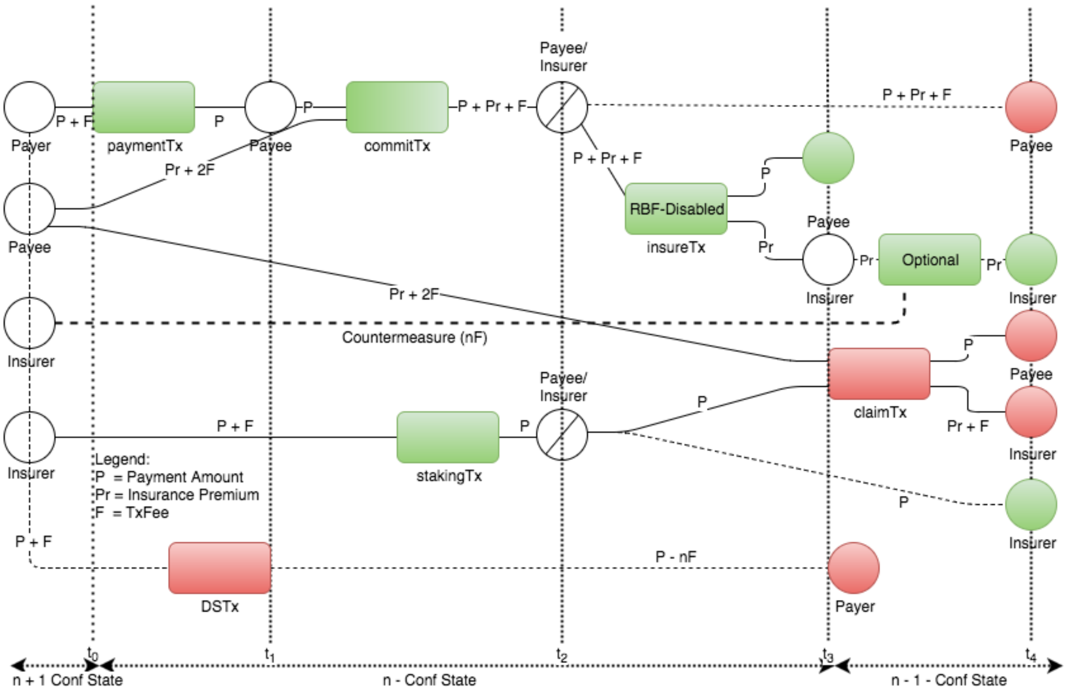

Figure 4.3: Failed Doublespend against Insurance Scheme

eventual commitment in the Blockchain, while red coloured transactions indicates eventual invalidation. As illustrated in both figure, the insuree is always guaranteed a payout of P regardless of whichever sequence of transactions get committed to the Blockchain, while the Insurer is also always guaranteed to receive a premium, Pr.

In summary, we can observe that there are two contracts that are dependent on a common output, which is the set B. Let T_C denote the Claim Contract and T_I denote the Insurance Contract. Deterrence against a doublespend attack requires that $Commit(A) \wedge Commit(B)$ evaluate to true, allowing for $Commit(A) \wedge Commit(B) \rightarrow T_I$ to yield true. As the T_C is a timelocked contract, it does not matter if $Commit(C) \wedge Commit(B)$ evaluates to true, as it is only spendable after one block has been generated. With this relationship between contracts established, we can also assert that at all times, an insuree is always guaranteed P as $Commit(B) \wedge Commit(C)$ always evaluates to true, therefore $Commit(B) \wedge Commit(C) \rightarrow T_C$ will always hold true, which guaran-

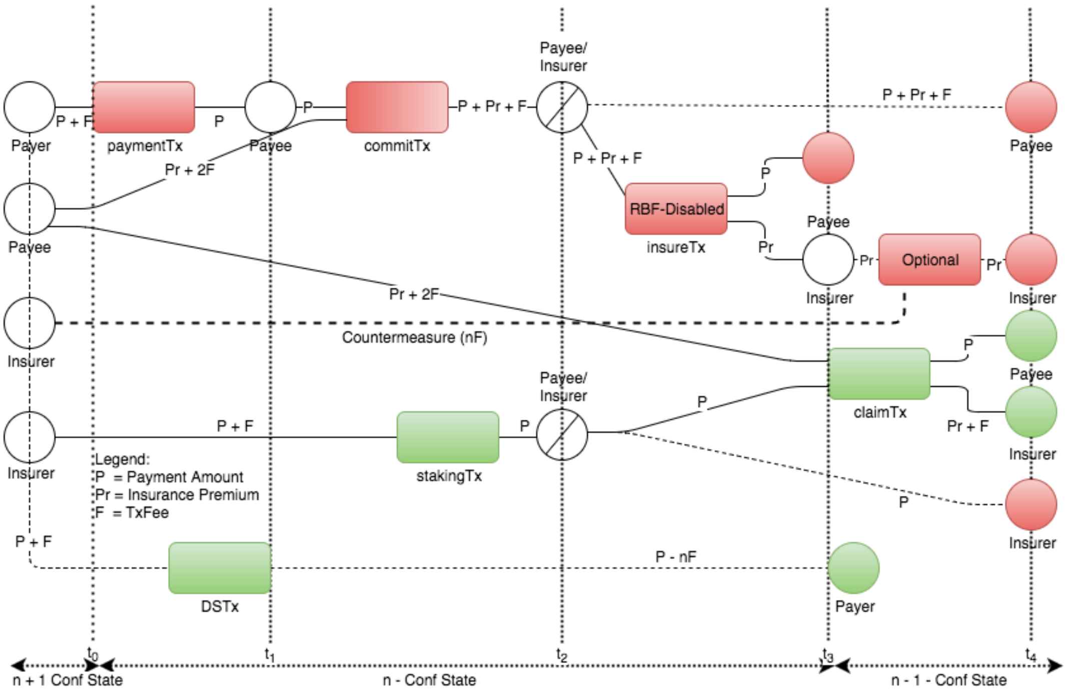

Figure 4.4: Successful Doublespend against Insurance Scheme

tees that the insuree is always entitled to P regardless of the successful outcome of a doublespend attack. Thus, simply by checking the result of $Commit(X)$, where $X \in \{A, B, C\}$, we can easily deduce if any contract in the insurance scheme is in a valid state.

4.3 Model

For the purposes of the insurance scheme, we assume thats miners in the Bitcoin network will eventually adopt a generic mining policy which consistently selects the highest yielding chain of transactions to be minted into discovered blocks. Every participating member in the insurance scheme effectively attempts to maximise their gains and the inherent trust model is that every party will attempt to collude in the interest of maximising their gains. As Bitcoin is a fully decentralised network, we assume that it is infeasible for neither participating parties in the insurance scheme to be able to collude with miners to arrange preferential commitment of transactions whilst selecting transactions to be minted into discovered blocks. Collusion in this case is dangerous, most of the game theoretic assurances of bitcoin break down in the case of collaboration between miners and select participants in the network.

4.3.1 Doublespend Vectors

The payer-payee model describes a generic insurance for a payment transaction that involves two parties, where one party is responsible for funding said transaction (i.e. Payer); while the other is responsible for receiving the funds involved in said transaction (i.e. Payee or Insuree). The generic transaction is the target transaction to be insured by a third party insurer. In this model, there are three possible doublespending scenarios.

- Payer doublespends Insured Sum against Payee. In this doublespend case, the insurance scheme is designed to prevent this from happening at the first place, therefore this case is covered.

- Insuree doublespends Premium against Insurer. With the design of the insurance scheme, the claim contract and the insurance contract are the contracts that provide a payout of the insured sum back to the Insuree. As both of these contract require the Premium as an input, an Insuree is never incentivised to doublespend the Premium against the Insurer.

- Insurer doublespends Insured Sum against Insuree. The production of the claim contract requires that the staking contract be broadcasted and accepted by the network. As a security measure, the Insurance Scheme requires that the Insurer pay the inputs used to compensate the Insuree into a multisig address. While this is not completely secure method of deterring a doublespend mounted by the Insurer, it makes it somewhat harder for an Insurer to doublespend the promised insured sum. If the Insuree is concerned about the doublespending by the Insurer, the Insuree could again insure the claim contract which ultimate guarantees the payout of the claim contract.

4.3.2 Doublespend Analysis

As shown in figure 4.5, there are two possible states for the insurance scheme to be in at any point in time after the successful setup of the insurance contracts. The insurance scheme does not take into account the double-spend risk beared by the insuree as it does not concern the insurer until all contracts are setup. Let α denote the event where all contracts are setup and the refund contract is broadcasted to Bitcoin network. The two states which follows after this event is the safe state, denoted S, where the refund contract will eventually be minted into the block and subsequently committed to the Blockchain. The attack state, denoted A represents the state at which a transaction which doublespends the payment transaction will eventually be minted into the block and subsequently commited to the Blockchain. E_1 represents an action undertaken by an attacker which incentivises miners in the network to accept the double-spend transaction in favour of the refund contract. While E_2 represents an action that counteracts E_1 leading to the eventual acceptance of the refund contract in favour of the double-spend transaction. At any point in time, either state may be accepted pending the discovery of a block and whichever transaction is most lucrative for a miner to mint into the block.

The discovery of blocks for Bitcoin (i.e. Proof of Work) is a Poisson process. Therefore, the time difference between each block may be modelled as a inter-arrival and waiting time distribution (i.e. exponential distribution) with a mean

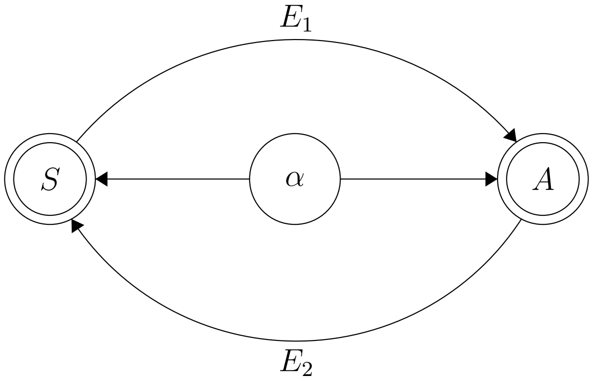

Figure 4.5: Finite State Machine for Transaction Insurance

time difference between the discovery of blocks of approximately 10 minutes as proposed in Satoshi's original paper on Bitcoin and further re-affirmed by Decker et. al. Therefore, we may express $X \sim E(1/600)$, where X denotes the random variable representing the time between the discovery of some block, B_i and the preceding block, B_{i-1}. If we approximate the time needed for each action E_2 to be approximately 20 seconds[1].

$$\begin{aligned} P(X \leq t) &= 1 - e^{-\lambda t}, \lambda = \frac{1}{600} \\ P(X \leq 20) &= 1 - e^{-(\frac{1}{600})20} \\ P(X \leq 20) &= 0.0327839 \end{aligned}$$

Assuming that an insurer may react immediately to a double-spend attempt, but accounting for propogation delay of countermeasure transactions provides us with the probility of success of a double spend attack $P(X \leq 20)$. Therefore, the expected loss of an insurer is bound to be $IS \times P(X \leq 20)$ where IS is the value of the insured sum which the insurer can easily charge back to an insuree to offset her potential losses.

In reality, both the double-spending party and the insurer have finite resources. Therefore, it is impossible for an attacker to attempt a double-spend by staking an infinite amount of Bitcoins as transaction fees to beat an insurer. Assuming

[1] based on 90th percentile propogation from Decker's bitcoinstats.com

that a malicious payer is motivated by maximising the net difference between the payment amount and fees attributed to double-spending by creating a higher paying transaction, the maximum amount a malicious payer would spend is also equal to the payment amount less double spend fees. Therefore, so as long as an insurer takes more than the payment amount less the double spend fees, it is always possible for an insurer to succeed in beating the malicious payer. In a scenario where a payer and payee are colluding to defraud the insurer, the situation becomes less favourable to the colluding party as the maximum amount that may be used to double-spend profitably is further reduced to the net difference of the payable amount less the transactions fees and cost of premium. Therefore, in both cases, so as long as an insurer has a staking amount of $2 \times IS$, the Insurer will always be in a position to beat a malicious double-spender.

Game Theory The double-spend action by a malicious payer and counter-measure reaction by an insurer may be modelled as a game and we can represent the eventual outcomes in a payoff matrix. The values in the game matrix represents the expected outcome of an action undertaken by both insurer and malicious payer. In this game, the payee is excluded as the insurance scheme in effect transfers all systematic risk associated with double-spends from the payee to the insurer. In the following payoff matrix, Player I denotes the insurer of the payment transaction, Player DS denotes a malicious payer attempting to double spend an insured payment transaction. S is the sum of the outputs in an insured transaction. F denotes the transaction fees payable by the payer to fund the payment transaction and P is the premium payable by the payee to insure the payment transaction. Assuming that a payee always insures a transaction where the payment sum is always larger than the transaction fee and the payment sum is always larger than the premium amount, we can further apply a contraint to the value of the variables $S > F$ and $S > P$.

As we can see from the payoff matrix in the non-colluding case, the nash equilibrium is always a double spend attack by the attacker regardless of whether an insurance scheme is in place and the insurer's dominant strategy is to always react with counter measures. This is always the case because the maximum bound for the transaction fees to be spent by a malicious double-spender is less than

		Player I	
		CM	∼CM
Player DS	DS	$(0.0328S - 2F, P - S + 2F)$	$(S - 2F, P - S)$
	∼DS	$(-S - 2F, P)$	$(-S - 2F, P)$

Table 4.1: Payoff Matrix for Non-Colluding Payer-Payee against Insurer

$S-2F$, for her to effectively make a profit from the doublespend attack. Therefore an insurer will always have to spend a maximum of $S - P - 2F$, which incurs an out of pocket expense less than $S - 2F$ to mount a success counter-measure against a malicious double-spender. By subscribing to an insurance scheme, a payee is always guaranteed an outcome of $S - 2F - P$ at all times.

		Player I	
		CM	∼CM
Player DS	DS	$(0.0328S - 3F - P, P - S + 3F)$	$(S - 3F, P - S)$
	∼DS	$(-S - 3F - P, P)$	$(-S - 3F, P)$

Table 4.2: Payoff Matrix for Colluding Payer-Payee against Insurer

Similarly, the same calculations will apply for the case of a colluding payer-payee model. The only difference this time is that the maximum spendable sum for double-spend attack is further reduced to less than $S-3F-P$ as the colluding payer-payee may be considered as a single entity. This effectively allows an insurer to spend up to a maximum of $S - 3F - P$ to defend against a malicious doublespend. Therefore, in both cases, we have successfully demonstrated that the dominant strategy for the game would effectively stem out possible double-spend attacks and the maximum staking amount for countermeasures is bounded, effectively enabling an insurer to effectively defend against malicious doublespends.

		Payer	
		DS	∼DS
Payee	I	$(S - P - 2F, 0.0328S - 2F)$	$(S - P - 2F, -S)$
	∼I	$(-S, S)$	$(S, -S)$

Table 4.3: Payoff Matrix for Payer-Payee with and without Insurance

While it is apparent that the insurance scheme does not fully deter a double-spender from doublespending an insured transaction, it is still imperative that a payee has access to an insurance scheme as shown in Table 4.3. As we can see from our analysis of the payoff matrix with the payee subscribing to an insurance scheme and without subscribing to an insurance scheme, it is always the case that a payer will always doublespend and a payee will always be insuring a transaction in the case of accepting zero-confirmation transactions. The top-left case is the nash equilibrium as both parties will attempt to maximise their gains from each other's actions, therefore we can assume that if every actor in the network are profit maximising agents, it is plausible to assume that every recipient who chooses to accept zero-confirmation transactions will eventually be attacked with a double-spend attack assuming that $0.0328S > 2F$

Chapter 5

Bitsurance Implementation

To execute the Bitsurance contracts presented in the previous chapter, we implemented a library of the insurance contracts which setups the contracts required for the insurance scheme on demand. Due to time constraints, we were not able to implement the entire insurance scheme as an independent service. The library implementation enables any interested party to setup an insurance service based on the proposed scheme. A test suite is also included to guarantee the correct implementation of the complete scheme.

5.1 Method

Our library implementation was done in Python with the use of the python-bitcoinlib. The library used exposes an API interface which enables us to create, sign and broadcast transactions. It is important to note that deployment of the insurance scheme with the reference client is not recommended as the reference client is not designed to facilitate many outbound connections and it does not track current UTXOs that are in zero-confirmation state. Our test suite currently covers all possible scenarios where either party participating in the insurance scheme may be honest or dishonest. Therefore, in totality there are 9 combinations of scenarios which are all covered by our regression test cases.

5.2 Specification

5.2.1 Premium Calculation

Premium calculation is left as a policy decision for any Insurer who decides to provide an Insurance Service. As a minimum, an Insurer providing Bitsurance as a service should charge a premium which compensates for the expected value of compensations as a result of successful doublespend attacks. Based on the model we presented in the previous chapter, we estimate this value to be approximately $0.003 * P$, where P is the payment sum expected to be received by the Insuree. As an example, assuming that a payer has paid a payee a sum of 1 BTC, the premium payable to insure this transaction will be 0.003 BTC to offset the expected value of successful receipts from doublespend attacks. This is also the default premium calculation formula provided by our library implementation.

5.2.2 Proposed Contract Exchange Protocol (CEP)

As Bitsurance is a trustless scheme which guarantees that neither party is capable of defrauding each other, the exchange of contracts must be executed in a specfic order. This order of contract exchange is dictated by the Contract Exchange Protocol (CEP). The sequence of steps to execute CEP is as follow:

1. Insuree sends Commitment Contract to Insurer. Insurer then checks the inputs of the Commitment Contract. Requirements for the inputs were previously described in Chapter 3.

2. Insurer produces Staking Contract which commits the UTXOs required to compensate the Insuree in the event of a successful doublespend attack. The staking contract is then signed and broadcasted to the network, then sent to the insuree. At the same time, a partially signed claim contract is also returned to the Insuree in the same message.

3. The insuree checks the received staking contract and claim contract. Once she is satisfied with the validity of the both contracts, she can broadcast the insurance contract which effectively initiates the insurance scheme. The

insuree is always guaranteed to collaborate as the disincentive to not collaborate would be a delayed claim payout due to the claim contract time lock.

5.3 Results

Due to the current diverse set of mining policy in Mainnet, we have decided to test our insurance contracts in Testnet instead as the mempool replacement behaviours in Testnet is significantly more predictable and is in line with our model. Using our insurance contracts, 10 of 10 RBF doublespend attempts were successfully deterred with the insurer commiting $P - F$ to raise the transaction fee of the insurance contract, where P is the insured sum and F is the transaction fee that was paid by the payer to fund the insured payment transaction.

The following are the steps undertaken to execute the test:

1. First, we ensure that our wallet had 4 UTXOs to facilitate the setup of the insurance scheme.
2. Secondly, our test script written in Python was executed which setups all the contracts required to setup the insurance scheme.
3. Finally, we broadcasted a replacement transaction which is aimed at replacing the payment transaction with a signficantly higher fee, followed with a countermeasure funding into the insurance contract.

5.4 Challenges

Exchanging messages between clients in the internet requires time. Assuming an average round-trip time of 100ms for each exchange, the CEP protocol requires a cumulative total of 300ms to set up the complete insurance scheme, excluding complications associated with transaction propogation delays in the Bitcoin network. During the exchange process, the insuree is unprotected as long as the insure contract has not been broadcasted to the Bitcoin network, therefore an insurer could easily backout of the entire insurance setup process if a doublespend attack was discovered during the setup phase.

Nodes in the network are not homogeneous in reality as different nodes may implement varying transaction acceptance policies and their hashing power is unevenly distributed between every peer in the network. Therefore, without an effective probe to measure the distribution of hashing power, it will be difficult to model the preference of accepted transactions, which affects the ability to calculate an accurate premium value to charge to insurees.

5.5 Future Improvements

As it stands now, there is little incentive to continue for a full implementation of the insurance scheme as there is no incentive to do so until such time miners in Bitcoin Mainnet elect to mint transactions which pays the highest fees, which we anticipate will happen in the long run as mining rewards gradually diminish and main source of income for miners is simply transaction fees.

Due to time constraints, we were not able to fully implement Bitwatch according to our initial design specifications. Without Bitwatch, it will not be possible to deploy a full insurance service as it is essential to have a custom built Bitcoin node which tracks all current UTXO sets at all times. In our test runs, we were only able to perform doublespends in a controlled setting where each doublespend was immediately addresses with a corresponding countermeasure sent immediately to the network. In reality, it will take time for a monitoring node to notice a doublespend attempt and this duration should be studied in future research.

Our current proposed insurance scheme requires that an absolute lock time be specified well ahead of time, therefore it is necessary to set the locktime to an arbitrary time into the future. This is ineffective as it is difficult to predict when a preceding contract will be committed into the blockchain. This can potentially be addressed in the future once the the BIP112 soft-fork [5] is complete. This BIP formally introduces a new opcode, `OP_CHECKSEQUENCEVERIFY` which enables relative time locks to be set. This is particularly useful to ensure that a claim contract is well-embedded in the blockchain for a specific block depth before a reimbursement transaction is broadcasted to claim the outputs from the claim contract. The advantage of using this as oppose to `OP_CHECKLOCKTIMEVERIFY` is that the claim contract now has to be strictly N blocks apart from the reimbursement contract, which guarantees that an insurer is well-guarded against potential forks in the blockchain.

More study about the mining selection policy in the form of a formal survey with mining pools in the network will need to be conducted to be able to create a model that can more accurately model the potential acceptance of RBF-enabled replacement transactions. As it is right now, it simply a dice roll as to whether a RBF-enabled replacement transaction would be included in the Blockchain as it really depends on which mining pool was successful at generating a block to include said transaction. While our insurance scheme covers a recipient in both cases, due to the uncertainty of transaction acceptance, it is unlikely that there will be any insurance providers that will be willing to take up these hidden risks.

Chapter 6

Conclusion

In this thesis, we have developed a novel solution to support safe acceptance of zero confirmation transaction with deterministic guarantees which caters for any possible outcome of a doublespend attack against an insured transaction. We then proceeded to analyse the feasibility of the proposed scheme by measuring the rate of acceptance of RBF transactions by the network by deploying probe RBF transactions to identify the proportion of nodes that have accepted them. Furthermore, we also executed several doublespend RBF attacks on both Mainnet and Testnet.

Mining policy is inherently a private decision to be decided by miners in the Bitcoin network and based on our findings it is apparent that the majority of miners in Mainnet (i.e. F2Pool, Antpool and BTCC) implements the old First-Seen-Safe (FSS) acceptance policy which disregards higher paying replacement transactions. We anticipate that this situation will change in the long run as miners move to maximise on their transaction fee receipts and adopt mempool replacement policies which enables the preferential mining of higher paying transactions.

References

[1] Gavin Andressen. Bip 16: Pay to script hash. https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki, 2012. Accessed: 2016-05-14. 5, 12

[2] Adam Back, G Maxwell, M Corallo, Mark Friedenbach, and L Dashjr. Enabling blockchain innovations with pegged sidechains. Technical report, 2014. 5

[3] Adam Back et al. Hashcash-a denial of service counter-measure, 2002. 5

[4] Tobias Bamert, Christian Decker, Lennart Elsen, Roger Wattenhofer, and Samuel Welten. Have a snack, pay with bitcoins. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–5. IEEE, 2013. 5

[5] Eric Lombrozo BtcDrak, Mark Friedenbach. Bip 112: Op_checksequenceverify. https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki, 2015. Accessed: 2016-05-14. 53

[6] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *Proceedings on Advances in cryptology*, pages 319–327. Springer-Verlag New York, Inc., 1990. 5

[7] Peter Todd David A. Harding. Bip 125: Opt-in full replace-by-fee signalling. https://github.com/bitcoin/bips/blob/master/bip-0125.mediawiki, 2015. Accessed: 2016-05-14. 3, 5

[8] Christian Decker. Bip 140: Normalized txid. https://github.com/bitcoin/bips/blob/master/bip-0140.mediawiki, 2015. Accessed: 2016-05-14. 5

[9] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE, 2013. 5

[10] Christian Decker and Roger Wattenhofer. Bitcoin transaction malleability and mtgox. In *Computer Security-ESORICS 2014*, pages 313–326. Springer, 2014. 5

[11] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Stabilization, Safety, and Security of Distributed Systems*, pages 3–18. Springer, 2015. 5

[12] Pieter Wuille Eric Lombrozo, Johnson Lau. Bip 141: Segregated witnesses. https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki, 2015. Accessed: 2016-05-14. 5, 17

[13] Matthias Lei. Exploiting bitcoin's topology for double-spend attacks. *Bachelor's Thesis*, 2015. 5, 16

[14] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. 5, 6

[15] Satoshi Nakamoto. Original bitcoin transaction replacement source code. https://github.com/trottier/original-bitcoin/blob/master/src/main.cpp#L434, 2013. Accessed: 2016-05-14. 13

[16] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. Technical report, Technical Report (draft). https://lightning. network, 2015. 5

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

[18] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security*, pages 507–527. Springer, 2015. 5