Paying your Internet, One Byte at a Time

Sandra Siby
Supervisor: Christian Decker

December 16, 2013

Abstract

With the growth in Internet capable devices, users expect to be able to access the Internet at all times. Wireless access points are abundant but users cannot access them because of restrictions placed by the owners of the access points. One of the main reasons for this is compensation; owners do not want to give access for free when they are paying for it. This could be resolved by owners accepting payments for opening their access points. However, current methods of payment involve high transaction fees. Bitcoin, a decentralized digital currency, looks promising since it has lower transaction fees but even these fees might be too significant when the transaction amount is small. A micropayment channel primitive can be built on top of Bitcoin, that allows users to make very small payments on the go without incurring high transaction costs. This thesis builds a proof-of-concept that allows an access point to provide Internet access to untrusted users in exchange for payment in bitcoins, using a micropayment channel.

Contents

Chapter 1

Introduction

Over the years, there has been a tremendous growth in the number of Internet capable devices, especially smartphones. Users now expect to be able to access the Internet at all times. However, when in a foreign country, users often do not have mobile Internet coverage. At the same time, they are surrounded by wireless access points but are unable to use them because the owners restrict access to untrusted users. Access is restricted for two reasons - liability and compensation. Owners do not want to be held liable for the actions performed by someone else while using their access point. This issue could be resolved by VPN and other security mechanisms. Compensation is a more complicated problem. Owners do not want to open up their access points and provide free service when they are paying for the Internet access. This could be resolved by the owner accepting payments for opening the access point. A convenient method is needed for a user to make payments to the owner of an access point on the go and receive Internet access in return. However, current methods of payment, such as credit cards, require a long process of agreement with the access point and high transaction fees. This is undesirable when the transaction amount is small.

Bitcoin is a decentralized digital currency that has been gaining popularity over the last few years. Its relatively low transaction fees make it an attractive option for small payments. However, as small as the fees are, the overhead might still be too high as compared to the transaction. This issue can be solved by building a *micropayment channel* primitive on top of Bitcoin. A micropayment channel allows users to securely make very small payments on the go, without incurring high transaction costs. The channel also ensures that the party making payments does not lose money in the event of the party providing the service going down. The micropayment channels' properties make it a viable alternative to current payment methods for solving the compensation problem.

This thesis builds a proof-of-concept that allows an access point to provide Internet access to untrusted users in exchange for payment in bitcoins, using a micropayment channel. The rest of this chapter gives an overview of the Bitcoin protocol and the micropayment channel and discusses related work. Chapter 2 provides details on the implementation of the proof-of-concept. Chapter 3 discusses the performance of the micropayment channel. Finally, in Chapter 4, possibilities for improvement of the proof-of-concept are discussed.

1.1 Bitcoin Overview

Bitcoin is a decentralized digital currency system based on peer-to-peer technology. It was first introduced in 2008 in a paper by Satoshi [1] and the Bitcoin network became operational in early 2009. Unlike traditional currencies, all functions, such as creation of new currency units called *bitcoins* and verification of transactions, are performed collectively by the network instead of a central authority. Bitcoin's low transaction costs and independence from a central entity make it an attractive alternative to current popular payment methods.

Bitcoin uses public-key cryptography to authenticate transaction information. A user that wants to perform transactions using bitcoins requires a file called a *wallet*. A wallet contains a set of keypairs (public- and private-keys). The public-key is disclosed to others and is used to derive an address to which bitcoins can be sent. The private-key is kept secret since it authorizes the owner to spend bitcoins. A wallet also contains information about the transactions that were carried out using the keys and the current balance.

A transaction represents a transfer of the ownership of bitcoins from one address to another. Assuming a user has some bitcoins and wants to transfer them to another user, she creates a transaction record, signs this record with her private key and broadcasts it to the network. A transaction is implemented as a set of *inputs* and *outputs*. An output consists of a value denominated in bitcoins and claiming conditions that have to be met in order to spend the associated value. Inputs are references to outputs of previous transactions of the sending party and are used as a proof of ownership, i.e., the sender does indeed own the bitcoins that she will be spending in this transaction. The sum of the amounts claimed by the transaction will be greater than the sum of the amounts created by it. The difference between the total input value and the total output value is a *transaction fee*.

While a transaction proves the sender's ownership of bitcoins, it does not prevent her from using the same bitcoins in another transaction, a problem called *double-spending*. In order to ensure that the sender spends the bitcoins only once, the transaction record is added to a public ledger known as a *blockchain*. Should two distinct transactions attempt to spend the same bitcoins, only one is added to the blockchain. The blockchain's name stems from the fact that it is a chain of entities called blocks. Each block contains a set of new transactions not present in any previous blocks.

A new block of transactions is added to the block chain by nodes called miners. When a transaction is broadcast to the network, it is received by miners. Miners collect unconfirmed transactions and attempt to create a block. A Merkle tree of the transactions is created, its root is calculated and added to the block header. A hash of the last block in the current blockchain is also added to the block header, along with a timestamp and a nonce. Including the hash of the previous block effectively chains the blocks together since in order to be included, the hash of the previous block, and hence the block, must have been known to the miner. The chain provides a chronological order on the blocks and the transactions therein. Finally, since several miners are working on creating a block but only one block can be added to the block chain, miners compete to get their block accepted into the chain by searching for a nonce. The nonce should cause the block hash, interpreted as integer, to be below a predetermined threshold. Since obtaining the input of a hash function given its result is computationally infeasible, a miner has no choice but to keep on trying different nonces and hashing the block until the target value is obtained. This nonce is known as proof-of-work, as it implies that the miner put effort into finding it, but is easy to verify by others.

The first miner that finds the value broadcasts the new block to the network. The other peers verify the value and the block is accepted and added to the block chain. Transactions in this block are now considered to be valid or confirmed.

Since mining is a resource intensive task, it has to be incentivized. Hence, every time a miner is successful in creating a block that is accepted into the block chain, she gets a reward of newly minted bitcoins. This is how new coins are introduced into the network. The number of new coins per block started off at 50 initially but halves every 210,000 blocks. This regulates the supply of new coins. In addition to the block reward, miners get the transaction fees of transactions they confirm in the block. The total number of bitcoins is capped at 21 million. Once this ceiling is reached, miners will earn coins only via transaction fees.

1.2 Micropayment Channel

A micropayment is a transaction involving a very small amount of money. Traditional modes, such as credit cards or bank transfers, are not efficient when it comes to such transactions because the transaction fee becomes comparable to the amount being transferred.

Bitcoin, with its lack of a third party and relatively low transaction fees, has high potential to be used in micropayments. However, there are still a few issues that need to be resolved. First, a transaction is not free and for really small transactions, the overhead might be higher than what the transaction was worth. Second, if several transactions are sent in very fast succession, they might be down-prioritized or not relayed due to anti-flooding algorithms in the network [2]. For the use-case of the access point, we want to pay as we go, i.e., send multiple small payments during our session every time we want to prolong the session. In order to handle the issues caused by micropayments, we need to implement a micropayment channel. The micropayment channel allows a party to make repeated micropayments without high transaction fees or adverse impact on the network.

Consider a client that wants to use the services provided by a server. In exchange, it will pay the server in bitcoins. However, the client and the server are untrusted parties to each other. The micropayment protocol allows the client to make repeated payments to the server without losing money in the case of the server not providing the service. The client first creates a multi-signature transaction. A multi-signature transaction can be considered to be a shared account between the client and the server and it requires both their signatures to authorize outgoing transactions. This transaction locks in a certain amount of money. However, before signing this transaction and allowing it to be broadcast, the client creates a transaction, called a *refund transaction*, and gives it to the server to sign. This transaction refunds the entire amount to the client but is time-locked. This means that the value would be refunded only after a certain period of time has elapsed. This refund transaction is to protect the client from losing money if the server goes down or does not provide the service it is supposed to. Once the server has signed the refund transaction, the client sends the multi-signature transaction with its signature. The server signs it and broadcasts it to the network. The channel can now be considered to be open between the two parties.

The client then creates a third transaction, called an *increment transaction*, which has two outputs, one to its own address and one to the server's. Initially, the amount is completely allocated to the client's address, like the refund transaction but without a timelock. The client signs and sends this to the server. Whenever the client wishes to make a payment to the server,

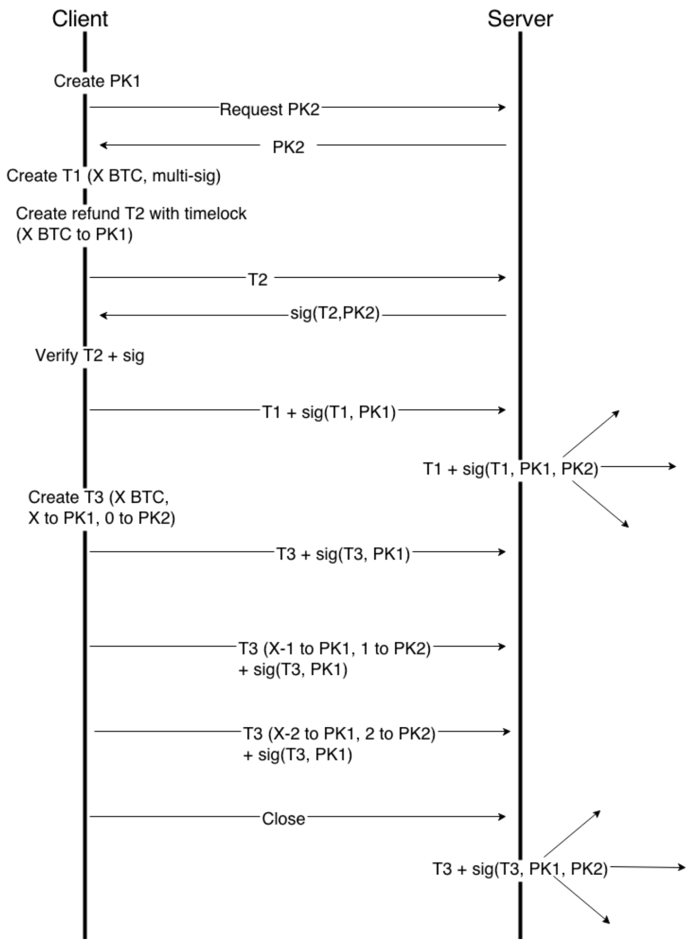
Client
Server
Create PK1
Request PK2
PK2
Create T1 (X BTC, multi-sig)
Create refund T2 with timelock (X BTC to PK1)
T2
sig(T2,PK2)
Verify T2 + sig
T1 + sig(T1, PK1)
T1 + sig(T1, PK1, PK2)
Create T3 (X BTC, X to PK1, 0 to PK2)
T3 + sig(T3, PK1)
T3 (X-1 to PK1, 1 to PK2) + sig(T3, PK1)
T3 (X-2 to PK1, 2 to PK2) + sig(T3, PK1)
Close
T3 + sig(T3, PK1, PK2)

Figure 1.1: Micropayment protocol. T1: opening of shared account, T2: refund transaction, T3: incremental payment.

it adjusts and signs its copy of the transaction by reducing the amount allocated to it and increasing the server's output. It lets the server know about these changes. Finally, when the client is done, it notifies the server. The server signs the final copy of the transaction, broadcasts it to the network and closes the channel. The mechanism prevents misbehavior from both the client and the server. If the server leaves before the client closes the channel, the client gets the money refunded on expiration of the timelock. The client cannot try to release the refund transaction because it is timelocked and the server can release the valid incremental transaction. The client can also not release an older version of the incremented transaction because it is missing the server's signature. The micropayment protocol is shown in Figure 1.1.

1.3 Related Work

The Bitcoin protocol was first described by Satoshi[1]. Since then, there have been a few papers covering different aspects of Bitcoin. Karame et al.[3] were the first to analyze the probability of successfully performing double spend attacks. Double-spend attacks were further investigated by Bamert et al. [4], who provided a few strategies to reduce the probabilities of such attacks. Their work helps us in analyzing failure scenarios in Chapter 3. Decker et al.[5] researched the dissemination of information in the Bitcoin network. Sompolinsky et al.[6] built upon their findings and analyzed the effect of block size on network delay. The conclusion was that that having too many transactions, and hence larger blocks, causes adverse network impact. This validates our decision to implement the micropayment channel and avoid flooding the network with transactions. Reid et al.[7] discussed the anonymity in the Bitcoin network and how it could be compromised in certain scenarios. We do not discuss anonymity in our thesis but their findings could be useful in possible future work regarding identification of clients that return to use the access point.

Chapter 2

Implementation

As proof-of-concept, we implemented a wireless access point using micropayments. Our goal was to make the configuration of the access point simple and easy for an end-user. Hence, we used a Raspberry Pi as a Wi-Fi access point. The Raspberry Pi is a low cost, compact, single-board computer that can be used for a variety of applications. Its simplicity and low cost make it a good candidate when compared to vendor access points which can be complicated to set up for the end-user. Its ability to run standard operating systems such as Linux makes it easy for the end-user to modify.

A client that wants to connect to the access point can be considered to be in one of two states - authorized or unauthorized. Authorized means that the client has paid and is allowed to access the Internet. New clients are initially in the unauthorized state. We identify the following steps in the communication between a client and the access point:

- Discovery - The access point announces its availability to clients. Clients connect to the wireless network provided by the access point but do not receive access to the Internet. Instead, they are redirected to a captive portal.

- Negotiation - The captive portal serves as a point of contact between the client and the access point. The access point advertises its services and pricing model. The client can pick its preferred service and confirm that it will pay for the service in bitcoins.

- Channel Establishment - Once the client and access point have negotiated an agreement, they set up a micropayment channel for the client to make payments on the go. The access point provides Internet access while the channel is established.

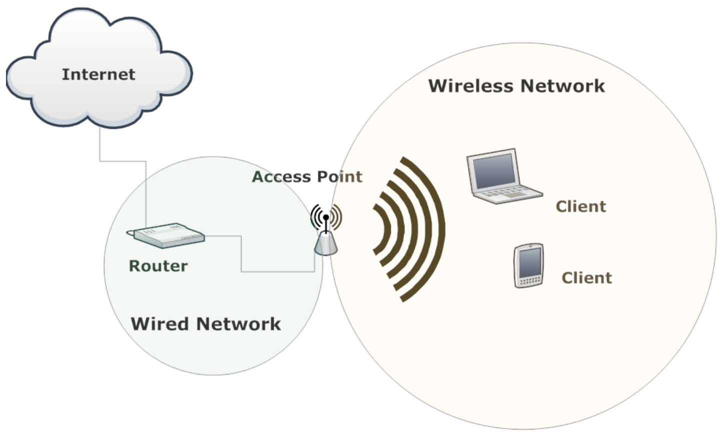

Figure 2.1: Access point.

- Termination - If the client runs out of funds, loses contact or decides to actively stop using the access point's service, the micropayment channel is closed. Upon termination, the access point stops providing access to the client. In case of the access point going down before the agreed time period, the client is refunded the money.

The communication steps and their implementation are described in detail in the following sections.

2.1 Discovery

The Raspberry Pi is configured as an access point that offers Wi-Fi connectivity to clients and then routes the traffic to an Ethernet cable. The Raspberry Pi is equipped with a USB Wi-Fi dongle and an Ethernet cable connected to a router. It acts as a gateway router between the wireless and wired networks. The setup is shown in Figure 2.1. Upon running the host access point server, the new wireless network provided by the access point appears in the list of networks as shown in Figure 2.2. Since the access point is supposed to be used by untrusted clients, it is not password protected and does not require the client to authenticate. The lack of encryption might appear to pose a risk of eavesdropping or man-in-the-middle attacks. However, as mentioned earlier, we require VPN for liability. Hence, we circumvent the encryption problem by requiring the users to adopt such security mechanisms.

Once the client is connected to the wireless network, the access point assigns it an IP address. We implemented this by installing a DHCP server

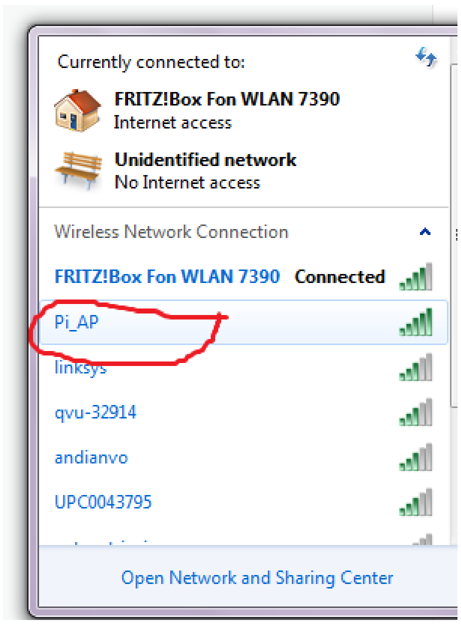

Figure 2.2: Discovery.

on the Raspberry Pi. DHCP (Dynamic Host Configuration Protocol) is a protocol used to assign IP addresses and configuration information to devices so that they can communicate on the network. The DHCP server accepts clients' requests to connect to it and replies to them.

We then implemented a captive portal mechanism. Clients are tracked using their MAC addresses. We modified the firewall rules on the access point such that unauthorized clients are not immediately given Internet access but first redirected to an information page for negotiation with the access point.

2.2 Negotiation

The information page describes the services provided by the access point and their prices. The access point can choose to have a time-based service model, where the client pays based on time period of usage. Additionally, it can have a data-based service model where the client pays a higher price to get access to a higher bandwidth. The information page describes these pre-defined service profiles and the client can pick one based on its needs. The information page also contains a section outlining the terms of the agreement between the client and the access point. Once the client picks its service profile and accepts the agreement that it will pay for the services of the access point, the micropayment channel is established for payment and the server modifies its firewall rules to allow the client to access the Internet.

We implemented the captive portal mechanism by intercepting all the packets of the client and responding to the HTTP packets by redirection to the information page using a CherryPy redirect server. We implemented a time-based service model for this project.

2.3 Channel Establishment

Once the client accepts the agreement, the micropayment channel establishment process begins. A Bitcoin client application is started on the client. The access point has a Bitcoin server application running, that listens for new connection requests and acts as a local proxy to the Bitcoin network for unauthorized devices. Details for the micropayment server are on the information page, encoded in URI. The channel establishment process starts when the URI is clicked. The client connects to the server and starts the micropayment channel establishment process. The client and the server create a shared account by means of a multi-signature address.The client first creates a transaction that locks in a certain amount of money in the shared account. However, before broadcasting this transaction, the client creates a refund transaction. This transaction is linked to the output multi-signature transaction and refunds the entire amount to the client. However, it is time-locked, i.e., it ma only be confirmed after a specified time period has elapsed and, more importantly, it may be replaced. The refund transaction protects the client from losing money in case of the server not providing the service it advertised. The client sends the refund transaction to the server for it to sign. Once the refund transaction has been signed, the client signs its multi-signature transaction and sends it to the server. The server signs it and broadcasts it to the network. The channel is now considered to be open and the client is considered to be authorized.

When the channel is opened, the server has to modify the firewall rules to allow access to the client. It takes the client's MAC address and adds it to the firewall rules so that the client no longer gets redirected to the portal page. It also removes previous connection tracking information about the client. If this tracking information is not removed, the client might still get redirected to the portal. Once the server has modified the rules, the client is able to access the Internet.

The client then creates a transaction that is connected to the output multi-signature transaction it created in the beginning. This transaction has two outputs, one to the client's address and the other one to the server's address. Initially, the entire amount is allocated to the client. It signs this

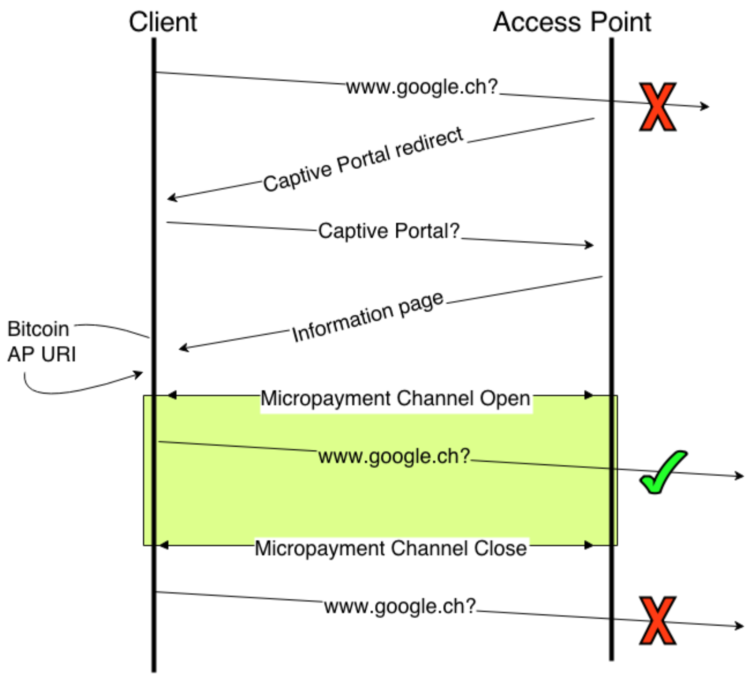

Figure 2.3: Client-AP communication.

and sends it to the server. Every time the client wants to make a payment, it adjusts its copy of this transaction - it reduces the amount allocated to itself and increases the amount allocated to the server. When the client informs the server about these changes, the server adjusts its copy. This process continues, with the server's amount increasing and the client's amount decreasing until the micropayment channel is closed.

We implemented the micropayment channel with the help of *bitcoinj*[1], a Java implementation of the Bitcoin protocol.

2.4 Termination

If the client runs out of funds or decides to actively stop using the services of the access point, it closes the channel. When the server detects a channel closure, it signs the final copy of the payment transaction and broadcasts this to the network. It also modifies the firewall rules to no longer give access to the client. It removes the client's connection track information so that the client is once again redirected to the portal page.

If the server disappears before the channel is closed by the client, the

[1]http://code.google.com/p/bitcoinj/

refund transaction comes into play. The client is refunded the amount 24 hours later.

If the client has not closed the channel before the channel has expired, there is a risk of the client getting the refund even though the server has been providing Internet access. To avoid this, the server has to close the channel before the expiry time and a new channel has to be created. The communication between the client and the access point is outlined in Figure 2.3.

2.5 Packaging

One of the goals of the project is to ensure ease of use for both the access point owner and the client. Our choice of the Raspberry Pi ensures a low cost, easy to configure option for the access point owner. Not only this, the software for the Bitcoin server and the captive portal mechanism, to be run on the access point, do not require any intervention by the owner, other than to set the payout address. On the client side, we developed an Android version of the micropayment channel client. Hence, it becomes very convenient for users to get the app from the App Store and install on their mobile devices.

Chapter 3

Evaluation

In order to evaluate the proof-of-concept, we subjected it to various tests. In the following sections, we analyze the performance of the proof-of-concept in terms of timing, security and usability. We also describe possible failure scenarios and how the micropayment channel handles them.

3.1 Timing Analysis

We ran tests to evaluate the performance of the micropayment channel. Times taken to perform the following activities were measured:

- Open a channel.
- Send the refund transaction to the server and receive the signed refund transaction back.
- Close the channel.

Our experimental setup consisted of the Raspberry Pi (access point) running the Bitcoin server code and one laptop (client) running the Bitcoin client code. The client was made to initiate a micropayment channel with the server, send a few micropayments and close the channel. Timings for each step of the micropayment channel process were recorded in log files. Values were obtained for 100 runs. Figure 3.1 shows the distribution of time periods between a client initiating a connection and the channel being established. During this time period, the client signs the multi-signature transaction, sends a refund transaction, receives it back and finally sends the multi-signature transaction to the server. Figure 3.2 shows the distribution of time periods between a client sending a refund transaction for the server

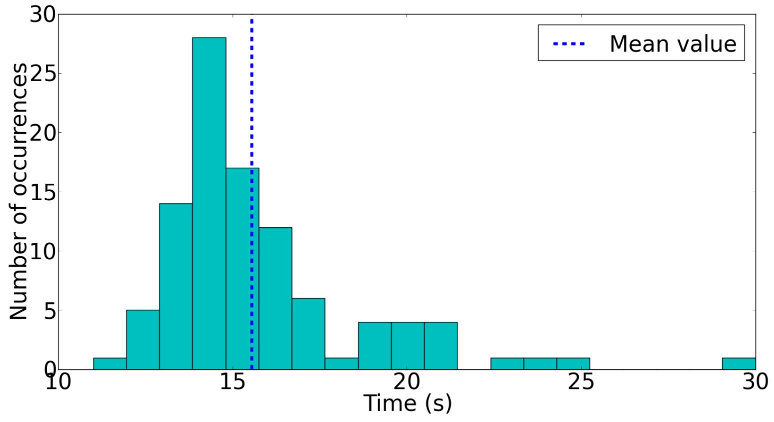

Figure 3.1: Channel open time distribution.

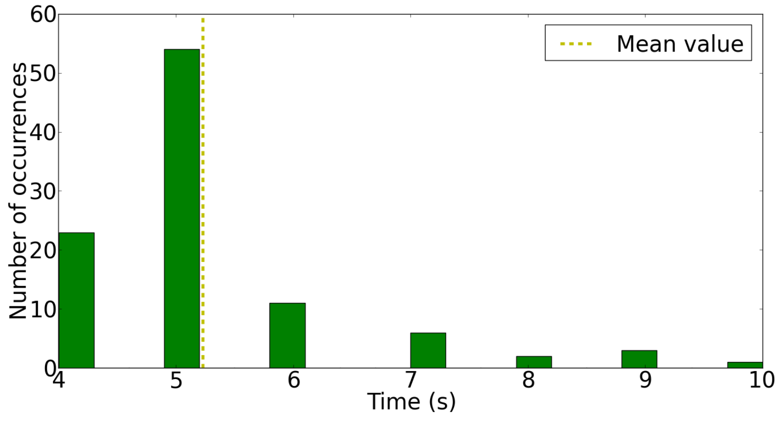

Figure 3.2: Refund transaction time distribution.

to sign and the client receiving it back. Figure 3.3 shows the distribution of time periods between the server receiving a close message from the client and the server closing the channel. During this time period, the server changes the firewall rules, signs the final version of the multi-signature transaction and broadcasts it.

We observed that channel establishment and closure were relatively fast processes, taking only a few seconds each. For channel closure, it was the broadcast of the final transaction that constituted majority of the time taken. Time taken by the server to change firewall rules on receipt of payment was in the order of milliseconds. We also observed that if the client closed the channel but opened it again before the channel expiry time, the previously created channel was resumed rather than a new one being created. This

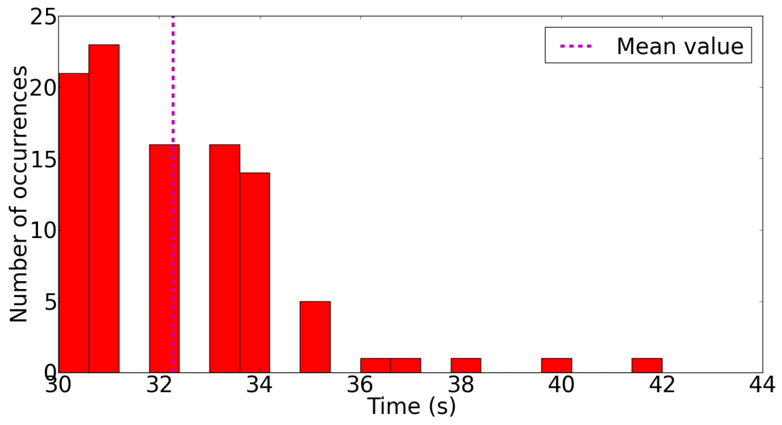

Figure 3.3: Channel close time distribution.

means that clients do not have to spend time recreating a channel but can instead resume the previously created one.

The Bitcoin server supports an arbitrary number of devices. We limited our evaluation to 2 clients and found that the server handles the isolation between the client states correctly. Timing results were comparable to the case with one client.

3.2 Failure Scenario Analysis

There are 2 failure scenarios to be considered - one where the server does not behave as expected and one where the client tries to perform a double-spend attack. The scenarios and their impact are discussed below:

- The server goes down before the channel is closed by the channel and does not offer the service the client paid for. This is resolved by the refund transaction created by the client and signed by both parties during the micropayment channel establishment process. The refund transaction is time locked and will be valid if the server goes down before channel closure. Thus, the client is ensured to not lose money. For our evaluation, we analyzed the log files and confirmed that the micropayments are started by the client only after the server has signed the refund transaction. We also simulated the condition where the server is stopped before the channel is closed. On checking the blockchain, we noticed the refund transaction confirmed when its time lock expired, i.e., 24 hours after channel establishment.

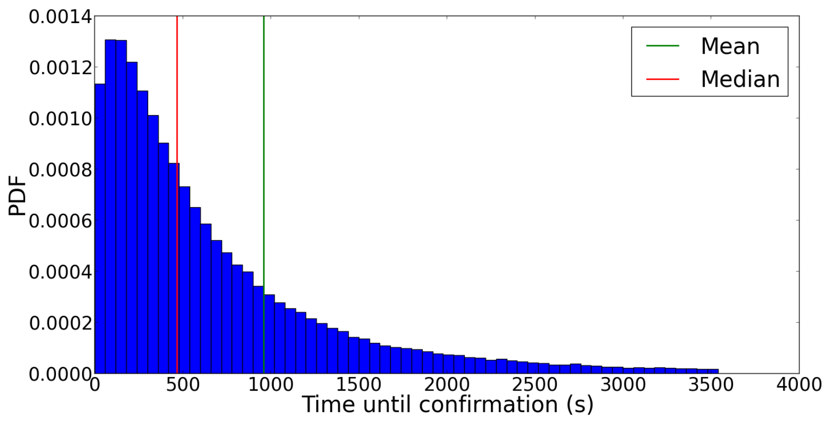

Figure 3.4: Confirmation time distribution.

- There is a probability of the client trying to spend the coins it already locked in the transaction with the server. This is avoided by the server first broadcasting the multi-signature transaction to the network. Not only this, the server can broadcast later transactions which have the client's signature and a later timestamp proving the correct sequence of events. Note that the micropayment channel establishment does not wait for the initial multi-signature transaction to be confirmed before proceeding with payments since confirmation takes 10 minutes on average and the client should not have to wait that long to use the service. This leaves open a small window of opportunity for the client to attempt a double-spending attack. However, the probability of this attack succeeding is very low [4]. In the rare occasion lost by the server will be too small to be considered significant. The average confirmation time is shown in Figure 3.4. The mean time is about 16 minutes, which is relatively high. However, this high value is due to a few transactions that took a large amount of time to be confirmed. A more relevant parameter would be the median, which is about 7.8 minutes.

3.3 Security Analysis

While the trend of providing hotspots to untrusted users is on the rise, the owner of the access point faces liability concerns. The question arises as to whether the owner should be held liable for illegal activity performed by a user of the access point and whether it is the owner's responsibility to ensure that user's information is not stolen. Certain countries such as

Germany have laws holding the owner liable for all actions performed on the public Wi-Fi network she provides, even if they were carried out by a third party. Liability and security concerns could potentially dissuade access point owners from opening up their networks to untrusted clients. Our goal is to enable owners to provide a paid hotspot service and we suggest that a few precautionary measures be taken to reduce liability issues.

The owner could have a Terms of Service section on the information page where all activities that are allowed or prohibited on the network are described. Clients would have to read and agree to these terms before they can use the service. In addition to this, as mentioned in our Implementation chapter, clients are tracked using their MAC addresses. Having logs of client activities as proof can be useful to the owner in case of legal issues. Not only this, mentioning the fact that their address is being tracked on the information page might discourage some clients from violating the terms. Owners could even filter certain websites to prevent access. With regards to the security of the client, it is advised that clients use VPN to avoid eavesdropping or man-in-the-middle attacks and protect their information.

Chapter 4

Conclusion

We implemented a proof-of-concept that allows an access point to provide Internet access to untrusted users in exchange for payment in bitcoins. Our proof-of-concept is low cost and easy to configure for end-users. We also successfully implemented a micropayment channel mechanism between the client and the access point and evaluated its timing and security performance.

There are a few potential avenues for improvement. In this project, we implemented a time-based service, where users pay based on the time they use the service for. We can also have a data-based model where users pay according to the bandwidth they require. Another point is regarding the refund transaction for the micropayment channel. Currently, the time lock for the refund transaction has been set to 24 hours in the bitcoinj protocol. This means that the client would receive money (in case of server failure) only after 24 hours. While we have not altered this time period in our proof-of-concept, it is trivial to do so if the need arises.

Bibliography

[1] S. Nakamoto: *Bitcoin: A peer-to-peer electronic cash system*

[2] *Contracts.* Retrieved 29 November 2013 from https://en.bitcoin.it/wiki/Contracts

[3] G. Karame, E. Androulaki, and S. Capkun: *Two bitcoins at the price of one? Double-spending attacks on fast payments in bitcoin* in Proc. of Computer and Communication Security (2012)

[4] T.Bamert, C.Decker, L.Elsen, R.Wattenhofer, S.Welten: *Have a Snack, Pay with Bitcoins.* 13th IEEE International Conference on Peer-to-Peer Computing (2013)

[5] C.Decker, R.Wattenhofer: *Information Propagation in the Bitcoin Network.* 13th IEEE International Conference on Peer-to-Peer Computing (2013)

[6] Y.Sompolinsky, A.Zohar: *Accelerating Bitcoin's Transaction Processing Fast Money Grows on Trees, Not Chains*

[7] F. Reid and M. Harrigan: *An analysis of anonymity in the bitcoin system* in SocialCom (2011)

[8] MB&M LLP: *Wi-Fi Hotspots and Liability Concerns.* Retrieved 15 December 2013 from http://www.mbm-law.net/newsletter-articles/wi-fi-hotspots-and-liability-concerns/1229/