Paying your Internet, one byte at a time

Sandra Siby
Supervisor: Christian Decker

December 9, 2013

Abstract

Acknowledgements

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 2

Background

2.1 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 coins ('bitcoins')and verification of transactions, are performed collectively by the network instead of a central authority. Bitcoin's low transaction costs and lack of reliance on 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 is required by the owner to send bitcoins. A wallet also contains information about the transactions that were carried out using the keys and the current balance. There are currently different implementations of the Bitcoin wallet.

A transaction represents a transfer 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*. Outputs provide instructions on sending bitcoin - an output consists of a value and the address to which it should be sent. A transaction can have multiple outputs. Inputs are reference outputs of previous transactions to the sender 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 difference between the total input value and the total

output value is a *transaction fee*.

While a transaction verifies 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.

For a transaction to be considered valid, it has to be committed to a public ledger that contains all previous transactions - the blockchain. The blockchain's name stems from the fact that it is a chain of entities called blocks, where each block contains a set of new transactions not present in the previous block.

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. These miners collect several transactions and try to create a block by performing the following operations: 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. This hash helps in proving the chronology of the block chain. 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 value. This value, when added to the block and hashed, should give a predetermined target value (a large number starting with several zeroes). 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 till the target value is obtained. This value is known as proof-of-work - it implies that the miner put effort into finding the value 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, he/she gets a reward of brand new 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 this, miners get a transaction fee. The total number of bitcoins is capped at 21 million. Once this ceiling is reached, miners will earn coins only via transaction fees.

(Picture???? Maybeee TBD)

2.2 Micropayment Channel

A micropayment is a transaction involving a very small sum 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 very fast, they might be down-prioritised or not relayed due to anti-flooding algorithms in the network. For our 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 these 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 network impact.

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. 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 this transaction, the client sends the mutlisignature 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 which has two outputs, one to its own address and one to the server's. Initially, the amount is completely

allocated to C'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, it adjusts 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. The server adjusts its copy of this transaction to reflect the 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. If the server leaves before the client closes the channel, the client gets the money refunded on expiration of the timelock. The micropayment protocol is shown in Figure 2.1. (NOTE SS: IMPROVE PIC FONTSIZE + WORD ORDERING)

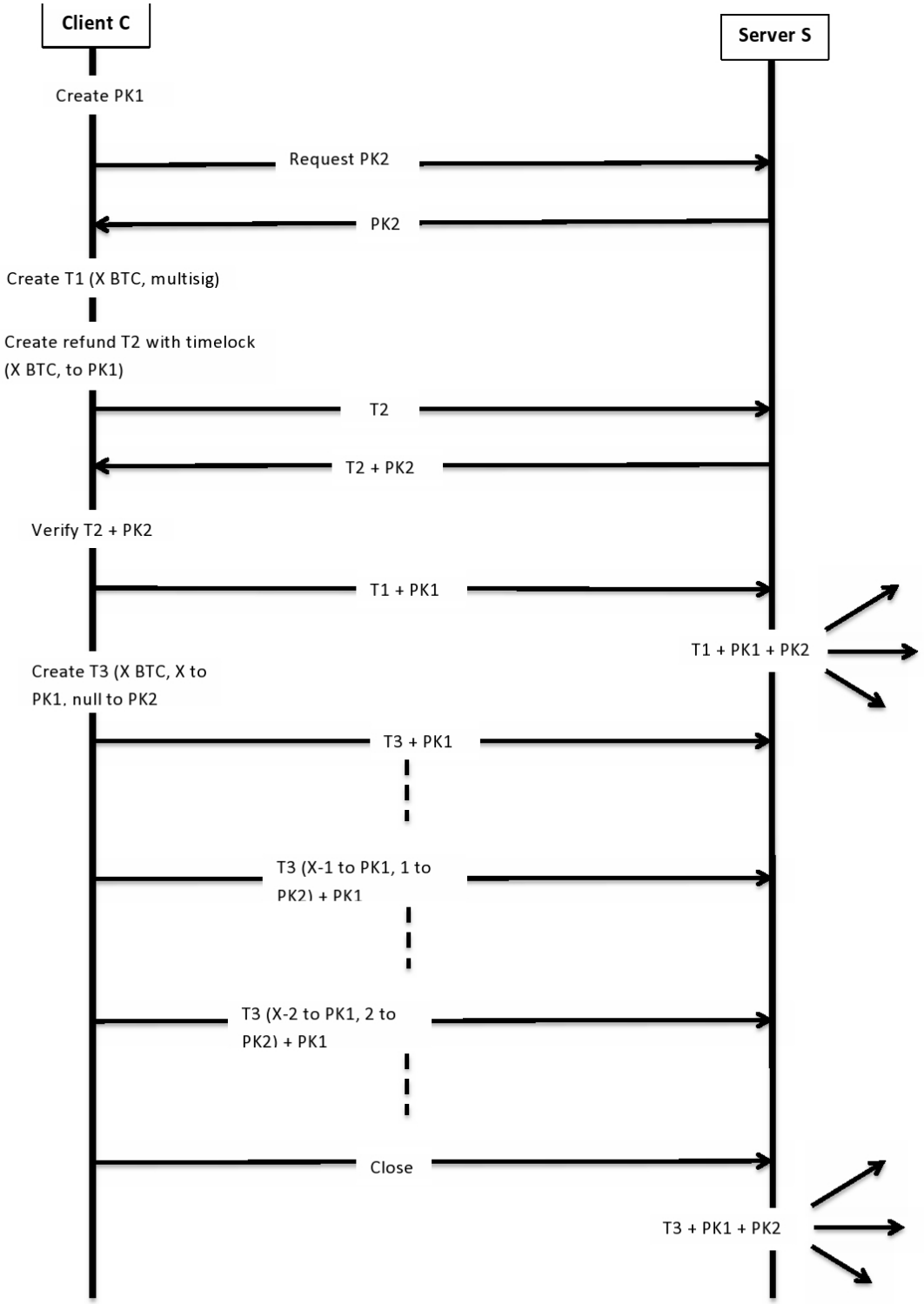

Figure 2.1: Micropayment protocol

Chapter 3

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.

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 when the channel is established.

- 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

Figure 3.1: Discovery

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 the following sections. (Note SS: Add Pic - Rpi, figure showing AP/client).

3.1 Discovery

We set up the Raspberry Pi as an access point that broadcasts a Wi-Fi service to clients and then redirects the traffic to an Ethernet cable. The Raspberry Pi is equipped with a US 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 X (Note SS: Take pic tomorrow). On running the host access point server, the new wireless network provided by the access point appeared in the list of networks as shown in Figure 3.1. Since the access point is supposed to be used by untrusted clients, it is not password protected. There is no encryption scheme which means that there is a risk of eavesdropping or man-in-the-middle attacks. However, our goal is to provide an open network to which anybody can connect. We would suggest that users take measures to ensure security such as using VPN.

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

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 ISC DHCP server accepts clients' requests to connect to it and replies to them.

We the implemented a captive portal mechanism: we modified the firewall rules on the access point such that new clients are not immediately given Internet access but first redirected to an information page for negotiation with the access point.

3.2 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 predefined 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.

3.3 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. The client connects to the server and starts the micropayment channel establishment process. The client first creates a transaction that locks in a certain amount of money. This transaction is a multi-signature transaction that requires both the client's and the server's signatures. The multi-signature transaction can be thought of as a shared account between the client and the server. However, before broadcasting this transaction, the client creates a refund transaction. This transaction is linked to the out-

put multi-signature transaction and refunds the entire amount to the client. However, it is time-locked, i.e., it refunds the amount only when a specified time period has elapsed. 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 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.

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 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 it 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 till the micropayment channel is closed.

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

3.4 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 goes down before the channel is closed by the client, the 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. (Note to CD: Combine 3.3 and 3.4 to one section - micropayment channel? should I write about transaction confirmation? Also mention Android app?)

Chapter 4

Evaluation

Chapter 5

Conclusion

Bibliography

[1] Satoshi paper

[2] Decker paper

[3] Khan video

[4] Bitcoin website