Implementation of an Incentivised Mesh Network using Micropayments facilitated through Blockchain and Payments Channel Technology

Prepared By:
Elle Mouton
(MTNELL004)

Prepared for:
Mr Jarryd Son and **Assoc Prof. Amit Mishra**
Department of Electrical Engineering
University of Cape Town
South Africa

14 October 2019

Submitted to the Department of Electrical Engineering at the University of Cape Town in partial fulfillment of the academic requirements for the degree of Bachelor of Science in Electrical and Computer Engineering

Abstract

The inability to make fast and secure, device-to-device, micropayments has lead to a development stunt in the Internet of Things industry. IoT devices provide data or services in continuous streams and yet payments made to the devices are either pre-paid or post-paid due to the fact that making micropayment using trusted third parties such as banks is infeasible due to the high transaction fees and latency. Both these methods of payment are unideal as they allow opportunities for malicious parties to either claim payment without providing services (pre-paid) or use services without paying (post-paid).

With the introduction of blockchain technology, cryptocurrencies and payment channel technology, fast micropayments between devices can be possible. Using cryptographic techniques, the exchange of payment and service can also be made atomic and thus more secure.

The aim of this project is to identify an IoT use case in which the introduction of device-to-device micropayments would be beneficial and to implement and evaluate such a system. The chosen use case for this project is an incentivised mesh-network. In such a system, users of the network can connect to the internet indirectly by having their packets routed by other devices in the network and paying these devices for doing so. Users are incentivised to leave their devices connected to the network as relay nodes because they will earn value (in Bitcoin) for contributing to the successful delivery of packets.

In this report, a description of the design and implementation of a basic incentivised mesh network using Bitcoin and payment channels is given. After evaluating the basic implementation, it was found that the computational overhead involved in signing Bitcoin transactions made the solution infeasible for the purpose of the mesh-network due to the low bandwidth achieved by the system. Adjustments were then made to the basic implementation so that a single payment process could be cryptographically connected to the successful delivery of multiple packets. The adjusted implementation performed significantly better than the basic implementation and resulted in an implementation that meets the minimum bandwidth requirements for a variety of popular internet applications. Estimates were made of the performance of a larger incentivised mesh network where computers with faster processors are used and from these estimates it is concluded that the proposed solution is in fact viable.

Acknowledgements

I would like to give thanks to the following people without whom this project and the last four years would not have been possible:

To my Mom for being by my side every step of the way for the last 22 years.

To Craig for introducing me to the wonderfully logical world of Bitcoin and for just being awesome.

To my Dad, Ronelle, my grand parents and the rest of the family for all the continuous love and support.

To Kea for being my go-to person since day one.

To Walter, Loll, Tamir, Tristan, Anna and Emma for teaching me how to take breaks and have fun.

To Dillon and Stefan for managing to tolerate me for the last four years.

To my supervisors, Prof Amit Mishra and Mr Jarryd Son for giving me free reign to explore.

To Ms Dianne Leonard for pushing me to start trying harder in high school.

To Mrs Jane Behne and Mr Graham Reggiori for teaching me to love and obsess over maths and physics.

To Rene Pickhardt for taking the time to read through some of my paper and provide useful feedback.

And finally, to Satoshi Nakamoto for creating Bitcoin.

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Signature:.....
E. Mouton

Date: 14 October 2019

Contents

.

List of Figures

Listings

List of Tables

Chapter 1

Introduction

1.1 Background to this study

The Internet of Things (IoT) industry has grown exponentially over the last few years resulting in a more seamlessly integrated digital world. One major development stunt in the IoT industry has been caused by the inability to integrate micropayment methods between devices in a reliable, secure and cost-effective way. A possible solution to this problem could be developed through the use of new technologies such as distributed blockchain and payment channels [1]. With these tools, IoT devices could easily make fast micropayments between each other which would allow for a range of applications such as pay-as-you-go services for the use of data, sensors, electricity and more.

An example of a use case in which IoT devices would benefit from micropayments is a mesh network. Mesh networks, in theory, offer an effective way of providing reliable internet over an area where perhaps only a few people have a reliable internet connection. With a mesh network, devices connect to each other wirelessly and help to deliver the data packets of neighbouring devices by routing the packets through the best possible path and thus enabling any device in the network to connect to the internet. The problem with using these devices for routing is that people have no incentive to leave their devices on as active routing nodes and this creates dead spots in the mesh network making it unreliable [2]. In such a situation, an incentivesed mesh network would be beneficial as it would allow devices to pay other devices for helping to successfully deliver their data packets. Since data is communicated in packets and due to the fact that a mesh network requires routes to be chosen dynamically, it needs to be possible for devices to make micropayments so that they can pay other devices in amounts proportional to the service that they deliver.

Micropayments are not feasible using payments facilitated through third parties such as banks due to the high transaction fees and slow processing times. For the same reasons, micropayments are not feasible using cryptocurrency payments facilitated through blockchain technology alone [3]. With the introduction of payment channels and the Lightning Network, it can be possible to integrate secure and fast

micropayments into the IoT world.

1.2 Objectives of this study

1.2.1 Problems to be investigated

In this study the following main topics will be investigated:

- The workings of Bitcoin and the Bitcoin blockchain along with the workings of payment channels and the Lighting Network and how these technologies can be used to facilitate secure, trust-less and fast micropayments between IoT devices.
- The requirements of using Bitcoin and Lightning Network technology for payments between IoT devices.
- The details and requirements of a mesh network.
- How a mesh network can be adapted to effectively use Bitcoin and the Lighting network to incentivise devices in the mesh network to contribute to relaying data.
- Methods for improving upon any similar implementations in order to make such an implementation viable for its intended purpose.

1.2.2 Purpose of this study

The purpose of this study is to investigate how blockchain and payment channel technology work and how they could be used to enable fast and secure device-to-device micropayments. An investigation will be made into how such micropayments could allow IoT devices to provide services to other IoT devices in exchange for payment on a pay-as-you-consume basis. The aim is then to implement a basic form of an incentivised mesh network which would be a realistic use case for device-to-device micropayments and to investigate possible improvements of such a system. The end result will then be analysed in order to determine if the proposed solution is viable. To be viable, the implemented mesh-network must be able to provide a realistic bandwidth for users of the system.

1.3 Scope and limitations

This project will focus on the payment aspect of device-to-device micropayments and how this can be used to create an incentivised mesh network. Mesh networks

are complex and this project will not go into detail regarding mesh network infrastructure or mesh network routing algorithms and will assume that the payment structure that will be discussed can be integrated into existing meshing structures. In terms of time and budget, there are 11 weeks within which to complete this project with a budget of R1500.

1.4 Plan of development

Figure 1.1 below shows the plan of development of the project and the report. A detailed description of this plan follows.

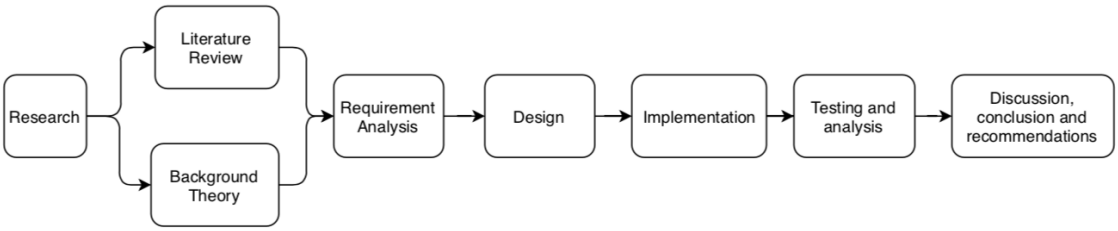


Figure 1.1: Plan of development

1. **Research:** The initial stage of this project involved doing enough research to develop a high level overview of the objectives of the project as outlined in Chapter 1.

2. **Background Theory:** In Chapter 2, a detailed description is given of various concepts that are crucial to the workings of Bitcoin, payment channels and multi-hop payments. This chapter is also used to explain the details of various encryption techniques and cryptographic concepts used in the design and implementation of the project.

3. **Literature Review:** Chapter 3 is used to describe and discuss the details of three other implementations of incentivised mesh networks.

4. **Requirement Analysis:** In Chapter 4, a detailed description of the system is given and using the description, various user requirements, functional requirements and acceptance test protocols are identified.

5. **Design:** In Chapter 5, the sub-systems and sub-processes of the system are identified and the design of each is explained.

6. **Implementation:** In Chapter 6, the implementation of each of the sub-systems and sub-processes described in the design section is described.

7. **Testing and Analysis:** In Chapter 7, a description is given of the tests done to ensure that they system works as is expected. The ATPs identified in the requirements analysis are checked to ensure that all the ATPs are passed. Lastly, the performance of the system is tested using a variety of tests and from

the results of these tests, interpolation is used to estimate the performance of a larger scale implementation of the mesh-network.

8. **Discussion, Conclusion and Recommendations:** In Chapter 8, the results determined in Chapter 7 are discussed. The bandwidth results are compared against the minimum requirements of various internet applications in order to determine if the system will be able to cater for these applications. In Chapter 9, a summary of the project aim and results is given and in Chapter 10 various recommendations are made for any future work.

Chapter 2

Theory

2.1 Bitcoin

Bitcoin is a cryptocurrency that allows people to send value (in Bitcoin) to one another by recording transactions on a public distributed ledger called a blockchain. In this section, various aspects of Bitcoin technology will be explained including how Bitcoin uses public-private key cryptography to protect a users access to their funds, how Bitcoin transactions work, how they are verified and how a blockchain is used to store these transactions. This section will also include the details of how Bitcoin technology can be used to form payment channels between users and then how payment channels can be used to create a Lightning Network.

2.1.1 Elliptic Curve Cryptography

Elliptic curve cryptography makes use of maths on an elliptic curve using numbers defined over a finite field [4]. Figure 2.1 shows the Bitcoin elliptic curve plotted over real-numbers. On such a curve, point scalar multiplication is easy to calculate but point scalar division is impossibly hard to calculate.

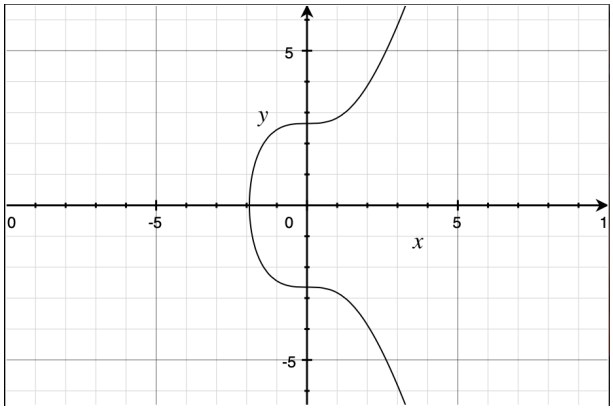


Figure 2.1: Bitcoin Elliptic Curve

For example, if a publicly known generator point on the curve is $G = (g_x, g_y)$ and this point is then multiplied by a constant k then another point on the curve can be found: $P = (p_x, p_y)$. Due to the fact that this maths is evaluated over a finite field, it is easy to find point P if both G and k are known but it is not known how to calculate k if P and G are known. This asymmetric property is illustrated in figure 2.2.

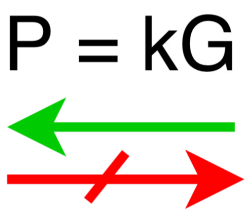


Figure 2.2: Public-private key asymmetric property

Bitcoin uses public-private key cryptography based on this asymmetric property [4]. Each Bitcoin user has a private-key, k, which they use to produce a public-key, P. The public-key can be shared without the risk of an attacker being able to determine the private-key, k.

Bitcoin users can use their private-key to sign and therefore spend the outputs of transactions that were addressed to their public-key and anyone can see this signature and verify that it was produced by the person with access to the private-key that generated the public-key, P, and they can do this verification without knowing the private-key, k.

2.1.2 Transactions

Bitcoin coins are chains of transactions that represent changes in ownership of the coins. Each transaction is made up of inputs and outputs. The inputs of a transaction point to previously unspent transaction outputs and include a signature to sign the spending of these outputs. The outputs of a transaction include the public-key addresses to which the inputs should be paid to. These outputs will only be spendable by an entity who owns the private-key that produces the public-key in the output and this is done by creating a valid signature [5].

Figure 2.3 shows an example of the transactions involved if a user, Alice, wanted to pay another user, Bob. In this example Alice has private-key $k1$ with which she produced her public-key, $P1$ and Bob has private-key, $k2$, with which he produced public-key, $P2$. To pay Bob, Alice creates transaction TX2. TX2 has an input that references the output of transaction TX1 and Alice is able to sign this input due to the fact that the output in TX1 is addressed to her public key, $P1$. Since she owns the private-key that produced $P1$ she is able produce a valid signature which Bob can then verify. Alice constructs the output of TX2 so that it is spendable by anyone who has the private-key corresponding to public-key $P2$ which in this case would enable Bob to spend this output.

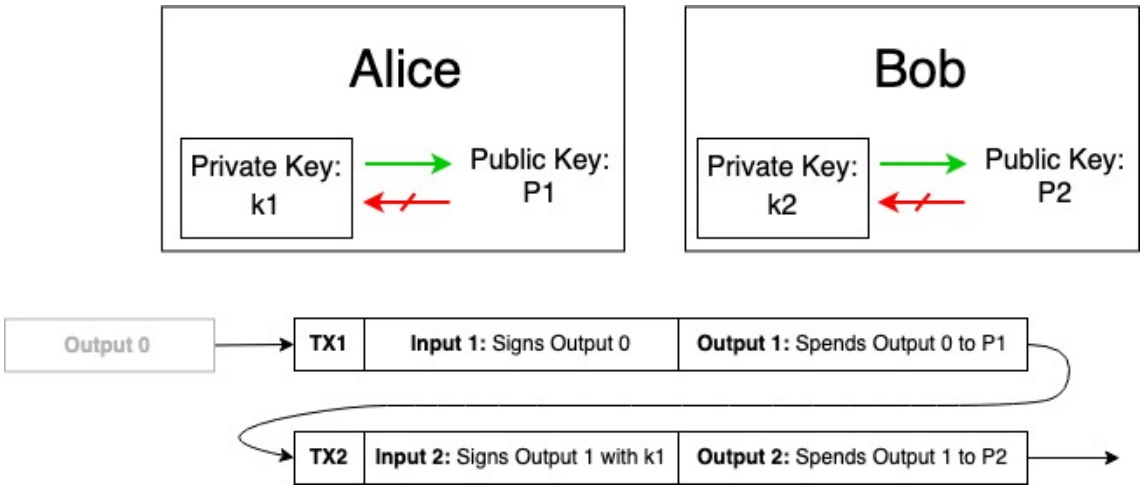


Figure 2.3: Transactions

A different type of transaction called a multi-signature (multi-sig) transaction can also be formed. The output of such a transaction would require multiple signatures in order to be valid. Figure 2.4 shows an example of such a transaction. In this example, the multi-sig transaction is transaction TX2 and the parties involved are Alice, with keys $P1$ and $k1$, and Bob, with keys $P2$ and $k2$. This transaction has two inputs, one of which references an output spendable by Alice and the other an output spendable by Bob. The output of the transaction is a 2-of-2 multi-sig script that is only spendable if the input that references it is signed by both Alice and Bob [5] [4]. This type of transaction forms the basis of payment channels which are explained in section 2.1.4.

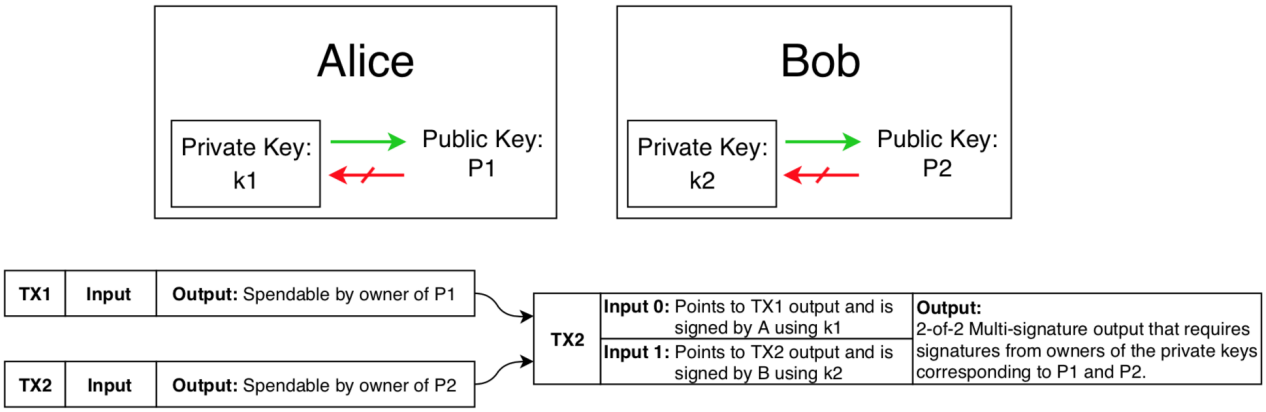


Figure 2.4: Multi-sig Transaction

2.1.3 Blockchain

For a Bitcoin transaction to be valid, it must be broadcast to the entire Bitcoin network and this is done by including transactions in the blocks of a public blockchain. This enables any user of the system to validate any transaction and trace back the origins of the transaction inputs as can be seen in figure 2.5. For a transaction to be

included into a block and mined, it is necessary to incentivise the miners by means of transaction fees. In the Bitcoin network, it takes approximately 10 minutes for a block to be mined and added to the blockchain. For these reasons publishing a transaction on the blockchain (an on-chain transaction) is both costly and slow. Using on-chain transactions for micropayment transactions is thus not feasible nor scalable [5]. Payment channels provide a way to perform fast and cheap off-chain transactions and these are discussed next.

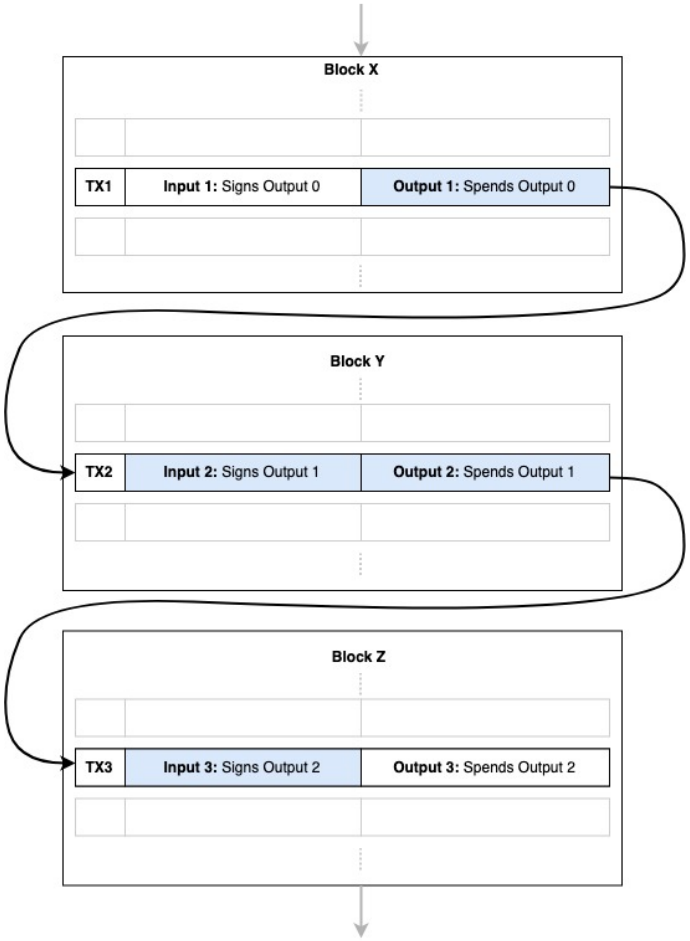


Figure 2.5: Blockchain

2.1.4 Payment Channels

Payment channels provide a way for two parties to exchange an unlimited number of Bitcoin transactions (within the limits of the channel capacity) and do so off-chain with minimal transaction fees. In this section, the aim is to explain the basic setup and use of a payment channel, to define what *funding transactions* and *commitment transactions* are and then to provide an example of the process of setting up and using a payment channel. More details regarding payment channels can be found in reference [5].

To set up a payment channel, two parties must both commit funds to a 2-of-2 multi-signature script that is only spendable if signed by both parties. This commitment

of funds to a multi-sig script is done by creating a transaction called a *funding transaction*. This funding transaction will need to be broadcast to the public Bitcoin blockchain and verified (on-chain). Once the funding transaction is verified, the two parties can exchange an unlimited number of off-chain transactions called *commitment transactions*. Commitment transactions are valid Bitcoin transactions that spend the on-chain funding transaction and determine how the funds of the channel are divided between the two parties. Each time the parties both sign a new commitment transaction that changes the division of funds, any previous commitment transactions become undesirable to use because parties who broadcast older commitment transaction risk loosing all the money that they committed to the channel if they are caught (this is done using asymmetric revocation contracts which will not be described here). At any point in time, one of the parties can broadcast the latest, valid, commitment transaction to the Bitcoin network and effectively spend the funding transaction, close the payment channel and distribute the funds according to the state defined by the latest commitment transaction.

An example of two parties, A and B, setting up a payment channel and using commitment transactions to determine the division of funds in the channel is described below:

Step 1:

Two parties, A and B, decide to set up a payment channel and do this by each of them committing funds to a 2-of-2 multisig (see section 2.1.2). Both parties sign this transaction and then one party broadcasts it to the blockchain as an on-chain, funding transaction. This transaction and the future settlement transaction will be the only two transactions that need to be published to the blockchain. In the example in figure 2.6, A commits 10 satoshis and B commits 5 satoshis to the channel.

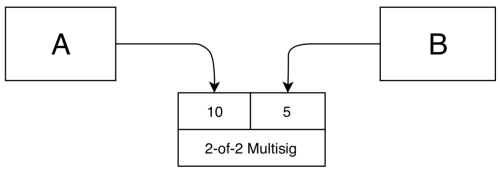


Figure 2.6: Payment channels step 1

Step 2:

Once the funding transaction is mined in a block, off-chain exchanges between A and B can start. In the example shown in figure 2.7, A decides to pay B 1 satoshi and so creates a commitment transaction that spends the funding transaction output and creates one output that pays 9 satoshis to A and another that pays 6 satoshis to B. A signs this transaction and sends it to B.

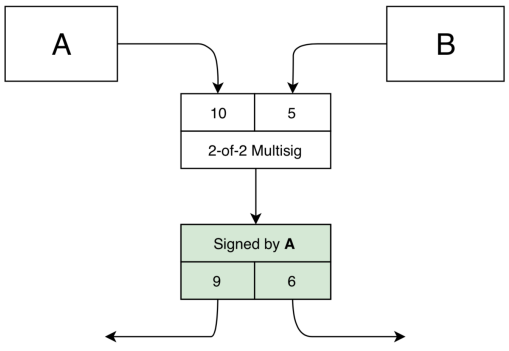


Figure 2.7: Payment channels step 2

Step 3:

B can analyse the commitment transaction and see that it would pay him 1 satoshi more than he had. B then signs the transaction and sends it back to A. Both A and B now hold the valid commitment transaction that would correctly pay A and B if it were to be broadcast to the blockchain. But instead of broadcasting it, both parties store it and agree to update the channel state to reflect the new commitment transaction. Both parties have the ability to broadcast the valid commitment transaction to the blockchain if necessary. See figure 2.8.

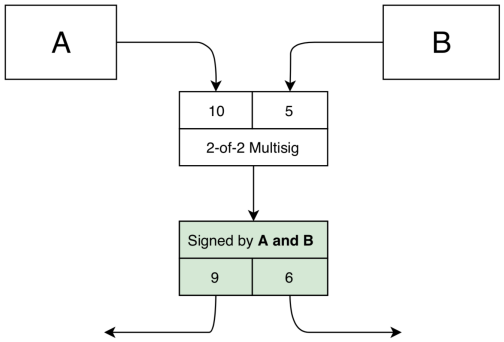


Figure 2.8: Payment channels step 3

Step 4:

If A want to pay B another 1 satoshi, then A creates a new commitment transaction which spends the same funding transaction and repeats the process of step 2 and 3. When A and B both sign the new commitment transaction then the previously created commitment transactions are made to be invalid (this is done using asymmetric revocation commitments which will not be discussed here). Both parties then store the new, valid commitment transaction and update the channel state accordingly. See figure 2.9.

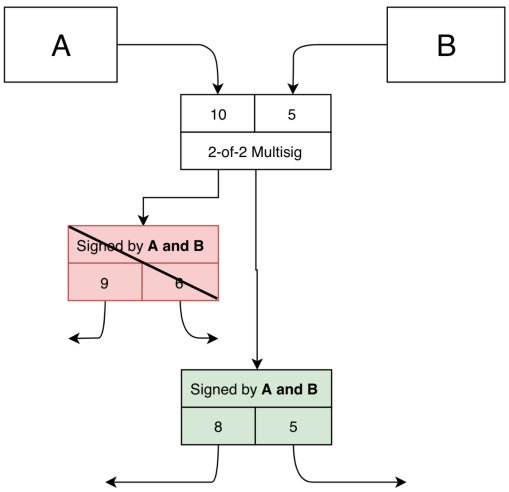


Figure 2.9: Payment channels step 4

Using payment channels, parties can create an endless number of commitment transactions to pay each other back and forth and can do so without needing to consult the blockchain. No fees are required to create new commitment transactions and no trust is required between the parties. If one party, for example B, acts maliciously by broadcasting an older commitment transaction that is more in its favor than the newest commitment transaction then party A can broadcast the newest transaction, invalidate the older transaction and claim all the funds within the channel. Node B is thus incentivised to act honestly.

2.1.5 The Lightning Network

The Lightning Network is made up of nodes connected by payment channels and enables micropayments to occur between any two nodes on the network as long as there is a path of payment channels between these nodes. The Lightning Network uses a combination of payment channel technology and a type of smart contract called a Hash Time Lock Contract (HTLC). A simple example will be used to explain how the Lighting Network can be used for secure payments across the peer-to-peer network. The details of this section can be found in [5].

Example:

In this example, the following assumptions can be made. See Figure 2.10 for a visualisation of the initial set-up.

- Party A wants to pay party C 10 satoshis
- Payment channels exists between A and B (100 to A and 20 to B) and another between B and C (50 to B and 10 to C).
- B charges a 5 satoshi routing fee
- A has the hash, H, which is the hash of a secret pre-image value, X, such that $H = \text{hash}(X)$
- A knows that C, and only C, has access to X (this can be achieved using various methods).
- The initial total wallet balances of each node is as follows:
  - A: 100 satoshis
  - B: $20 + 50 = 70$ satoshis
  - C: 10 satoshis

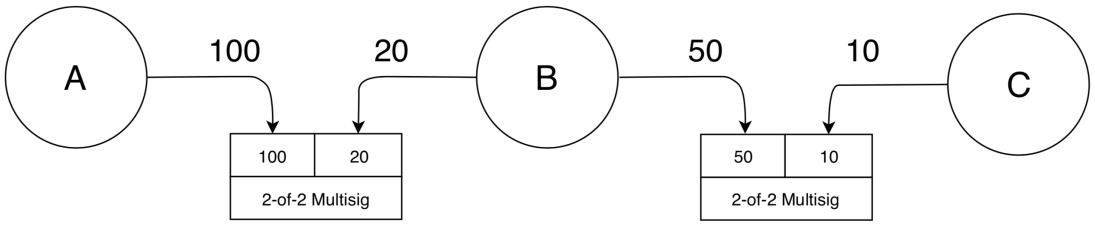


Figure 2.10: Multi-hop payment set-up

Step 1:

A creates a new commitment transaction with B. This commitment transaction spends the funding transaction of the $A - B$ channel and has three outputs:

1. **Output 1:** 20 satoshi to B
2. **Output 2:** 85 satoshi to A

3. **Output 3:** 15 satoshi to a HTLC that has two clauses and will be spendable by which ever clauses condition is first satisfied. The first clause pays B if B is able to produce the pre-image of H. The second clause refunds A if a certain time period, t, has passed.

A signs this commitment transaction and presents it to B. See figure 2.11 for an illustration of this step.

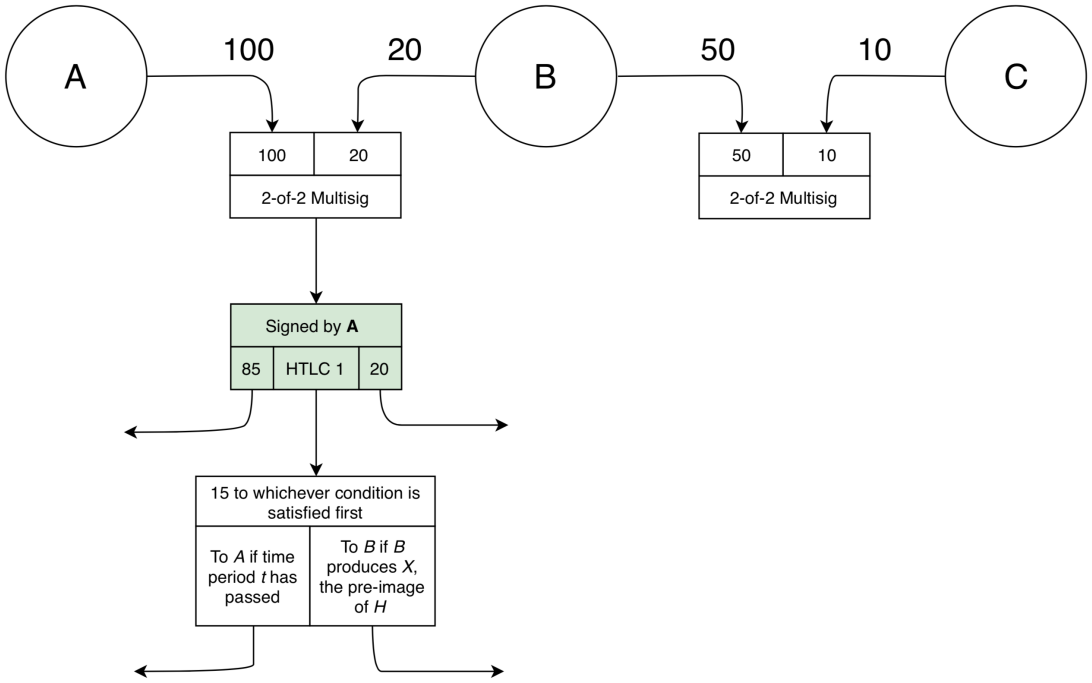


Figure 2.11: Multi-hop payment step 1

Step 2:

Party B will sign the commitment transaction proposed by A as it is clear that no money will be lost if it is not able to produce the hash pre-image and that it will gain 15 satoshis if it is able to produce the pre-image. B then repeats a similar process with node C and constructs a commitment transaction that spends the funding transaction of the $B - C$ channel. The outputs of this transaction are as follows:

1. **Output 1:** 10 satoshi to C
2. **Output 2:** 40 satoshi to B
3. **Output 3:** 10 satoshi to a HTLC that has two clauses and will be spendable by which ever clauses condition is first satisfied. The first clause pays C if C is able to produce the pre-image of H. The second clause refunds B if a certain time period, t, has passed.

B signs this commitment transaction and presents it to C. See figure 2.12 for an illustration of this step.

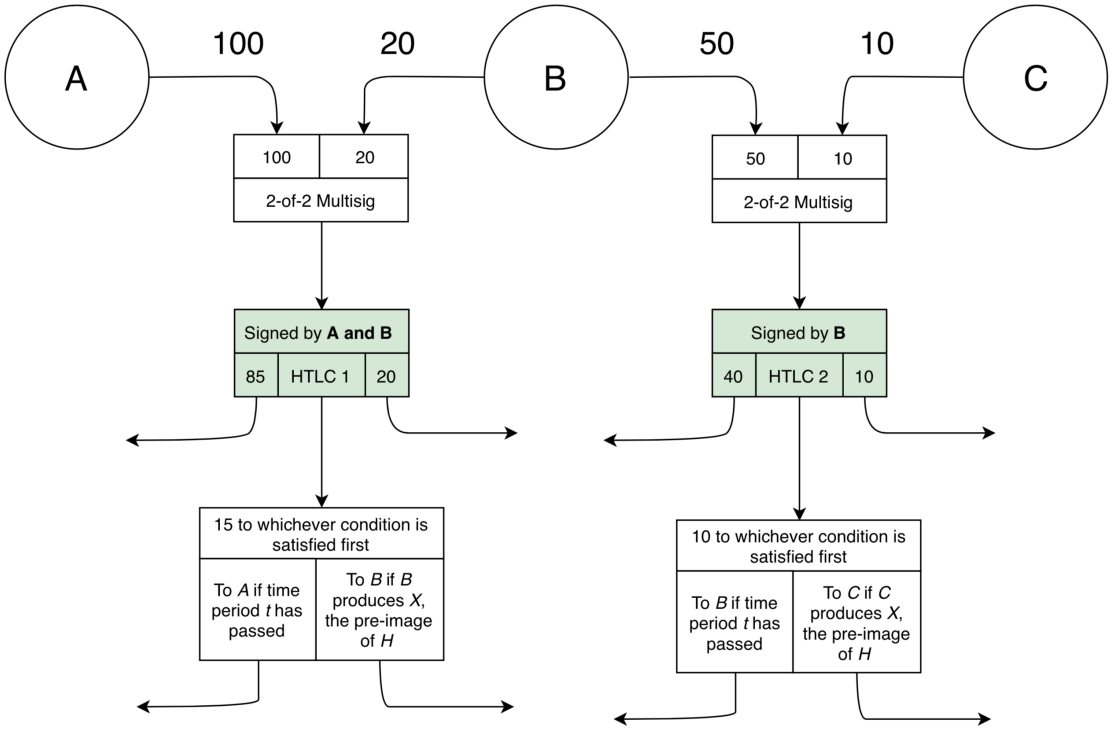


Figure 2.12: Multi-hop payment step 2

Step 3:

C receives the commitment transaction proposed by B and sees that it is safe to sign since C has access to the pre-image of H, X, and can thus claim the corresponding HTLC output. C signs the transaction and sends it back to B along with X so that B can see that C can claim the 10 satoshis committed to HTLC2. Both B and C update their channel state to reflect this. See figure 2.13 for an illustration of this step.

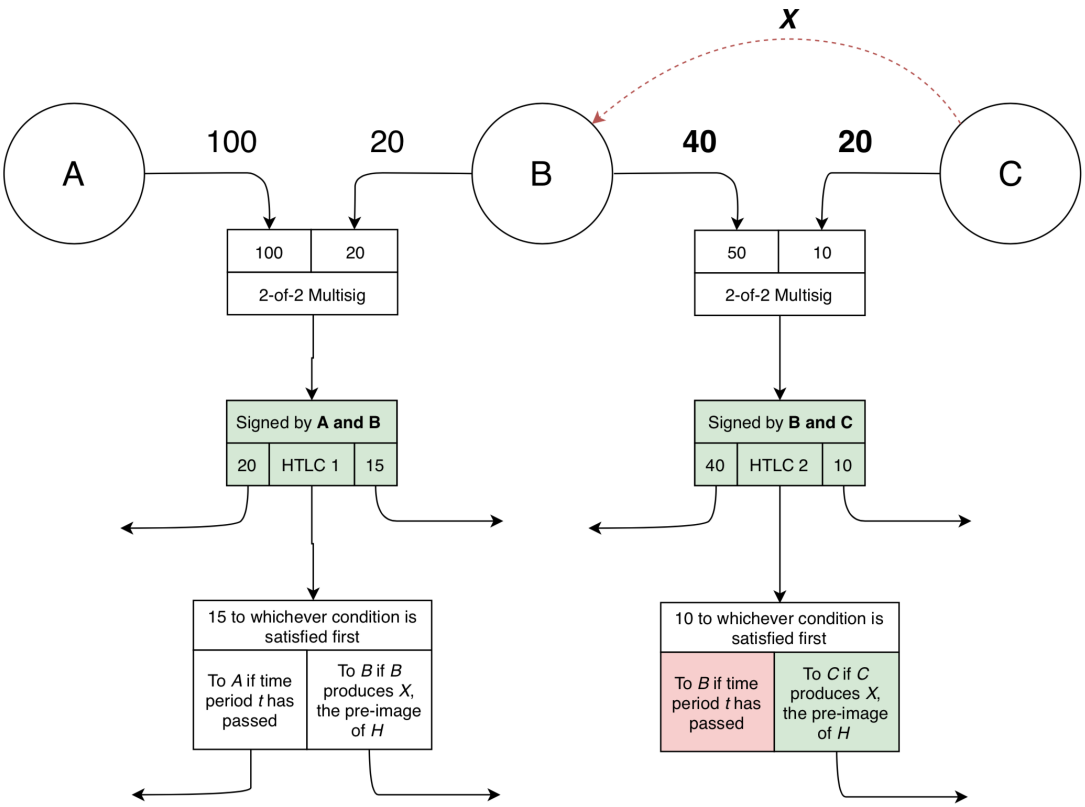


Figure 2.13: Multi-hop payment step 3

Step 4:

B receives pre-image, X, from C and sees that it is now able to claim the 15 satoshis locked to the HTLC output form the commitment transaction proposed by A. B then sends X to A and both A and B update their channel state to reflect the payments. See figure 2.14 for an illustration of this step. The final total wallet balances are now as follows:

- A**:** 85 satoshis
- B**:** $35 + 40 = 75$ satoshis
- C**:** 20 satoshis

It is clear that C has made 10 satoshis, that B has made 5 satoshis in routing fees and that A has spent 15 satoshis and has successfully paid C 10 satoshis.

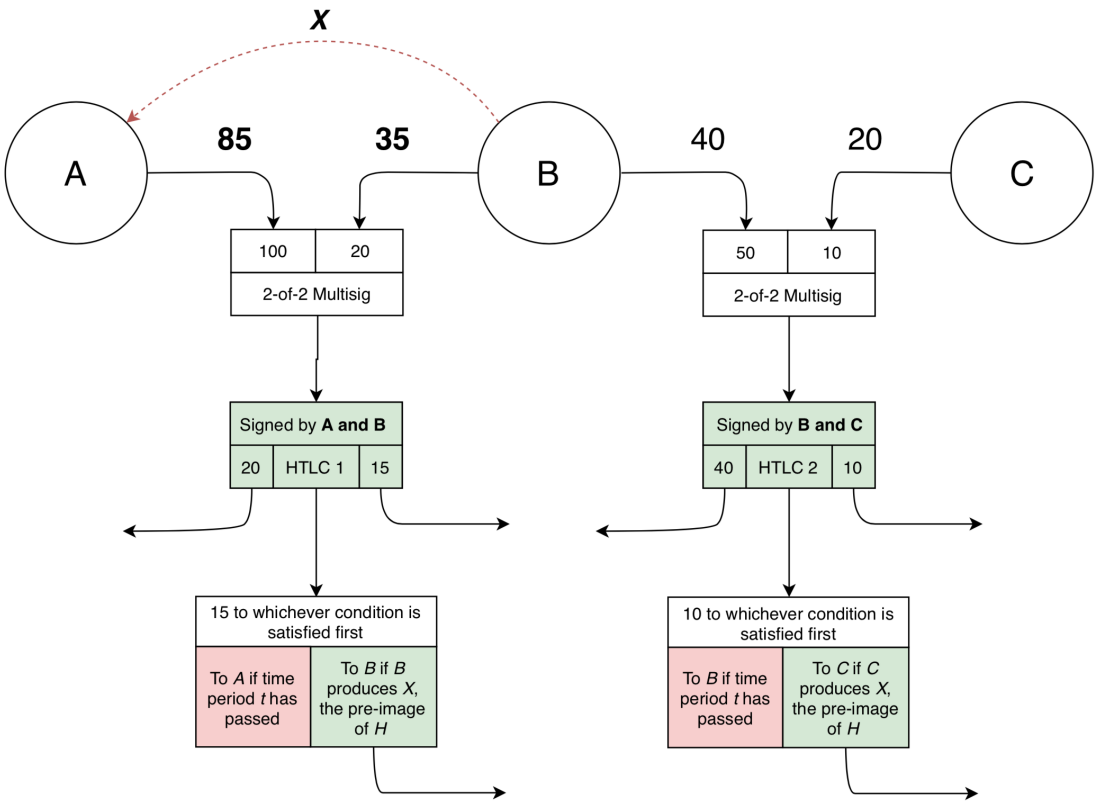


Figure 2.14: Multi-hop payment step 5

2.2 Symmetric key encryption using Bitcoin private-public key pairs

As indicated in section 2.1.5, the destination node and client node must be able to have access to some shared secret X. This section explains a method of achieving this when there is no way for the two nodes to directly communicate as will be the case in a mesh network.

Assume that node A wants to encrypt a message so that only node C can decrypt it. This can be done with shared-key symmetric cryptography in which it is assumed that the two communicating entities have some shared secret key which they can use to encrypt and decrypt messages using XOR-encryption. Since these nodes are assumed to be on a mesh-network with no direct communication path between them, they need to be able to create a secret shared-key without needing to communicate it to each other [6].

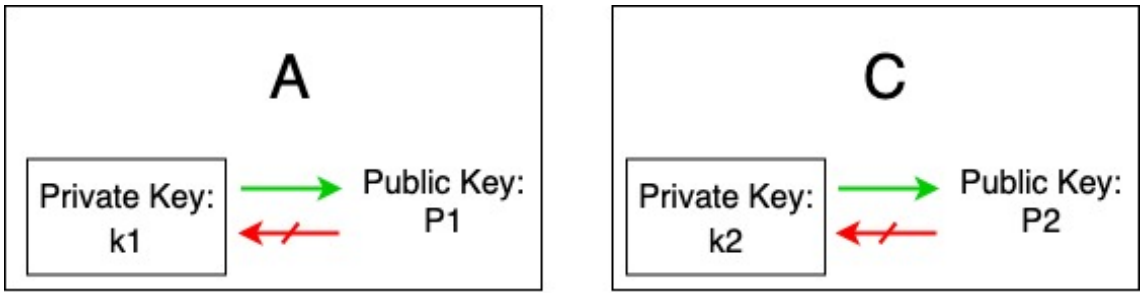


Figure 2.15: Node A and node C set-up

2.2.1 Shared Symmetric Secret Key

A and C both have a Bitcoin private-public key pair as shown in figure 2.15. As explained in section 2.1.1, these key pairs are generated using finite-field elliptic-curve cryptography using the asymmetric equation: $P = kG$, where G is a known generator point, k is a private key and P is the public key. Nodes A and C can then independently calculate the secret symmetric key that they will use for encryption and decryption. This can be done using the properties of Bitcoin private-public key pairs as follows:

- Node A knows the following: $k1, P1, P2$
- Node C knows the following: $k2, P2, P1$

Both A and C can then calculate points $S1$ and $S2$ respectfully:

- Node A: $S1 = k1 * P2$
- Node C: $S2 = k2 * P1$

It can be shown that the shared secret $S = S1 = S2$:

$$S1 = k1 \times P2 = k1 \times (k2 \times G) = k1 \times k2 \times G = k2 \times (k1 \times G) = k2 \times P1 = S2$$

2.2.2 Exclusive-Or (XOR) Encryption

As shown above, nodes A and C both have the ability to calculate the same shared symmetric key. This key can be used to encrypt and decrypt messages using XOR encryption [7]. The properties of the XOR operation are as follows:

$$a \oplus 0 = a$$
$$a \oplus 1 = 1 - a$$
$$a \oplus 1 = 1 - a$$

If A wants to encrypt message, m, with secret key, s, to create cipher-text, c, then this can be done as follows:

$$c = m \oplus s$$

Cipher-text, c, can safely be transmitted through the mesh network and when it reached its destination, node C, then it can be decrypted by C as follows:

$$m = c \oplus s$$

This can be proven using the properties of the XOR operation as follows:

$$c \oplus s = (m \oplus s) \oplus s = m \oplus (s \oplus s) = m \oplus 0 = m$$

2.3 Shamir Secret Sharing

Shamir Secret Sharing is a cryptographic algorithm used to break a secret up into N different pieces called shares. To reconstruct the secret, some or all of the shares are required [8].

Shamir Secret Sharing is based on the idea that to reconstruct a polynomial of degree k, only $k-1$ points on the polynomial are required. The process is outlined below:

- Set a threshold, k, and total number of shares, n.
- Chose a random secret, S. S is an element of a finite field, F.
- Choose $k-1$ random coefficients $a_1, ..., a_{k-1}$ and let a_0=S.
- Use the coefficients to construct a polynomial, $f(x)$:

$$f(x) = a_0 + a_1 x + a_2 x^2 + ... + {a_{k-1}}^{k-1}$$

- Sample n points from the polynomial as follows:

$$let \quad i = 0, ..., n \quad to \quad get \quad (i, f(i))$$

Each point is a share. At least k of these shares are required to reconstruct $f(x)$ and hence find a_0 which is equivalent to secret S.

2.4 Wireless Mesh Networks

A mesh network is a type of decentralised network made up of interconnected devices. In a WiFi mesh network, the devices in the mesh network are connected to each other through their local, on-board Wifi access points. Mesh networks enable devices in the network to work together in delivering data packets and do this by enabling nodes to only need to transmit and receive packets from their neighbouring nodes instead of all the way to a central router. The mesh network devices then work together to route a client nodes packets to a destination. The routes chosen are updated dynamically to so that the path chosen is always the most stable and well connected route. The advantages of a mesh network is that more devices are able to have access to services such as internet connection from a central router which also means that these services will cost less for users as they will be able to share internet services [9].

As mentioned, an important feature of a mesh network is that routes for data packets are updated dynamically to reflect the current state of each node in the network. When implementing a mesh network, it is thus important to provide a way for devices to communicate their current state to the rest of the network.

An example of a mesh network can be seen below in figure 2.16 where nodes are connected to each other using WiFi connections and node C has a reliable internet access. In this example, if node A wants to send a data packet to the internet it has two possible routes to chose from: $A - B - C$ or $A - D - C$. Node A would first evaluate the quality of the two routes and would then chose to send its data packet along the route with the most reliable connectivity.

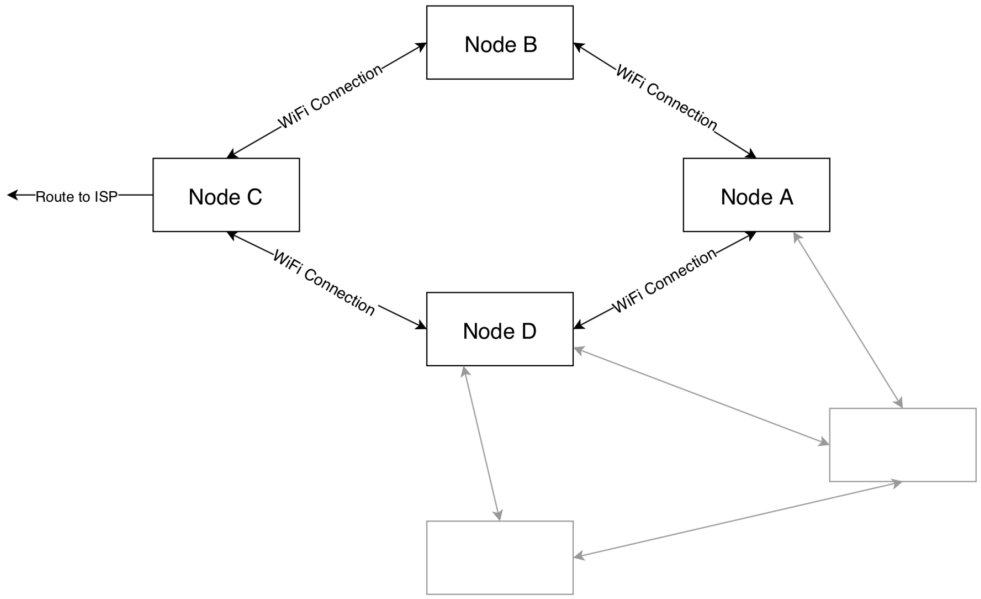


Figure 2.16: Mesh Network example

Chapter 3

Literature Review

During the research phase of this project, three other incentivised mesh networks were found that make use of blockchain technology. In this section, these different implementations will be described and analysed and conclusions will be drawn regarding the strengths and flaws of each implementation.

3.1 Althea

3.1.1 Aim

The Althea project aims to solve the problem of high cost and unreliable internet connection in developing worlds by replacing centralised ISPs with decentralised mesh networks. Due to the fact that the project is for developing areas and rural communities, a major aim of the project is to make it as affordable as possible as well as to make the network as easy to join as possible. Anyone should easily be able to join and participate in the decentralised Althea network so that they can send and receive data as well as be paid for helping other devices route data through the network [10].

3.1.2 Method

The Althea network allows any device with Althea software to join a local mesh network and immediately participate in the network. Any time a user wants to use the network to access the internet or send data to another device, the device will dynamically find the best route to the destination in terms of price and quality and will route the data packets accordingly. In order to incentive devices in the network to contribute with packet routing, they charge a routing fee which the owner of the device can set and this price will be broadcast throughout the network so that other devices can update their route choices dynamically.

The Althea network requires that devices only need to pay their neighbouring nodes to forward packets. The neighbouring node will then subtract their routing fees from this payment and will use the rest to pay the next hop in the route to the destination node. To make payments between devices, Althea uses its own blockchain built on the Cosmos platform. This blockchain is maintained and validated by a selection of the devices in the network who have the capacity to store this blockchain. What this means is that each device that does not store and validate the blockchain will need to trust the information received by the blockchain validating devices.

In the Althea network, nodes make payments after receiving service. This means that when a device wants to send a packet through the network to an ISP, each node in the route will only pay the next hop in the route once there is confirmation that the data packet has been delivered. Each node will then pay the next node in the route by broadcasting a transaction onto the public blockchain. If any node does not make a required payment, firewalls will be used to block that node from participating in the network in future.

Althea identifies three possible roles that a device in the network could play. These roles are illustrated in figure 3.1. The first is the role as a user. A user device is one from which internet requests will be made and this device will need to identify the best route through the network for packets to be sent. Devices can also play the role of a relay which helps forward traffic for other devices and earns money for doing so. The third role is a gateway. Gateway devices are relay devices that are connected to an internet source (such as an ISP) and can forward packets to and from the internet for other devices in the network.

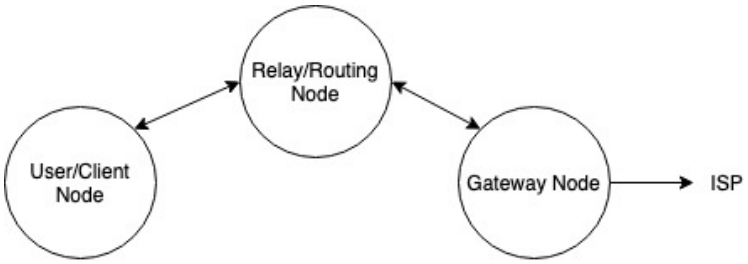


Figure 3.1: Althea Device Roles

3.1.3 Discussion

There are many flaws in with the Althea system. The first is that each device is required to trust another device the network for information regarding blockchain transactions. Any model of trust such as this is not scalable and creates a centralised network. Another problem with the blockchain is that it is custom built for the Althea network which means that there will be relatively few devices validating the blockchain and this makes it easy for attackers to join the network as a validating node and act maliciously by performing 51% attacks. Using a public blockchain for a system that will require thousands of micropayments to be recorded every day is not scalable as the storage requirements required for such a system will grow very quickly which will be a disincentive for anyone wishing to turn there device into a

validating node. Lastly, the system relies on devices only paying after receiving a service. Even though there are measures in place to block malicious devices from the network, this can only be done after the fact. This means that some devices will be left unpaid even after providing a relay or gateway service for the malicious device.

3.2 RightMesh

3.2.1 Aim

RightMesh is a company that aims to integrate blockchain and token technology into mesh networks. RightMesh believe that to create a strong mesh network, participants of the mesh network must be compensated for their data, battery and processing power that is used for helping to relay the data of other devices. RightMesh think that granular (micropayments) must be possible in such a network so that devices can pay for exactly what they receive unlike paying for regular ISP service packages where users are often overcharged. RightMesh believes that part of what makes a mesh network strong is its decentralised nature that does not depend on any one node in particular. Therefore if a payment structure is introduced into a mesh network, it is necessary that the payment structure also uses decentralised technology such as blockchain technology. RightMesh is focused on creating an ad-hoc mesh network which means that mobile devices in the network must be able to join, leave or move around in the network easily [11].

3.2.2 Method

RightMesh uses their own token, RMESH, built on top of the Ethereum blockchain. They chose to use their own token so that the price of the token could be less volatile and could be reliably linked to the amount of data being consumed. Like the Althea network, each node in the mesh network is paid when they assist with transmitting data packets to their destinations but unlike Althea, these payments are not made on the blockchain but rather using payment channels (see Chapter 2, section 2.1.4).

RightMesh acknowledges the fact that mobile devices do not have the storage capabilities required to run full cryptocurrency nodes (by storing the entire blockchain). Each device will have a private and public key and will be able to sign transactions but will need to pass these transactions through the mesh network to a device that is a full node device that is permanently connected to the Ethereum blockchain. Such a device is called a SuperPeer and it is required that each device in the mesh network is able to connect to a SuperPeer and that they establish a payment channel (section 2.1.4) with the SuperPeer. Devices on the mesh network can then pass transactions to these SuperPeers who will then broadcast the transactions on the public blockchain and send confirmations back to the original device.

To participate in this mesh network, devices must establish a payment channel with a SuperPeer (see figure 3.2). For a client node to send traffic to the gateway node, it must determine the best route to take though the mesh network, establish the price of the route and then must pay the SuperPeer the total amount and send the data packet to the next really node. The SuperPeer is then responsible for paying each node in the network who contributes to delivering the packet sent by the client node. Figure 3.3 shows the flow of payments that occur between the different nodes.

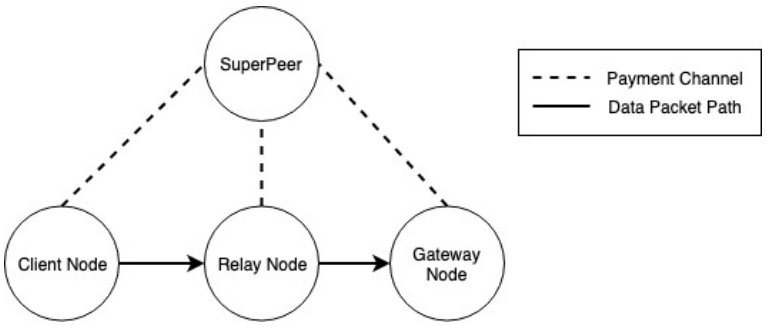


Figure 3.2: RightMesh Device physical connections and payment channels

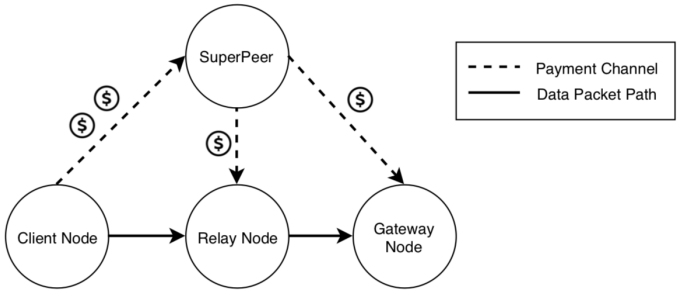


Figure 3.3: RightMesh payment process

3.2.3 Discussion

Unlike Althea, RightMesh uses payment channels between participating device nodes and SuperPeer nodes. This is a more scalable solution due to the fact that many micropayments can be made between the devices without needing to broadcast the transactions on-chain. However, RightMesh devices are required to trust a third party SuperPeer node for all transactions. This makes the network more centralised and leaves participants vulnerable to malicious SuperPeer nodes.

3.3 Lot49

3.3.1 Aim

The Lot49 project aims to build a trust minimised incentivised mesh network for low-bandwidth, mobile mesh networks and to do so using payment channel technology along with Bitcoins decentralised public blockchain.

The Lot49 white paper [12] explains the implementation of the system. The implementation is based on the Bitcoin Lightning Network but is adapted so that the overhead required for payment channel updates is lowered. The Bitcoin Lightning Network implementation requires commitment transactions to be transmitted between parties and each commitment transaction will contain either one or two digital signatures. The size of the signature data results in a large overhead which is a significant problem for low-bandwidth networks. Thus, much of the Lot49 implementation is focused around lowering the data transfer required for signatures.

3.3.2 Method

Like the Bitcoin Lightning Network, Lot49 uses connected (multi-hop) payment channels to enable payments to be made between nodes that do not have direct payment channels established between them. Lot49 then proposes to use the payment channel network to create an incentivised mesh-network and to do so by routing both data and payments over the same communication channels. This idea is illustrate in figure 3.4 where it can be seen that for every direct, physical mesh-network connection between nodes, there is an associated logical payment channel connection.

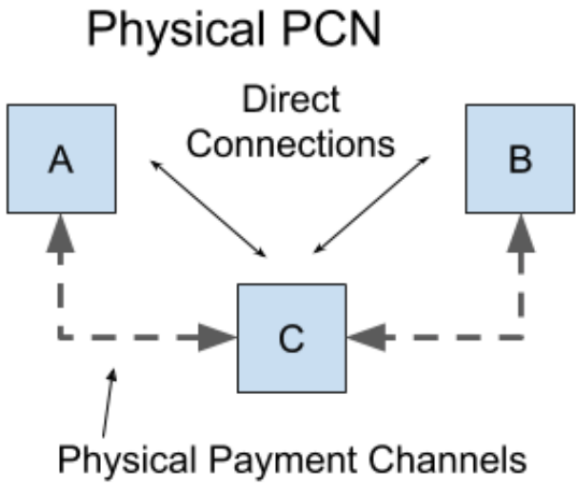


Figure 3.4: Lot49 physical and logical connections [12]

To transmit data through the network and make the necessary payments, the following is done:

- The payload to be transmitted will include a body and a header. The body of the payload will consist of the message that must be transmitted along with an encrypted secret that only the destination node will be able to decrypt. The payload header will include route information along with commitment transaction required to perform a payment channel update.

- The commitment transaction will include a script with rules on how to update the payment channel. The script will propose to pay the next node if the node can prove that the message of the payload has been successfully delivered to the destination node and does this by revealing the secret that was encrypted and included in the body of the payload. If the secret is not revealed then the payment will be refunded to the node that constructed the commitment transaction. This means that payments only happen once there is proof of data delivery and this process of settling payments in exchange for data delivery is atomic and requires no trust between nodes. Both nodes will store the latest commitment transactions so that they can publish them on-chain to prove the state of the channel if necessary (in the case of a malicious node who tries to prove an false state).

- This process is illustrated in figures 3.5 and 3.6. Figure 3.5 shows how a payload constructed by node A is passed along the route $A - B - C - D$ to destination node D. Node A includes an encrypted secret in the body of the payload that only node D will be able to reveal. Node A takes the hash of this secret and includes the hash in the script of the payment channel update that is included in the payload header. The script will confirm to node B that if it can prove successful delivery of the payload then it will be paid in the payment channel that exists between nodes A and B. This process is repeated between nodes B and C and again for nodes C and D.

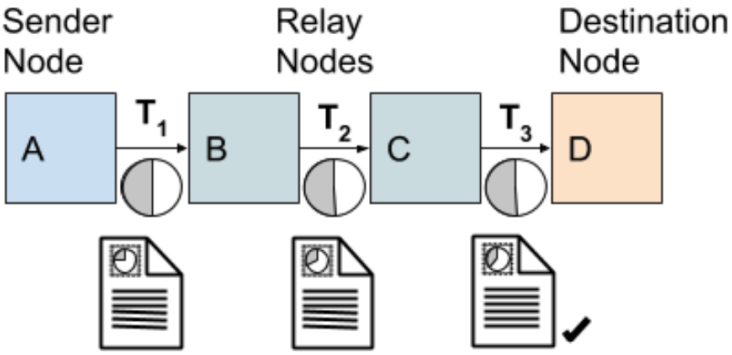


Figure 3.5: Sending data and waiting for proof of delivery [12]

- When node D receives the payload, it is able to reveal the hidden secret and thus claim the corresponding payment proposed in the commitment transaction created for the payment channel between nodes C and D. This process is repeated for the channels between nodes B and C and again for the channel between nodes A and B. This process is illustrated in figure 3.6 below.

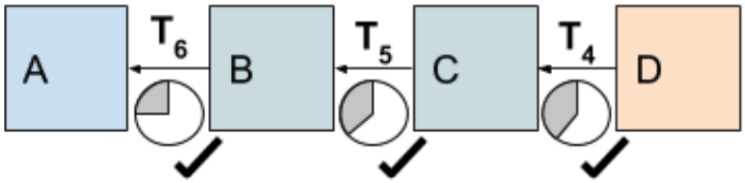


Figure 3.6: Payment channel updates after proof of delivery [12]

The transfer of value (in the form of Bitcoin tokens) is done through direct communication between nodes and requires no internet access or consultation with the Bitcoin public blockchain. The only time that nodes are required to have internet access is when funding transactions need to be broadcast to the blockchain in order to set up payment channels or when a commitment transaction or settlement transaction needs to be broadcast when a payment channel is being closed or in the case where a commitment transaction must be be broadcast after one party acts maliciously.

3.3.3 Discussion

Unlike the RightMesh an Althea implementations, the Lot49 system requires no trusted third parties in order to confirm and communicate transactions as all payments are made directly between nodes. Lot49 also makes the process of data delivery and payment for the data delivery atomic whereas in Althea and RightMesh, payments are made after receiving service and so there is a chance of malicious nodes not making the required payments.

Due to the fact that payments and data are routed through the same network routes, it means that the number of commitment transactions that are required to be created is directly proportional to the length of the the mesh-network route. Each commitment transaction requires two signatures to be created. Digital signature creation is a computationally expensive process and so it is expected that this will negatively effect the time required to transmit a single packet. In addition to this, the Lot49 implementation requires the entire process of creating and signing new commitment transactions to take place for every packet that is transferred. Thus if a client node is required to send multiple packets to the destination node then then number of signature operations required to complete the process could make the solution infeasible.

Chapter 4

Requirements Analysis

4.1 System Description

4.1.1 High level description

The project aim is to explore the implications and viability of facilitating micropayments between IoT devices using Bitcoin and payment channel technology. The chosen use-case for this project is an incentivised mesh network. A mesh network is useful for providing wide connectivity in an area but can only do so when enough devices partake in the network. Device-to-device micropayments could solve this problem by allowing devices to pay other devices for helping to transmit their data through the network. Payments such as this will incentive people to leave their devices connected to the mesh network even when they do not require any services because they will be able to earn money for contributing to the mesh network. The micropayments between devices will be facilitated using Bitcoin and payment channel technology. A working implementation should allow data to be routed through the network and relay nodes should be compensated for assisting in this. This process must happen fast so that realistic internet access can be provided.

4.1.2 Device Roles

In a wireless mesh network, there are three different roles that each device in the network must be able to play. These roles are as follows:

- **Client Node:** A device that is trying to connect to the internet or communicate with another node in the network without having direct access to either.
- **Relay Node:** A well connected device in the network that helps to route data for client nodes and requires compensation for doing so.

- **Gateway Node:** A device in the network with an internet connection. Such a device will be the destination device that a client node would want to send data too.

4.1.3 Description of a typical scenario

In this project, the aim is to implement a simple mesh network that can supply sufficient bandwidth to its participants with the focus being on how payments between nodes in the network can be made so as to incentives more people to connect their devices to the network. Figure 4.1 shows the layout of the simple mesh network that will be created as a proof of concept.

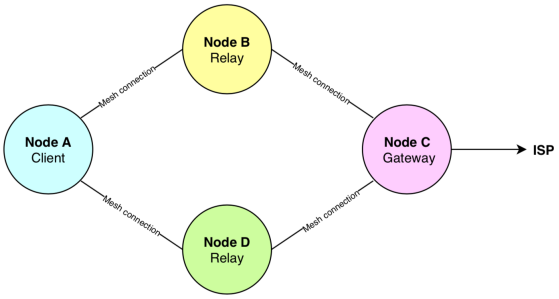


Figure 4.1: Mesh Network Connections

As can be seen in figure 4.1, the mesh network will consist of four devices connected to each other through wireless connections. For the sake of this project each device will play a dedicated role.

Node C is a gateway node that has a direct internet connection and requires compensation if other nodes make use of its services.

Node A is a client that requires data to be sent to the internet (perhaps a get-request or an image upload) but is not able to directly connect to the internet to do so. Node A routes its data packet through the mesh network to node C. To do so, node A must choose between two available routes: $A - B - C$ or $A - D - C$. In reality the choice between these routes will be based on a variety of factors such as cost and connection quality but for this project the choice will be made purely based on the cost of the route. Once node A has chosen a route to node C, node A will make the necessary payment only when there is proof that the data was successfully delivered.

Nodes B and D are acting as relay nodes on the mesh network. They require compensation for their services and want proof that they will be paid for the successful delivery of any data packet that they help deliver. Nodes B, C and D are all able to set their own routing fees and are able to accept payment in Bitcoin.

Since node A is not directly connected to node C, it is not able to pay node C directly and so it is a necessary requirement that nodes only need to pay their

neighboring node and only need to do so once the neighboring node has proved that it has successfully been delivered the data.

4.1.4 User Requirements

Using the system description described above, the user requirements for the system can be identified. These user requirements are outlined in table 4.1.

UR ID	User Requirement Description
UR1	Users without direct internet connection can connect their devices to the mesh-network and can receive a reliable internet connection.
UR2	Users require payments to be made using Bitcoin cryptocurrency and off-chain transactions using payment channel technology
UR3	Users require the ability to set a fee for their device to charge when it is contributing to the network as a relay node. The payments earned for routing data must be proportional to the size of the data packet being routed.
UR4	For each data packet sent by a client, the cheapest route must be chosen.
UR5	Users require that the data they wish to send across the mesh network only be readable by the intended destination device.
UR6	Users who use their devices in client mode, require assurance that they only compensate other nodes for relaying their data once there is proof that their data has been successfully delivered.

Table 4.1: User Requirements

4.1.5 Functional Requirements

Using the user requirements, the functional requirements of the system can be identified. These functional requirements are outlined in table 4.2

FR ID	Associated UR	User Requirement Description
FR1	UR1	A device is easily able to connect to the mesh-network via a wireless connection and can start using and supplying services.
FR2	UR2	Each device on the mesh-network must be able to create a valid Bitcoin private-public key pair along with a valid Bitcoin address. Each device must be able to construct, analyse, verify and sign funding and commitment transactions.
FR3	UR2	For a device to open a payment channel, it requires an initial wallet balance. This means that there must be a transaction on the public Bitcoin blockchain that the device is able to reference and validly sign.
FR4	UR3	Nodes must be able to advertise there current fee to the rest of the network.
FR5	UR4	Client nodes must be able to dynamically choose a route to the gateway (or destination) node based on the relay fees charged by other nodes in the network
FR6	UR5	A client node must be able to encrypt data in such a way that only the intended destination node will be able to decrypt it. This must be possible without requiring the two devices to have direct communication at any time.
FR7	UR6	Devices acting as relay nodes should only continue to create a commitment transaction with the next device en route if it is able to verify the validity of the commitment transaction that it received by the previous device.

Table 4.2: Functional Requirements

4.1.6 Acceptance Test Protocols

From analysing the system description, user requirements and functional requirements the acceptance test procedures shown in 4.3 have been developed.

ATP ID	**Associated FR**	**Acceptance Test Procedure Descriptions**
ATP1	FR1	Each device on the network automatically joins the mesh network when switched on and can perform client, relay and gateway node roles.
ATP2	FR2	Each device creates a unique Bitcoin private-public key pair and Bitcoin address.
ATP3	FR2, FR3	Devices on the network are able to create and verify funding or commitment transactions.
ATP4	FR4, FR5	If a device advertises a price that makes one route cheaper than the other then the client device must switch to start using the cheaper route immediately.
ATP5	FR6	Node A is able to encrypt data for node C and it must not be possible for node B to decrypt this data. Node A must be able to do this without any prior direct communication with node C.
APT6	FR7	Any attempt that a malicious device makes to falsely prove the successful delivery of data must be caught and there should be no change in channel states.
ATP7	FR1, FR7	Successful delivery of data results in corresponding payments to all the relay nodes and gateway nodes involved in the delivery.

Table 4.3: Acceptance Test Procedures

Chapter 5

Design

As identified in the literature review in section 3, current implementations of incentivised mesh-networks have many flaws. The Althea implementation uses a purely blockchain based solution that requires all payments between devices to be published to a public blockchain making the solution unscalable an unsuitable for fast micropayments between devices. The RightMesh implementation makes use of payment channels which results in a more scalable solution but it requires that devices open payment channels with trusted SuperPeer nodes rather than directly with each other. The RightMesh implementation is useful for a network where the devices in the network are expected to move around and therefore change the mesh-network structure. However, for a mesh-network where devices are expected to remain in fixed positions and thus maintain fixed mesh-network connections, a more secure and scalable payment system is possible using the concept of payment channel paths to do multi-hop payments as done in the Lot49 implementation. The Lot49 implementation, however, is mostly concerned with catering for low-bandwidth connections and so a focus is made on decreasing the size of digital signature information that must be communicated between devices. Due to the adjustments required for this, the the Lot49 implementation depends on the Bitcoin community adopting various protocol updates. The Lot49 implementation also makes the process of exchanging data delivery for payment computationally expensive due to the fact they the implementation required a payment process to take place for the transamission and delivery of each individual data packet.

For this project, it is assumed that the devices in the mesh-network communicate through fixed, high-bandwidth Wifi links. As is done in the Lot49 implementation, for each mesh-network connection that is made, a corresponding payment channel will be created and this will allow client nodes to construct payment channel paths to their intended destination nodes. Client nodes can then perform multi-hop payments using the same network path along which their data will be transmitted. An investigation will also be made into ways of enabling multiple packets to be routed through the network for every cycle of payment channel updates so as lower the computational overhead required for the payment feature.

Figure 5.1 shows an illustration of the mesh-network set-up. If node A wanted to

send data to node C and chose route $A - B - C$ to do so then the same route will be used for payments. Therefore, a payment will occur in channel $A - B$ and also in channel $B - C$.

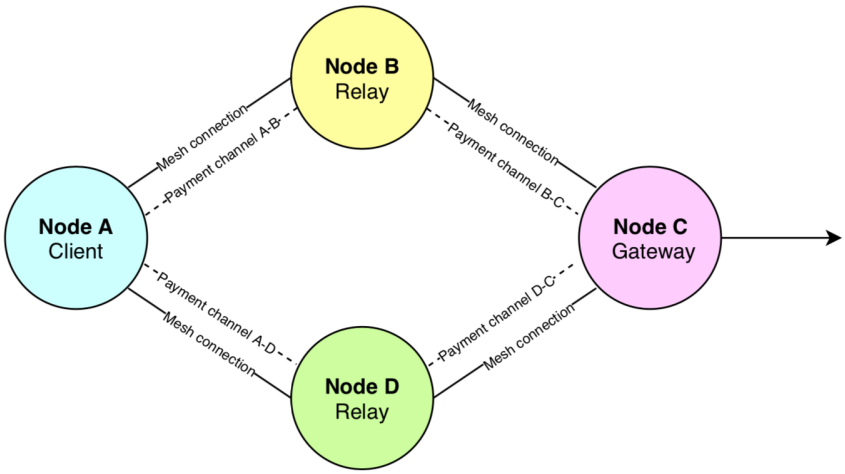


Figure 5.1: Illustration showing mesh network connections and corresponding payment channels

The following sub-systems and sub-processes of the system have been identified:

- **Mesh network:** A basic mesh-network set-up is required in order to implement the system.
- **Bitcoin nodes:** To participate in the mesh-network and to make and receive payments, all nodes must have unique and valid Bitcoin keys and addresses
- **Peer communication:** All devices on the network must store information about all other devices so that data routes and payment routes can be constructed.
- **Encryption:** Client nodes must be able to encrypt data such that only the destination node, node C, can decrypt it.
- **Payment channels:** Nodes must be able to set up payment channels for every mesh-network connection so that multi-hop payment paths can follow the same path as data packets being routed through the network.
- **Multi-hop payments in exchange for data delivery:** Each device must follow the correct procedure so that secure payments can be made for successful data delivery.
- **Transmitting multiple data packets for every payment cycle:** In order to reduce the computational overhead of needing to perform multiple digital signature operations for the process of routing a single packet, the design should include a way of allowing nodes to send and route multiple packets without requiring additional computational overhead.

- **User Interfaces:** User interfaces are required for each device so that a user can interact with the device and also see the states of each of the devices payment channels.

The design of each of the identified sub-systems and sub-processes will be described in this chapter.

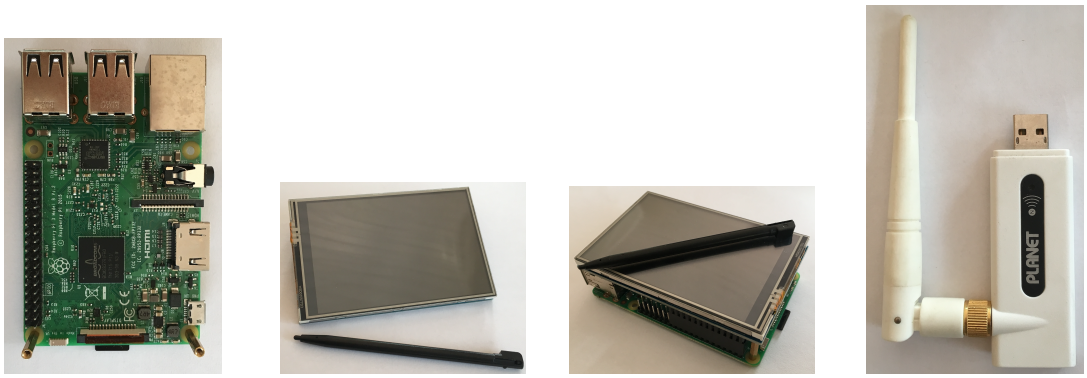
5.1 Wireless Mesh Network

5.1.1 Hardware

To set up a simple mesh network like the one shown in figure 4.1, the following hardware component will be required:

Part Name	Quantity
Raspberry Pi 3B	4
RPi LCD touch screen	4
USB Wireless Adapter	1

The Raspberry Pi's were chosen to represent the IoT devices on the mesh network (nodes A, B, C and D) due to the fact that they are affordable, powerful and have wireless LAN capabilities. One limitation of the Raspberry Pi's is that they have a Maximum Transmission Unit (MTU) size of 1500 bytes [13] which will be a limiting factor for the packet sizes being transmitted through the network. Each Raspberry Pi will be equipped with an LCD touch screen so that user interfaces can be designed for each device allowing the owners of client devices to change data packet sizes, owners of client and relay devices to change their routing fees and for all device owners to be able to view their wallet balances. See figure 5.2a for a picture of a Raspberry Pi 3B, figure 5.2b for a picture of the LCD screen and stylus and figure 5.2c for a picture of the screen attached to the Raspberry Pi. Figure 5.2d shows a USB wireless adapter that will be used for the Raspberry Pi device playing the role of the gateway node (node C) to allow it to connect to both the wireless mesh network as well as connect to the local WiFi network.



(a) Raspberry Pi (b) Screen (c) Raspberry Pi with Screen (d) Wifi Adapter

Figure 5.2: Hardware Components

5.1.2 Software

To create the mesh-network, a protocol called *Better Approach To Mobile Adhoc Networking* (*B.A.T.M.A.N*) will be used. This will allow the devices to wirelessly communicate with each other using TCP sockets. To reconstruct the scenario shown in figure 4.1, node A will have a socket connection with node B and D only and node B and D will only have socket connections with node A and C. Only node C will have access to the internet through the use of the Wifi adapter.

5.2 Bitcoin Nodes

5.2.1 Setting up key pairs and addresses

For an IoT device to be able to send or receive payments in Bitcoin it needs to have a unique private-public key pair as well as a unique Bitcoin address. The public key and address are required so that the device can be paid and a private key is required so that the device can sign a transaction when it pays another device as described in section 2.1.1. In other words the private key will enable the device to claim any transaction that is addressed to its bitcoin address. To create key pairs and an address, each device must supply a secret passphrase. The passphrase is converted into a number which is then used to generate a private key. The private key is multiplied with the Bitcoin generator point, G, to produce the public-key point (see section 2.1.1) which is then compresses to produce a Bitcoin address. See figure 5.3 for an illustration of this process.

Figure 5.3: Creation of Bitcoin private-public key pair and Bitcoin address

Each node will be required to store their key pairs and addresses and will do so by creating an object of type *BTC_node*. Figure 5.4 below shows a class object diagram of the *BTC_node* class.

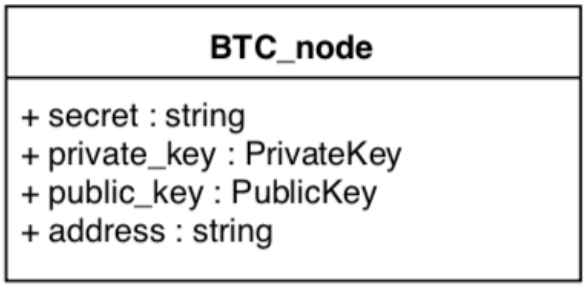


Figure 5.4: Class diagram of the *BTC_node* class

5.2.2 Funding the devices

Once a device has a valid Bitcoin address, it needs funding so that it is able to later fund a payment channel. For this project, the Bitcoin testnet blockchain will be used instead of the Bitcoin mainnet blockchain. The Bitcoins on the testnet blockchain have no value and so the testnet exists purely so that developers can run tests without needing to spend real Bitcoin. Due to the fact that testnet Bitcoins (tBTC) have no value, it is possible to go to a tBTC faucet website such as [3] and request that some tBTC be sent to a specific Bitcoin testnet address. This will create a valid transaction that sends tBTC to the address.

5.2.3 Storing Transactions

The Bitcoin nodes will need to be able to analyse, verify, store, create and sign transactions. As described in section 2.1.2, each Bitcoin transaction is composed of inputs that reference previous transactions and outputs that spend the inputs. Transactions will be stored in objects of type Tx and each will be composed of $TxIn$ and $TxOut$ objects which will define the details of the transaction inputs and outputs respectively. Figure 5.5 is a class diagram that illustrates the structure of the Tx, $TxIn$ and $TxOut$ classes.

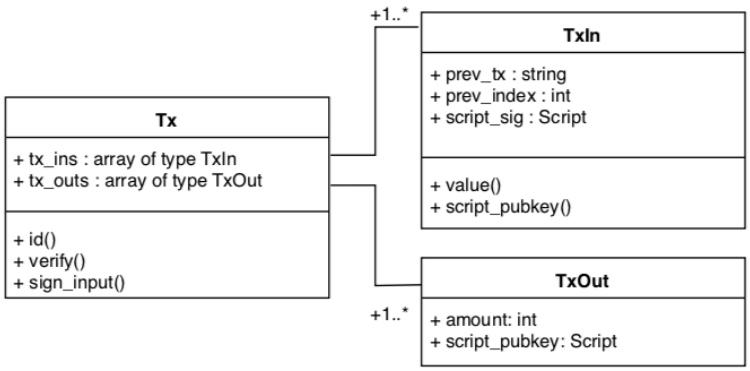


Figure 5.5: Class diagram of the Tx, $TxIn$ and $TxOut$ classes

5.3 Peer Communication

All nodes on the mesh-network need to store information about all other nodes. This is required so that when a node is acting as a client, it is able to construct a route through the network given the payment channels that exist in the network. It is also necessary for client nodes to have access to the fees charged by each node in the network.

To meet these requirements, when a device connects to the mesh network it will be given the details of all the nodes on the network and it will store this information in *Peer* objects. The *Peer* object will be used to store information such as the Bitcoin public key and address of each node in the network along with the shared symmetric key that will be used for encryption with that peer (discussed further in section 5.4). Lastly, if the node that is constructing the *Peer* object has a direct mesh connection with the peer in question then the *Peer* object will also store a *Socket* object that the two nodes will use to communicate with each other. Figure 5.6 below shows a class object diagram of the *Peer* and *Socket* classes.

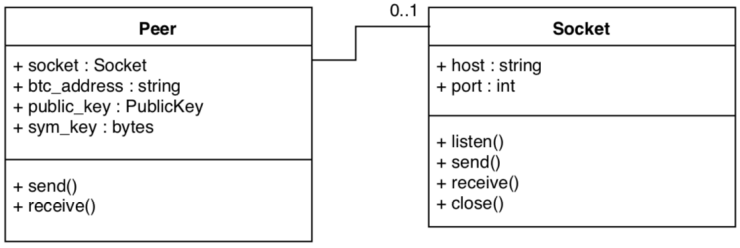


Figure 5.6: Class diagram of the *Peer* and *Socket* classes

The mesh network will be used to share information regarding existing payment channels and current relay fees to all the nodes in the network. Since this information is a necessary requirement for any node who wants to participate in the network, there will be no charge or compensation for advertising and sharing this information through the network. All nodes will be required to help propagate this information

throughout the network if they wish to participate in the normal activities of the mesh-network.

In terms of design, peer information will be broadcast throughout the network using separate socket connections to those used for the transfer of client node data packets.

5.4 Encryption

It is a requirement that client nodes are able to encrypt their data so that only the intended destination node can decrypt it. The encryption that will be used is exclusive-or encryption using shared symmetric keys. This encryption is described in detail in section 2.2. This encryption allows two nodes to create a secret shared symmetric key using the their Bitcoin keys and do so without the need for any direct communication. The client node can then encrypt data using xor-encryption with the symmetric key. The encrypted data can safely be transmitted across the network and no intermediary node will be able to decrypt the data as they do not know the symmetric key required for decryption. Only the destination node will be able to decrypt the payload and will do so using xor-decryption with the symmetric key. Figure 5.7 below illustrates the encryption method described in section 2.2.

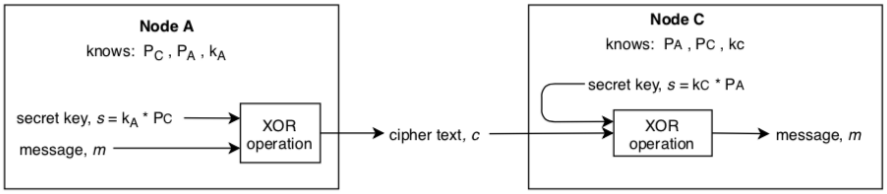


Figure 5.7: Exclusive-Or encryption and decryption between nodes A and C using a secret symmetric key

5.5 Payment channel set-up

Once each node has *Peer* objects to represent all the other nodes and once socket connections between the nodes exist in a way that resembles figure 4.1 then payment channels can be set up. A payment channel will be made for each pair of nodes that are directly connected to each other through a socket connection on the mesh-network as illustrated in figure 5.8. The following payment channels will therefore need to be created:

- Channel A-B
- Channel A-D
- Channel B-C

- Channel D-C

Note that there is no direct payment channel between node A and C. Figure 5.8 illustrates which payment channels are necessary given the mesh-network connections.

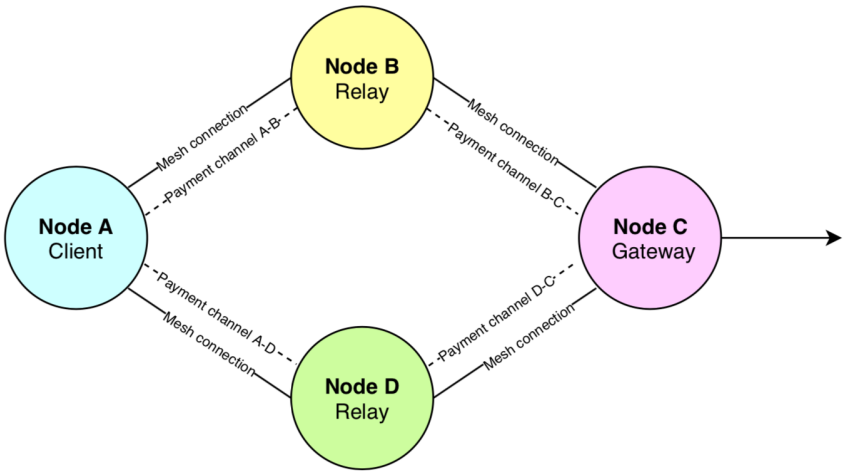


Figure 5.8: Illustration showing mesh network connections and corresponding payment channels

An overview of the concept of a payment channel was given in section 2.1.4. In this section it was shown that payment channels can be created between two parties by both parties funding the channel with separate input transactions. For this project, it is assumed that payments will be uni-directional in that either payments will be made from A to B to C or they will be made from A to D to C. Since payments will be uni-directional, only one party is required to fund the payment channel with an input.

As described in section 2.1.4, the creation of a payment channel requires a funding transaction that must be broadcast to the Bitcoin network and be included in a blockchain block (an on-chain transaction). Any transaction is composed of inputs and outputs. For the situation where node A opens a payment channel with node B, the following input and output is required to make up the payment channel funding transaction:

- **Input:** The input transaction will reference the transaction where node A was sent testnet Bitcoin from a testnet faucet (see section 5.2.2)
- **Output:** The output will be a 2-of-2 multi-sig script that is only spendable when signed by both node A and node B using their respective private-keys.

After constructing the input and output of the funding transaction, node A can create the funding transaction, sign the input of the transaction and create a *Channel*

object. Node A can then send a channel proposal with the details of the channel to node B over a socket connection and node B can analyse the details before also creating a *Channel* object. Once both parties are satisfied with the funding transaction then it can be broadcast to be included in the Bitcoin blockchain. Once the transaction is included in a block on a blockchain (which will take approximately 10 minutes) then both nodes can create *Channel* objects and can start creating off-chain transactions. A *Channel* object will be defined by the Bitcoin addresses of the two nodes using it, the on-chain funding transaction of the channel and the channel state which is the current division of the channel funds between the two nodes. Figure 5.9 below shows a class object diagram of the *Channel* class. See figure 5.10 for an illustration of the funding transaction if node A were to commit 1000 satoshis to the channel and figure 5.11a and 5.11b are object diagrams that show the state of the two *Channel* objects that will be created by node A and B respectfully to represent *channel_AB*.

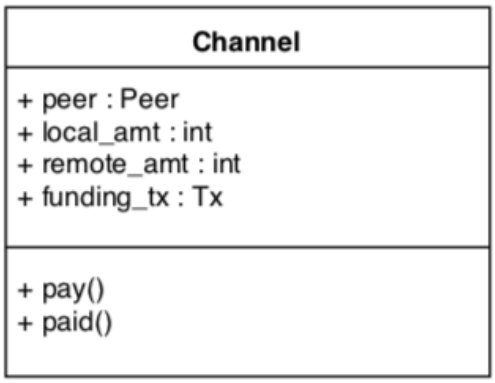


Figure 5.9: Class diagram of the *Channel* class

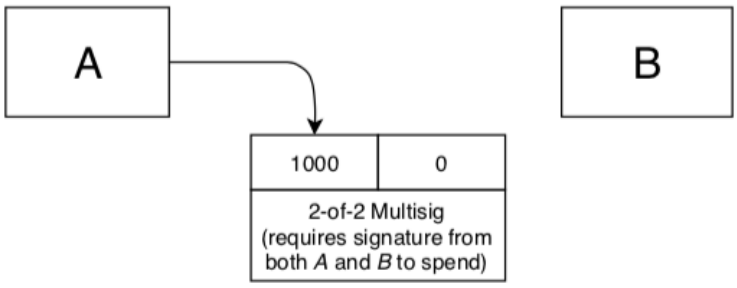
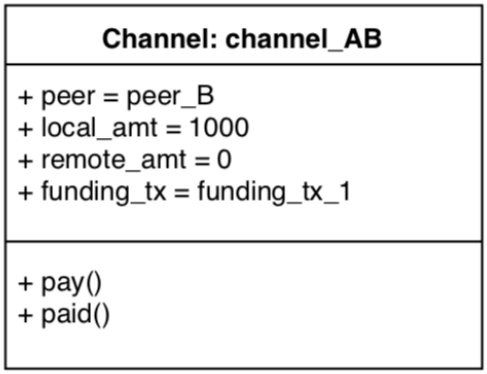
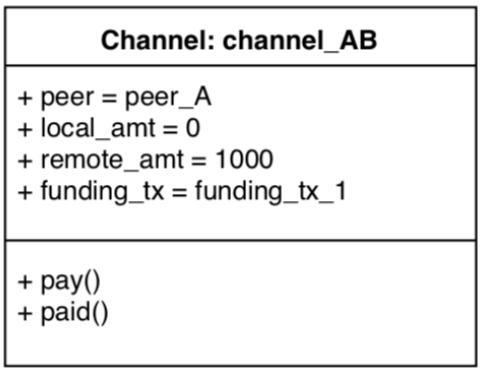


Figure 5.10: Funding transaction for channel A-B



(a) Object diagram of the *Channel* object created by node *A* to represent *channel_AB*



(b) Object diagram of the *Channel* object created by node *B* to represent *channel_AB*

Figure 5.11: Object diagrams of the *Channel* objects created by node *A* and *B* for *channel_AB*

The process of setting up a payment channel between node *A* and node *B* is summarised in the activity diagram shown in figure 5.12.

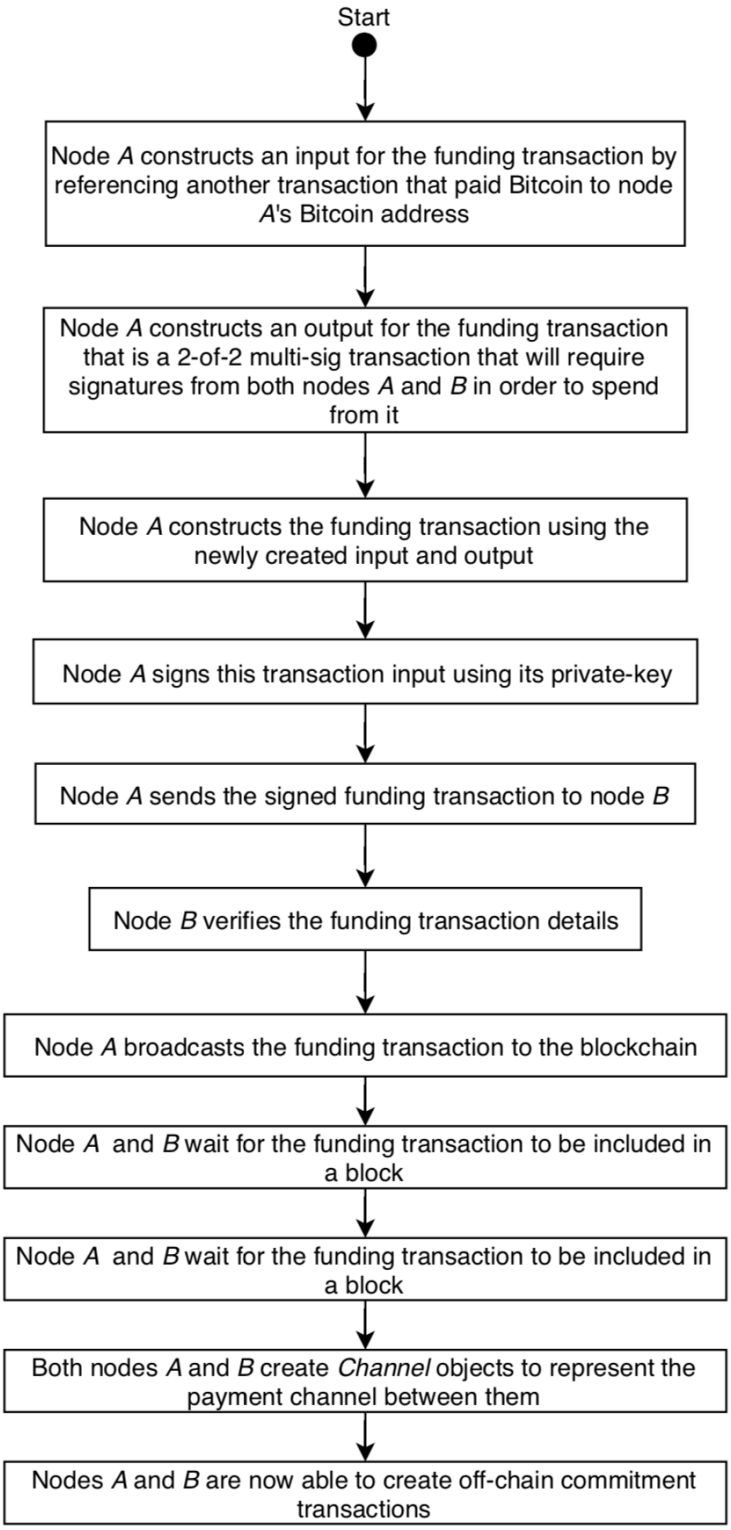


Figure 5.12: Activity diagram showing payment channel creation for Channel *A*-*B*

5.6 Multi-hop payments in exchange for data delivery

5.6.1 Description

Once all the payment channels have been set up, data can be transferred across the network in exchange for payments. In this section, the design of this process will be outlined by describing each step in the scenario where node A wants to send a data packet to node C.

Step 1:

Figure 5.13 below illustrates the initial set-up of payment channels in the mesh network between the various devices. The following points outline the scenario:

- Node A requires a data packet, message m, of 10 bytes to be sent to node C
- Node B is advertising a routing fee of 1 satoshi per byte and both nodes D and C are advertising routing fees of 2 satoshis per byte
- Node A has a choice between route $A-B-C$ which will cost 3 (1+2) sat/byte or route $A-D-C$ which will cost 4 (2+2) sat/byte.
- Node A will chose the cheapest route to node C which in this case is route $A-B-C$. It will cost node A 30 (10*3) satoshis to send the 10 byte data packet along this route.

Figure 5.14 shows the details of the chosen route, $A-B-C$.

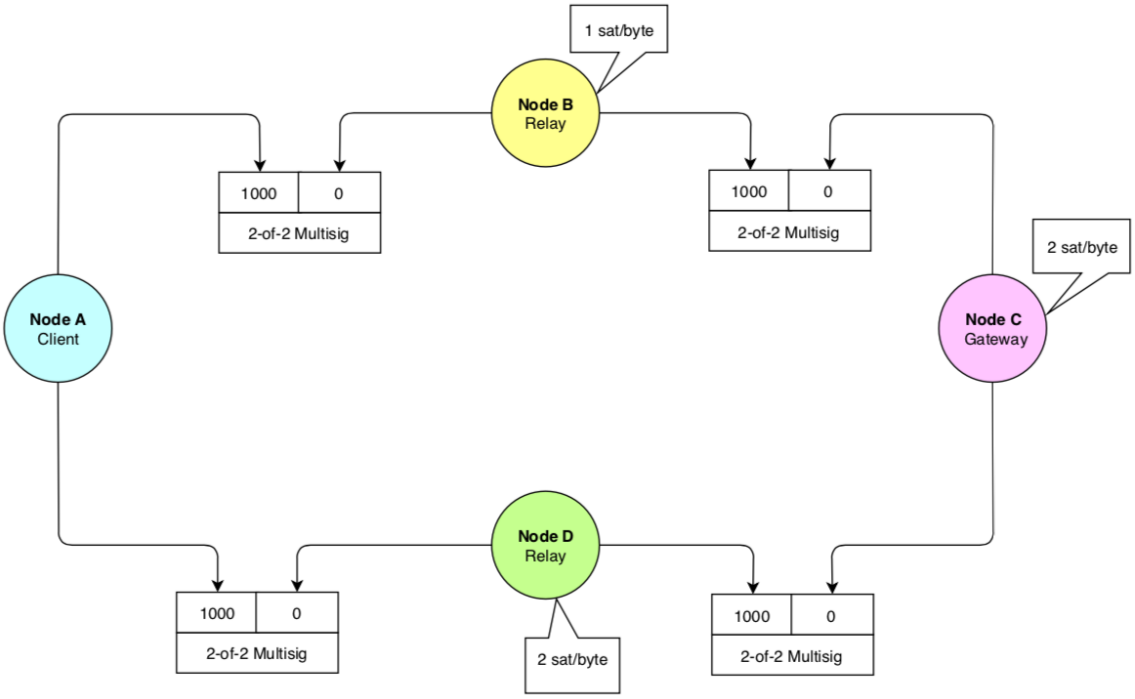


Figure 5.13: Mesh network payment channels initial set-up

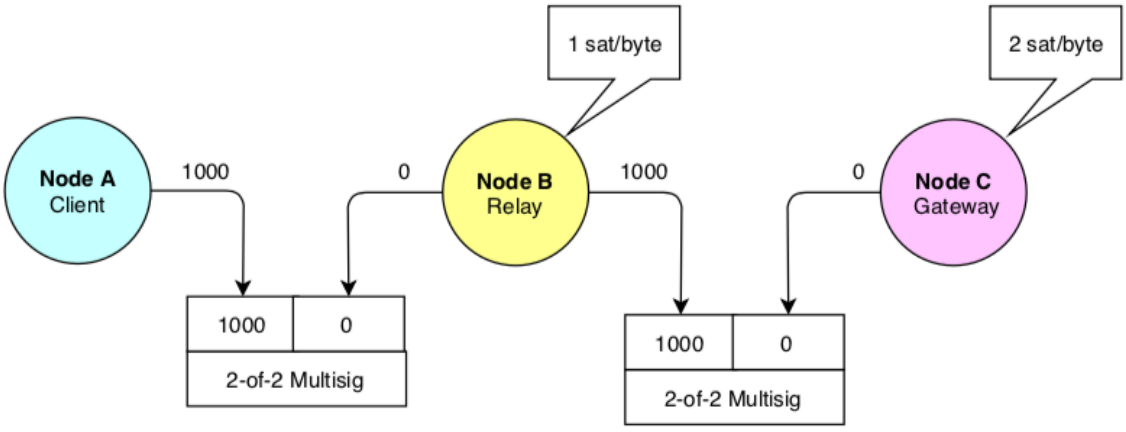


Figure 5.14: Multi-hop payment step 1

Step 2:

Once node A has chosen a route, it will commence with the following actions:

- Node A will generate a random secret, X. Node A will then take the $sha256$ hash (a popular cryptographic hash used in the Bitcoin protocol) of X to produce hash, H:
$$H = sha256(X)$$
- Node A now constructs the body of the payload that it wishes to send as follows:
$$body = \{`message' : m, `secret' : X\}$$

- Node A then encrypts the body of the payload such that only node C will be able to decrypt it. This is done using the secret symmetric key shared by A and C. Figure 5.16 below illustrates this process.
- The commitment transaction that node A will construct for $channel\_AB$ will have the following form (see section 2.1.4 and 2.1.5 for details on commitment transactions):

  - **Input:** The 2-of-2 multi-sig output of the funding transaction
  - **Output:**
    1. 970 satoshis paid to node A's address immediately
    2. 0 satoshis paid to node B's address immediately
    3. 30 satoshis paid to HTLC1. HTLC1 has two clauses and will be paid to whoever is able to satisfy one of the clause conditions first:
       - **Clause 1:** Pays node B if node B is able to produce the pre-image of H (the pre-image if the item that when hashed with the $sha256$ hash algorithm, produces H).
       - **Clause 2:** Refunds node A after time t has passed.

  See figure 5.15 for an illustration of how the commitment transaction proposes to spend the funding transaction of $channel\_AB$.
- Next, node A will construct the header of the payload. The header will consist of the payload source address (which in this case is the address of node A), the route information (this includes the addresses of each node along the route along with the number of satoshis to be paid to them) and a commitment transaction for $channel\_AB$. The header will have the following structure:

  $header = \{`source' = node\_A\_address, 'route' = route\_info, `commitment\_tx' = commitment\ transaction\ for\ channel\_AB\}$
- Node A can now construct the entire payload using the header and encrypted body as shown in figure 5.17. This payload can then be sent to the next node on the chosen route which in this case is node B.

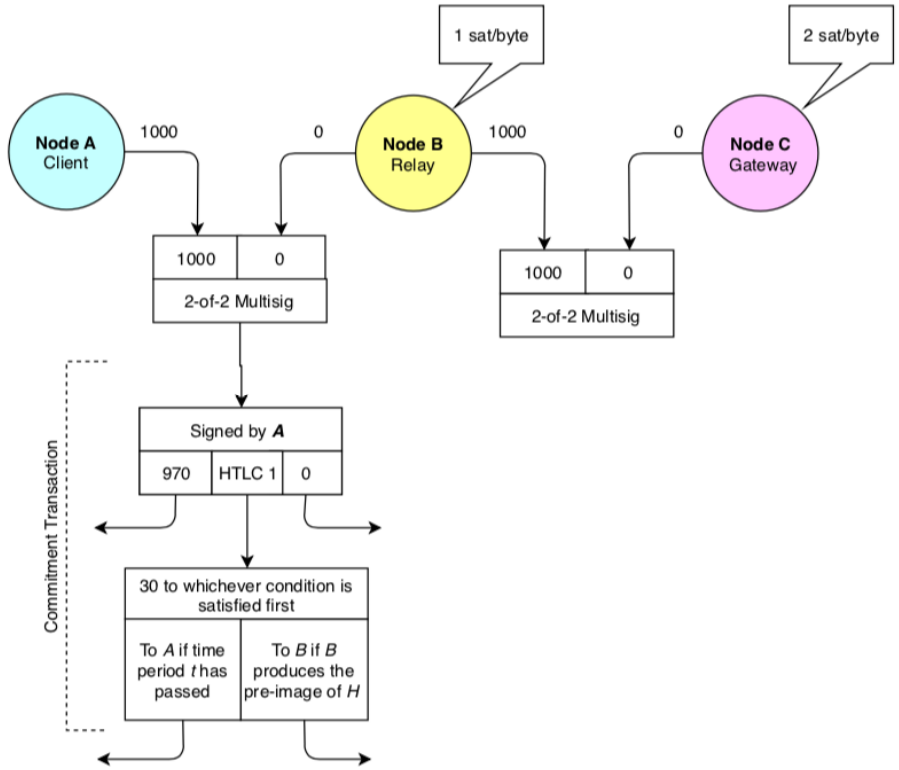


Figure 5.15: Multi-hop payment step 2

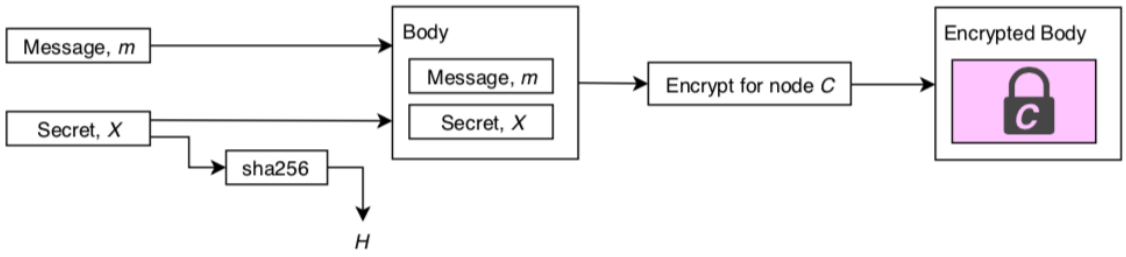


Figure 5.16: Encrypting the body of the payload

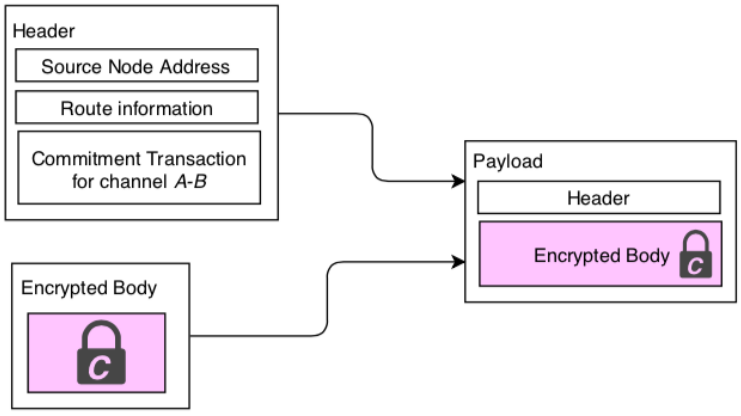


Figure 5.17: Payload construction

Step 3:

Node B then receives the payload from node A and is able to read the header but unable to decrypt the body of the payload since it does not own the necessary shared symmetric key to do so. Node B commences with the following steps:

- Node B can analyse the commitment transaction proposed by node A for $channel\_AB$. It can be seen that if the commitment transaction is ever broadcast to the Bitcoin blockchain, node B would be paid the amount already owed to it (in this case, 0 satoshis) and in addition, if node B is able to supply the pre-image of hash, H, then node B will be paid an additional 30 satoshis (clause 1 of HTLC1). This acts as an incentive for node B to assist in helping to transfer node A's encrypted message.
- Node B then analyses the route information and sees that the next-hop on the route is node C. Thus node B creates a new commitment transaction for $channel\_BC$ with the following input and output:
  - **Input:** The 2-of-2 multi-sig output of the funding transaction
  - **Output:**
    1. 980 satoshis paid to node B's address immediately
    2. 0 satoshis paid to node C's address immediately
    3. 20 satoshis paid to HTLC2. HTLC2 has two clauses and will be paid to whoever is able to satisfy one of the clause conditions first.
       - **Clause 1:** Pays node C if node C is able to produce the pre-image of H (the pre-image is the item that when hashed with the $sha256$ hash algorithm, produces H).
       - **Clause 2:** Refunds node B after time t has passed.

  See figure 5.18 for an illustration of how the commitment transaction proposes to spend the funding transaction of $channel\_BC$.
- Node B is now able to construct an updated header for the payload as shown in figure 5.19. The newly constructed payload is then sent to the next node on the chosen route which in this case is node C.

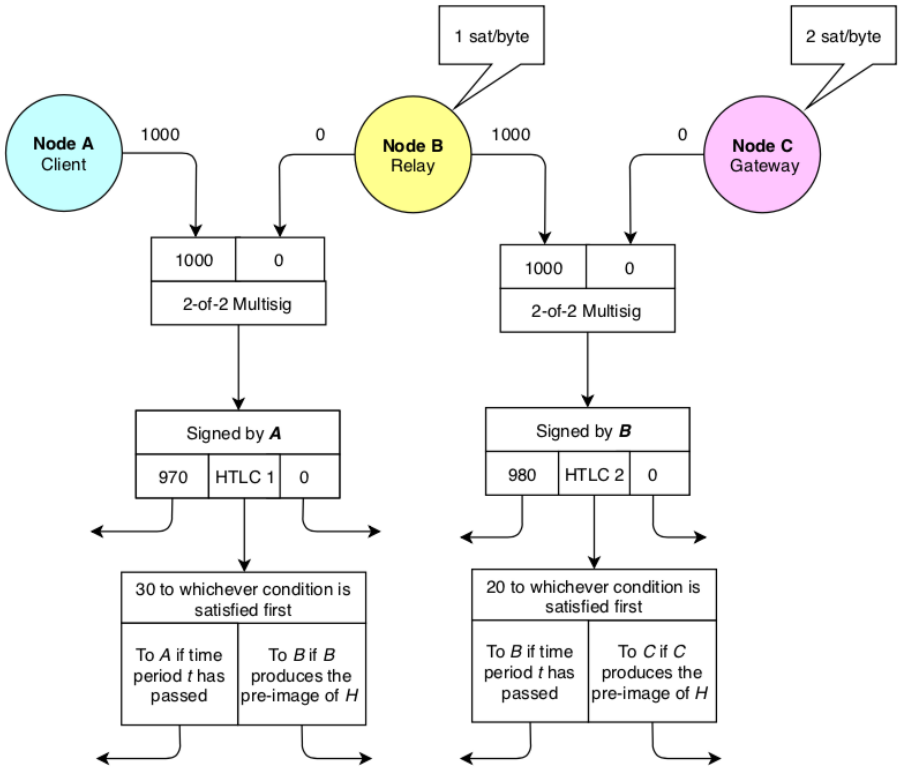


Figure 5.18: Multi-hop payment step 3

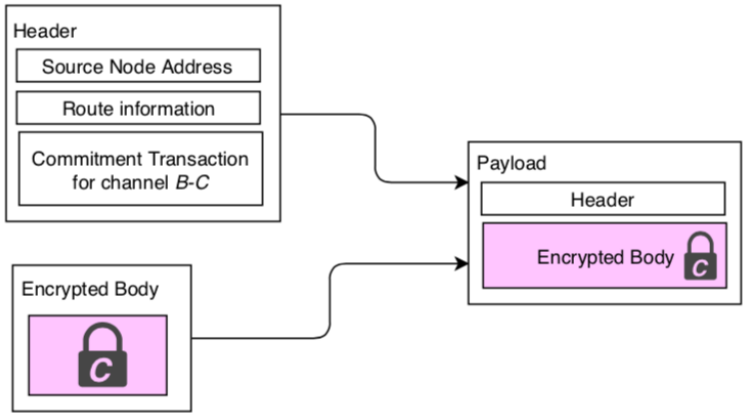


Figure 5.19: Payload construction

Step 4:

Node C receives the payload and commences with the following steps:

- Node C analyses the header of the payload and sees that it is the last node on the route, hence the destination node. It also sees that node A is the source of the payload. Node C then uses the secret symmetric key that it shared with node A to decrypt the body of the payload to reveal message, m, and secret, X. See figure 5.20 for an illustration of this step.

- Node C then analyses the commitment transaction that it received from node B and verifies that it is able to claim clause 1 of HTLC2 and hence gain 20 satoshis if it is able to produce the pre-image of H. Node C is able to do this since it has revealed secret, X, which is the pre-image of H.
- Node C then signs the commitment transaction proposed by node B for $channel\_BC$ and sends it back to node B along with secret, X.
- Node B receives the signed commitment transaction as well as secret X. Node B verifies that X is the pre-image of H. Once verified, node B and node C update the state of their respective channel objects to reflect the payment of 20 satoshis from node B to node C. This is illustrated in figure 5.21.

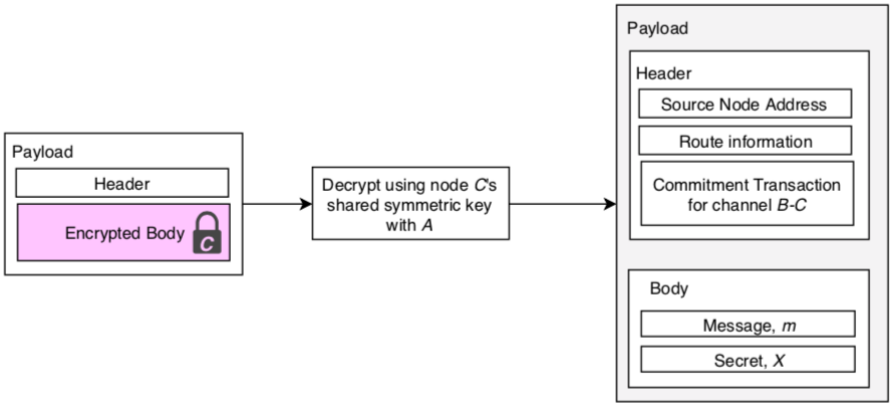


Figure 5.20: Decryption of the payload body

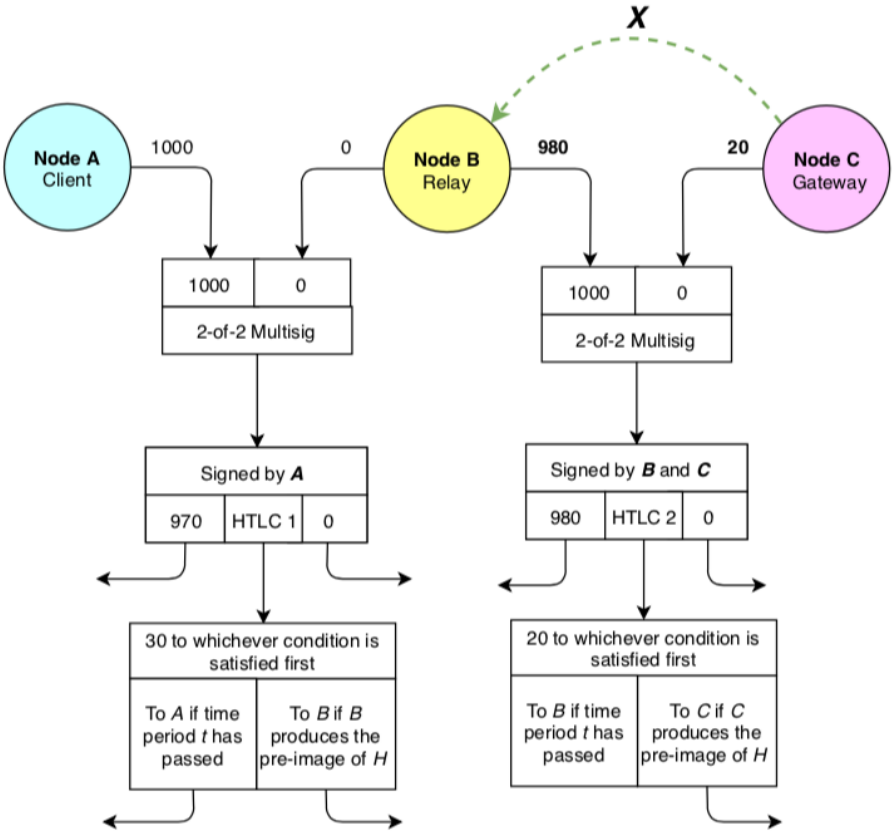


Figure 5.21: Multi-hop payment step 4

Step 5:

- Node B now signs the commitment transaction proposed by node A for $channel\_AB$ and sends the transaction along with secret, X, back to node A.
- Node A receives the signed commitment transaction as well as secret X. Node A verifies that the secret received is the same as what node A originally generated. Once verified, node A and node B update the state of their respective channel objects to reflect the payment of 30 satoshis from node A to node B. This is illustrated in figure 5.22.

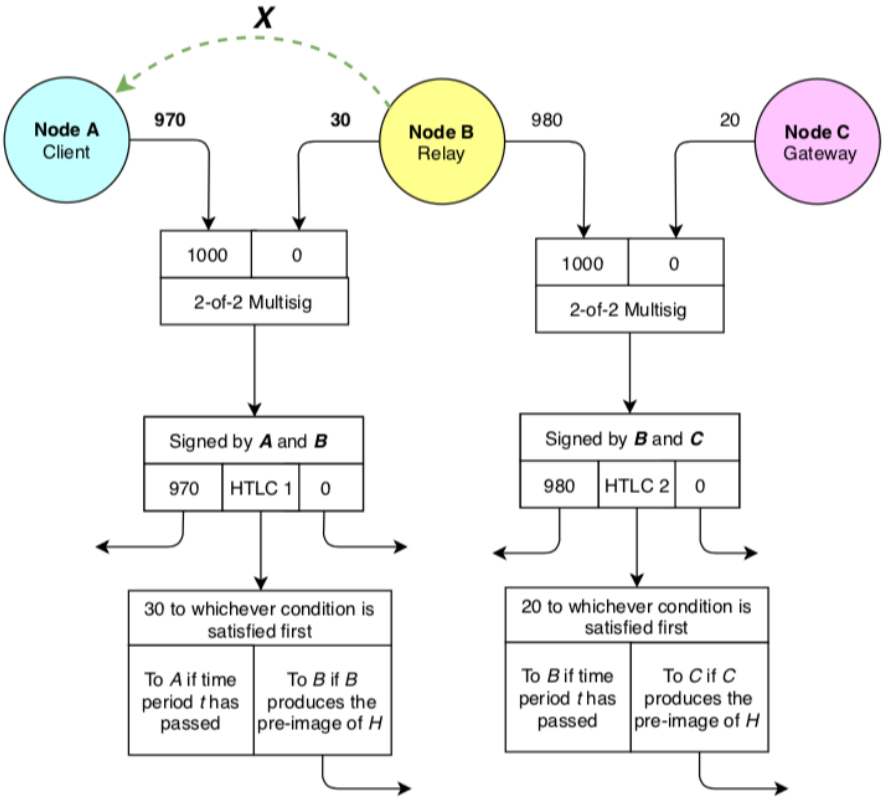


Figure 5.22: Multi-hop payment step 5

Final State:

After step 5 of the process is completed successfully, the following points can be noted:

- Node A has paid a total of 30 satoshis for the services of routing a 10 byte message.
- Node B has paid 20 satoshis and received 30 satoshis and hence made a profit of 10 satoshis for assisting in routing the 10 byte message.
- Node C received 20 satoshis for receiving the 10 byte message and performing a service such as routing the data to an ISP.
- All payments were only made once there was cryptographic proof that the data had been successfully delivered to the intended destination. If any disruption were to have occured that would have prevented the data from being delivered to node C then node B would not have been able to reveal the pre-image of H to node A and so would not have been able to prove to node A that it would be able to claim HTLC1, clause 1. Thus, no channel state updates would have occurred and node A would be left with its original balance.
- The steps described above happen without any need to consult the public blockchain.

5.6.2 Sequence Diagram

To further illustrate the steps required for the process of making multi-hop payments in exchange for data delivery, the sequence diagram in figure 5.23 is used to show the content and sequence of messages passed between nodes. For simplicity, it is assumed that the route chosen is $A - B - C$. Meaning that node A is the client node, node B is a relay node and node C is the destination node.

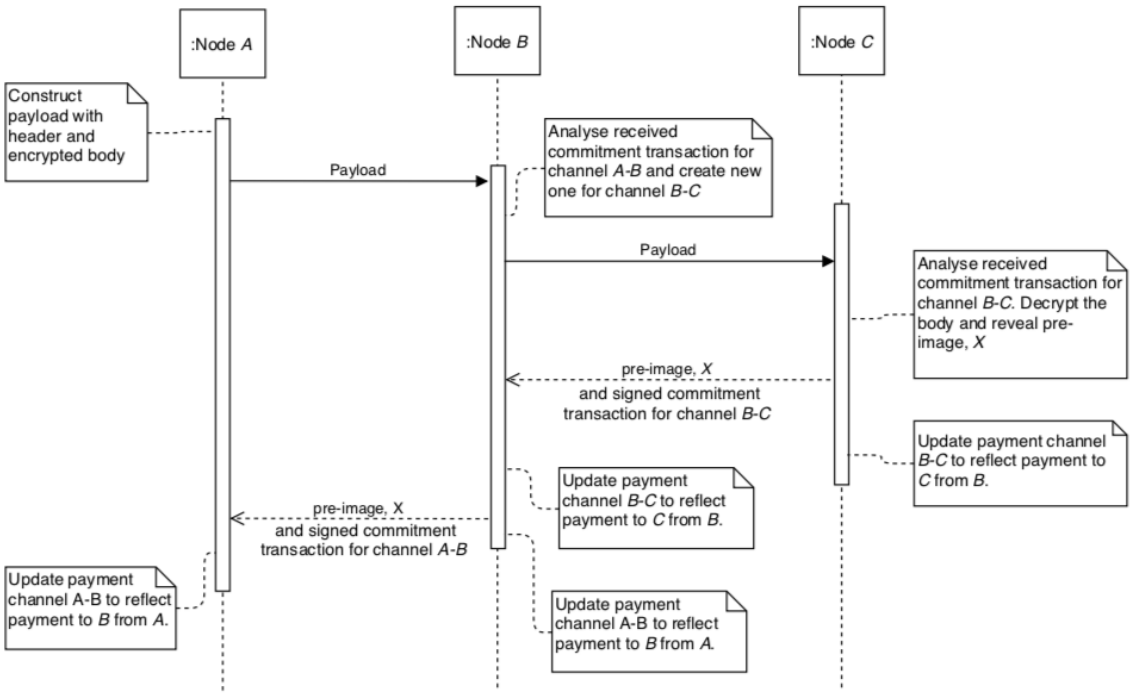


Figure 5.23: Sequence diagram showing messages passed between nodes A, B and C

5.7 Payment for the transfer for multiple data packets

In the previously described sub-system (section 5.6), the process of completing multi-hop payments in exchange for data transmission and delivery is explained. The problem with using this solution alone is that the process of signing transactions is computationally expensive and so to require the multi-hop payment process to be repeated for each data packet that is transmitted will result in slow data transmission.

In this section, a solution to the problem is proposed and the sub-system in section 5.6 is further developed to include this solution.

5.7.1 Description

As described in section 5.6, for multi-hop payments to be made in exchange for proof of packet delivery, the following is required:

- The client node must generate a secret, X, and calculate the hash of this secret to produce the hash, H. The secret, X, is then included in the body of the payload and encrypted such that only the destination node can decrypt it and the hash, H, is included in the HTLC script of the commitment transactions.
- The destination node will then decrypt the body of the payload to reveal X and then pass it back through the route to the client node. This will allow each node to satisfy the appropriate output of the commitment transaction scripts which are only satisfiable if the pre-image of H is produced. The pre-image of H is X.

To enable multiple data packets to be transmitted through a route while only requiring one payment settlement process, the following is required:

- The secret, X, must be split into multiple pieces so that each data packet contains a part of the secret.
- The destination node must only be able to reconstruct the secret, X, if it receives all the individual pieces that were divided amongst the data packets.

Using such a method, relay nodes and the destination node will only be able to claim payment once all the individual data packets have successfully been delivered to the destination node.

The method that will be used for dividing up secret, X, is called Shamir Secret Sharing which is described in detail in section 2.3. Using the Shamir method, X can be divided up in into n number of pieces called shares. Each share can be included into a payload body along with a message and the body can then be encrypted for the destination node. This process is illustrated in figure 5.24 and can be contrasted with figure 5.16 in section 5.6.

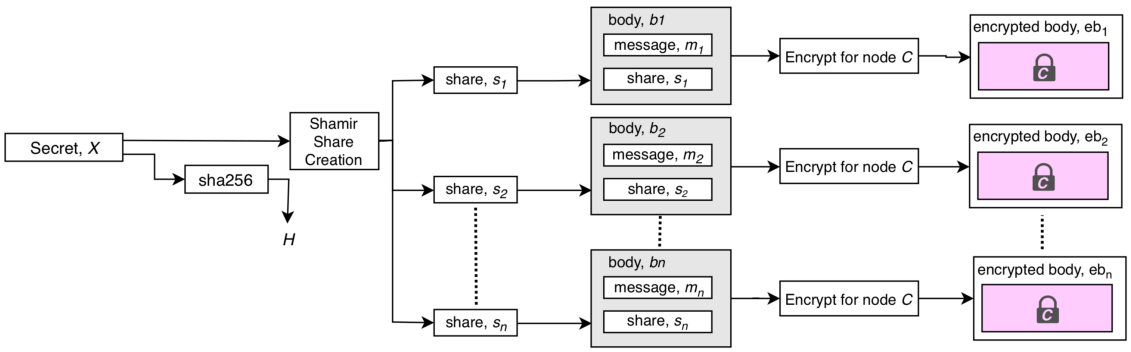


Figure 5.24: Construction of Shamir shares and the encrypted bodies

To reconstruct X, the destination node must receive and decrypt all the payload bodies and hence acquire all the Shamir shares. Only if the destination node receives all the shares will it be able to reconstruct the secret, X, and start the process of passing X back through the route so that payments can be settled. The process of reconstructing the secret is illustrated in figure 5.25 and can be contrasted with figure 5.20 in section 5.6.

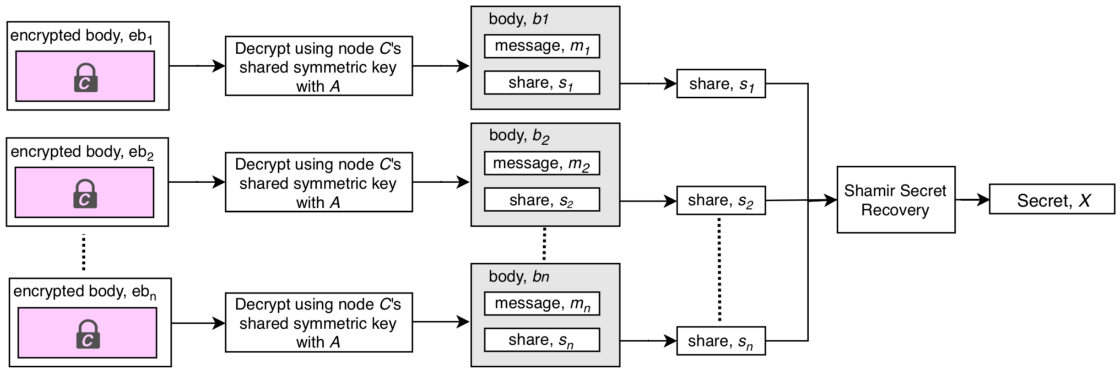


Figure 5.25: Decryption of the bodies and reconstruction of the Shamir secret

5.7.2 Sequence Diagram

To further illustrate the process described above, a sequence diagram is used. The sequence diagram shown in figure 5.26 shows the type and sequence of messages passes between nodes in the case where the chosen route is $A-B-C$. The sequence diagram in figure 5.26 can be compared to that in figure 5.23. In figure 5.23, only one payload consisting of one header and one body containing one encrypted secret is passed between nodes. Whereas in figure 5.26 below, one header is passed between nodes followed by n different bodies each containing a different share of secret, X. It should be noted that the number of commitment transactions being created and signed for the process has remained the same.

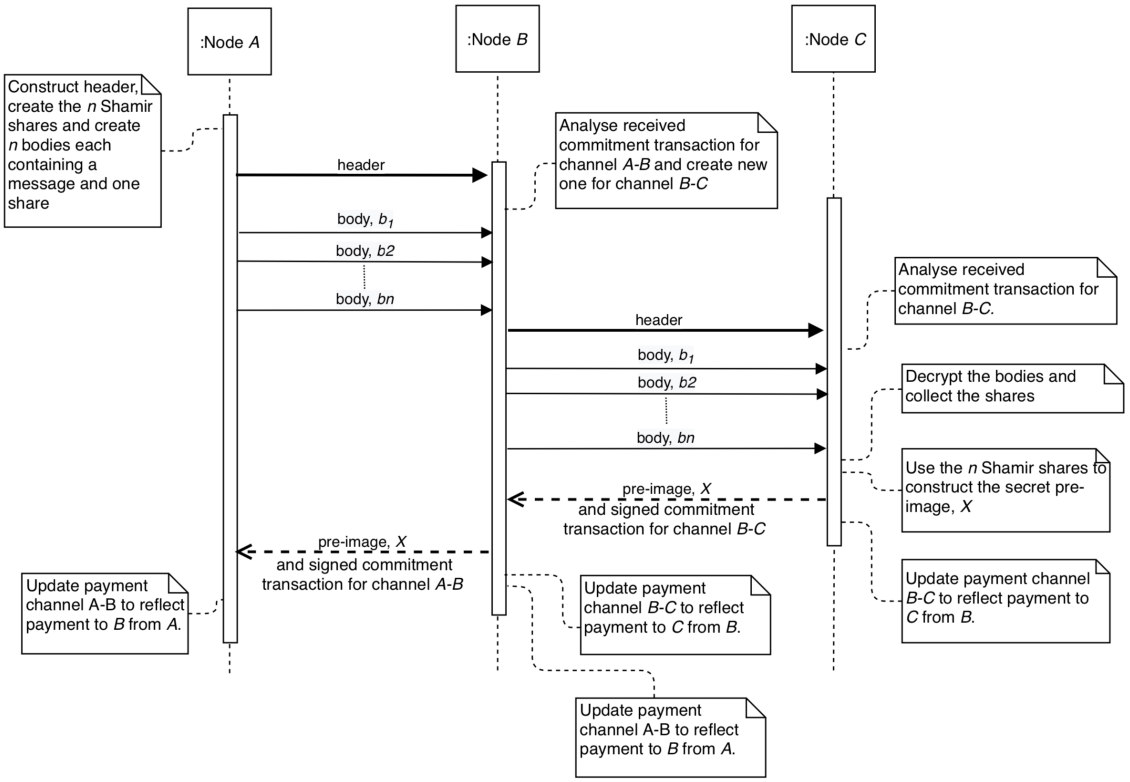


Figure 5.26: Sequence diagram showing messages passed between nodes A, B and C

5.8 User Interfaces

From the user requirements it was identified that client node A should choose a route depending on the total price that it would cost to send its data along each route. It is also required that node A is charged proportionally to the size of the data packet it wants to send along the route. The following elements have been identified as requirements of each nodes user interface:

- A node in client mode, such as node A, must give the user the ability to increase and decrease the size of the packets that will be transferred across the network as well as the ability to choose the number of packets to be transmitted for every payment.
- Nodes in relay mode, such as nodes B and D, and nodes acting as gateway nodes, like node C, must give the user the ability to increase and decrease the fee that they wish to charge for assisting in the successful delivery of data.
- All node should display the local balances of each of the payment channels that they are apart of.
- All nodes should display their total wallet balances which will be the sum of their channel balances.

Figure 5.27 below shows the user interface design for client node A. The display shows node A's local channel balances for $channel\_AB$ and $channel\_AD$ along with node A's total wallet balance. The display also shows that 10 packets of 1 byte each (a total of 10 bytes) will be transmitted and paid for at a time. The up and down arrows give the device owner an option to increase or decrease the number of packets and the packet sizes of the packets to be transmitted for every payment cycle.

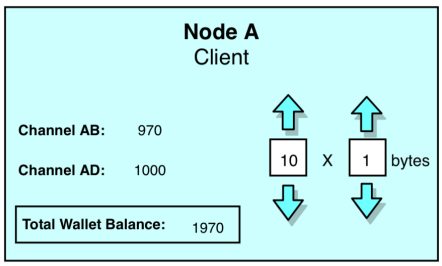
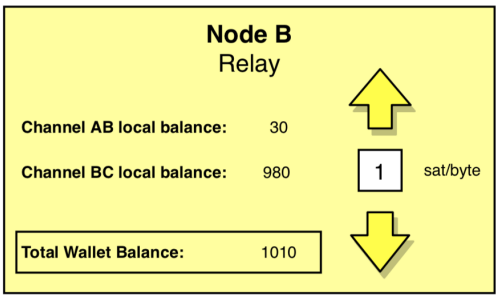
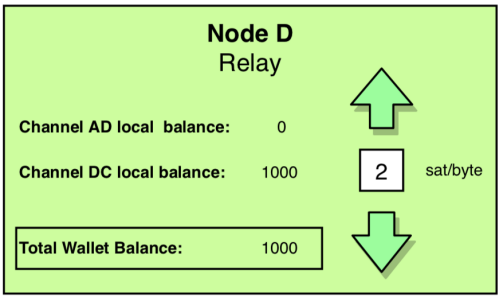


Figure 5.27: User Interface design for client node A

Figures 5.28a, 5.28b and 5.29 below show the user interface designs for relay nodes B and D and for gateway node C respectively. The displays show the channel balances for each node along with the total wallet balances for each node. The up and down arrows give the device owners an option to change the fee that they wish to charge using the units: satoshis per byte.



(a) UI design for relay node B



(b) UI design for relay node D

Figure 5.28: User interface designs for relay nodes B and D

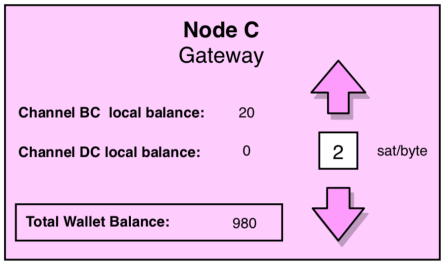


Figure 5.29: User Interface design for gateway node C

Chapter 6

Implementation

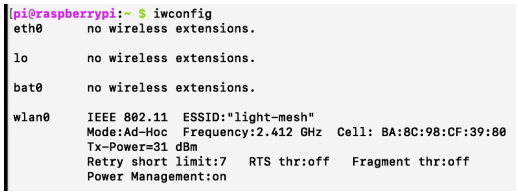
In chapter 5 the designs of the various sub-systems were outlined. In this chapter the implementation of each of the sub-system designs is described.

6.1 Wireless Mesh Network

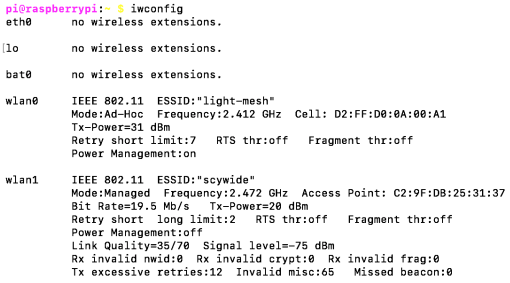
The Raspberry Pi's were all configured to connect to the same wireless network called "light-mesh" using the *B.A.T.M.A.N* protocol. Figure 6.1 below shows a picture of the entire set-up.

Figure 6.1: Raspberry Pi Mesh Network set-up

Once the mesh-network set-up is complete, all the nodes are on the same network called "light-mesh". This can be seen by using the "$iwconfig$" command which shows the devices wireless network interface details. The output of this command for nodes A, B and D can be seen in figure 6.2b. From this output, it is clear that the only connection that these devices have is to the wireless mesh network called "light-mesh". The output for the "$iwconfig$" command on device C can be seen in figure 6.2a. This figure also shows that node C is connected to the mesh network on the $wlan0$ interface but that it is also connected to the local WiFi network called "scywide" on the $wlan1$ interface.



(a) Output for nodes A, B and D



(b) Output for node C

Figure 6.2: *iwconfig* outputs

Figures 6.3 show the output of a device on the network when it is prompted to show the details of any neighboring devices on the same mesh network. From the figure it is clear that the device is aware of all the other three devices.

Figure 6.3: Screen shot showing that each device on the network can see all neighbours

Once all the devices are connected to the mesh network, they are all given a static IP address and so any communication between the devices can now be done using socket connections.

6.2 Bitcoin Nodes

6.2.1 Setting up private-public key pairs and addresses

Code listing 6.1 below shows the Python class used to create and store the Bitcoin wallet details of a device. Given a secret passphrase, the private key an be generated. Then using the private key the public key can be generated (see asymmetric property in section 2.1.1). The public key is an elliptic curve coordinate and must be compressed to create a Bitcoin address. The public key and Bitcoin address are safe to share but the private key is not. See the GitHub repository [14] for this project to see the details of the various classes used in the listing below (see specifically folder Final_Year_Project/Code/lightning/modules).

```
1 class BTC_node:
2   def __init__(self, passphrase):
3     self.secret = little_endian_to_int(hash256(passphrase))
4     self.private_key = PrivateKey(self.secret)
5     self.public_key = self.private_key.point
```

```
6       self.address = self.public_key.address(testnet=True)
```

Listing 6.1: *BTC_node* class

Code listing 6.2 below shows an example line of code that would be used by node A to create the necessary Bitcoin wallet details and does so using the *BTC_node* class shown in listing 6.1. In this example, node A uses the passphrase "nodeA".

```
1 node = BTC_node(b'nodeA')
```

Listing 6.2: *BTC_node* class example usage

The resulting Bitcoin address for node A, in base58 encoding, created by the code above is:

$$mst8broiaX4PFMFNbjfrBnMSnrVF42Jgd7$$

This process of creating a unique Bitcoin private-public key pair and address is repeated for each of the four devices used for the mesh implementation.

6.2.2 Funding the device

Once each device has a valid Bitcoin address, they can be funded using a Bitcoin testnet faucet website. For convenience, each payment channel will be funded by referencing a separate testnet faucet transaction. The following details determine how many testnet faucet payments to make to each node:

- Node A must fund two payment channels (*channel_AB* and *channel_AD*) and therefore requires two payments to be made to its Bitcoin address.
- Node B must fund one payment channel (*channel_BC*) and so requires one payment to be made to its Bitcoin address.
- Node C must fund one payment channel (*channel_DC*) and so required one payment to be made to its Bitcoin address.

For each of the payment requests made to the Bitcoin testnet faucet website [15] an amount of 0.009 tBTC (or 900000 satoshi) was requested and after transaction fees were deducted, each transaction funded the given address with 899999 satoshis.

Figure 6.4 below shows the details of a transaction that sends 0.00899999 tBTC (or 899999 satoshis) to the address created for node A (see section 2.1.2 for the details on inputs and outputs of transactions). This transaction is valid and has been broadcast to the Bitcoin testnet blockchain and any one can now verify that the owner of the private key that created the bitcoin address $mst8broiaX4PFMFNbjfrBnMSnrVF42Jgd7$, now has access to 899999 satoshis to spend (this transaction can be viewed at [16]).

Node A will need the transaction ID of this transaction as well as the relevant transaction output index of this transaction later on when it is used to create a funding transaction of a payment channel. The transaction ID and index of this transaction is as follows (tx_id:tx_index):

$$15fccae87a15395af0232ba7e1a5659a6d3ca67c90ebdf900025753fb6a57f3e : 0$$

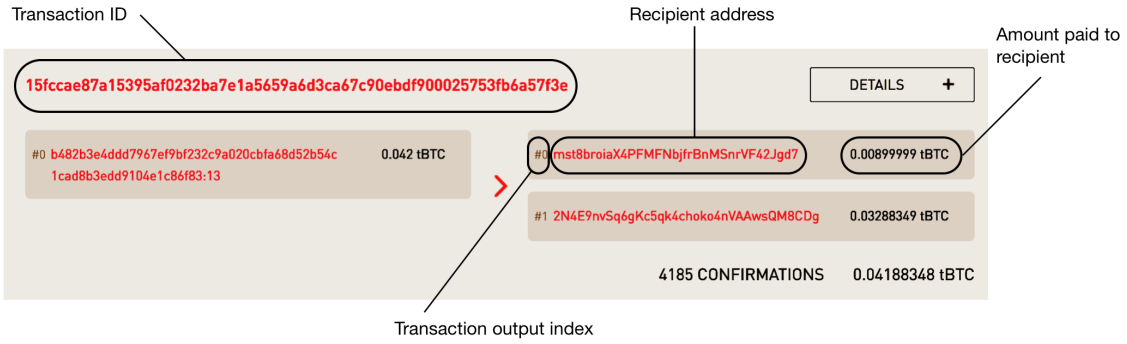


Figure 6.4: Bitcoin Transaction funding node A's Bitcoin address [16]

The process of creating addresses and funding the address is repeated for all the nodes that require funding in the mesh network. They are now all able to fund the necessary payment channels.

6.3 Peer Communication

Each node in the mesh network is aware of each other node and must store information about each node. This is done using the *Peer* class which is shown in code listing 6.3.

```
class Peer:
  def __init__(self, socket, btc_addr, public_key, sym_key):
    self.btc_addr = btc_addr
    self.public_key = public_key
    self.sym_key = sym_key
    self.socket = socket

  def send(self,data):
      self.socket.send(data)

  def receive(self,size=1024):
      return self.socket.receive(size)

  def __str__(self):
    return str(self.socket)+", peer btc address: "+str(self.
   btc_addr)
```

Listing 6.3: *Peer* class

As can be seen in the above listing, each *Peer* is defined by a Bitcoin address (*btc_addr*), a public key (*public_key*), a secret symmetric key (*sym_key*) and possibly a socket connection (*socket*) if the peer node happens to also be directly linked to the current node through a direct mesh network connection. The shared secret symmetric key is important because it will be used when two nodes want to encrypt and decrypt messages. The symmetric key is created using the method described in section 2.2. If, for example, node A wants to calculate the symmetric key that it should use when communicating with node B then it will calculate it as follows:

```
symmetric_key_AB = private_key_A * public_key_B
```

Listing 6.4: Calculation of the symmetric key

Nodes create *Peer* objects for each other node. For example, node A creates a *Peer* object for node B as follows:

```
peer_B = Peer(socket_B, node_B_btc_addr, node_B_public_key,
    symmetric_key_AB)
```

Listing 6.5: *Peer* object creation with socket connection

Node A will repeat the above process to create peer objects for node D but will create a peer object for node C's information as follows:

```
peer_C = Peer(\textbf{None}, node_C_btc_addr, node_C_public_key,
    node_C_sym_key)
```

Listing 6.6: *Peer* object creation without socket connection

The difference with the *Peer* object for node C is that there is no socket connection between node A and C and therefore no direct communication can happen between them.

Once the a *Peer* object exists for node B, node A can encrypt a message for node B using XOR encryption (see section 2.2.2). Code listing 6.7 shows the code for the XOR encryption and decryption. The *xor* function expands or shrinks the key to be the same length as the data and then the *xor* operation is performed. Both the *encrypt* and *decrypt* functions then make use of the *xor* funciton.

```
def xor(var, key):
  while(len(key)<len(var)):
    key += key
  key = key[:len(var)]
  int_var = int.from_bytes(var, sys.byteorder)
  int_key = int.from_bytes(key, sys.byteorder)
  int_enc = int_var ^ int_key
  return int_enc.to_bytes(len(var), sys.byteorder)

def encrypt(message, key):
  return xor(message, key)

```

```
13 def decrypt(message, key):
14   return xor(message, key)
```

Listing 6.7: *Peer* class

Using the code shown above, node A can encrypt and decrypt messages from any other node, such as node B, as follows:

```
1 cipher_text = encrypt(message, peer_B.sym_key)
2 message = decrypt(cipher_text, peer_B.sym_key)
```

Listing 6.8: Encryption and decryption example

6.4 Payment Channels

Code listing 6.9 below shows the *Channel* class that will be used by the devices to set up all their necessary channels. As can be seen in the listing, for the class to be instantiated it must be given a *Peer* object corresponding to the node that the payment channel will be opened with. The initial local and remote balances must be given along with a reference to the funding transaction object used for the channel. Both parties of a channel will need to define separate *Channel* objects and will need to update the state of the channel (the local and remote balances) accordingly using the *pay* and *paid* functions.

```
1  class Channel:
2    def __init__(self, peer, local_amt, remote_amt, funding_tx):
3      self.peer = peer
4      self.local_amt = local_amt
5      self.remote_amt = remote_amt
6      self.funding_tx = funding_tx
7
8    def pay(self, amt):
9      self.local_amt -=amt
10     self.remote_amt +=amt
11
12   def paid(self, amt):
13     self.local_amt +=amt
14     self.remote_amt -=amt
```

Listing 6.9: *Channel* class

Code listing 6.10 below shows pseudo code of a method called *add_channel* that makes use of the *Channel* class above to create a *Channel* object given the two parties partaking in the channel as well as the details of the transaction that is to be used as an input to the funding transaction. This method will be used to construct the input and output of the funding transaction. A funding transaction object will then be created and used to create and return a new *Channel* object. The details of this function will be described with an example.

```
def add_channel(local_node, remote_peer, input_tx_id,
    input_tx_index):

  # Construct the input
  tx_in = TxIn(input_tx_id, input_tx_index)

  #determine initial state of the channel (division of funds)
  local_amount = tx_in.value()
  remote_amount = 0

  # Construct 2-of-2 multi-sig script
  scriptPubKey = Script([2, local_node.public_key, remote_peer.
    public_key, 2, 'check_multisig)'])

  # Construct the output
  tx_out = TxOut(amount = local_amount, script_pubkey =
    scriptPubKey)

  # Construct the transaction object
  funding_tx = Tx([tx_in], [tx_out])

  # Sign the input
  funding_tx.sign_input(0, local_node.private_key)

  # Create new channel object
  new_channel = Channel(remote_peer, local_amount, remote_amount,
    funding_tx)

  # Send channel request to remote peer
  remote_peer.send(new_channel)

  return new_channel
```

Listing 6.10: Function for creating a new *Channel* object

When node A opens a channel with node B, then the following code in listing 6.11 is used. Line 1 and 2 define the transaction input and index of the transaction that was used to initially fund node A as shown in section 6.2.2. Line 4 shows how the *add_channel* function is then used to create a new *Channel* object.

```
tx_in_id = 15
    fccae87a15395af0232ba7e1a5659a6d3ca67c90ebdf900025753fb6a57f3e
tx_in_index = 0

channel_AB = add_channel(node, peer_B, tx_in_id, tx_in_index)
```

Listing 6.11: Calling the *add_channel* function

When the *add_channel* function is called with the parameters specified in listing 6.10, the following will happen in the *add_channel* function shown in listing 6.10:

1. **Line 4:** A transaction input object ($TxIn$) is defined using the given transaction ID and index.
2. **Lines 7 - 8:** The *local_amount* variable is set to the value of the input transaction which in this case would be a value of 899999 satoshis. The channel

is only funded by node A and so node B's initial balance (the remote balance) is set to 0 satoshi.

3. **Line 11:** A 2-of-2 multi-sig Script object is created. The Script includes both node A's public key ($local\_node.public\_key$) and node B's public key ($remote\_peer.public\_key$).

4. **Line 14:** A transaction output object ($TxOut$) is constructed with the multi-sig Script and the initial capacity of the channel (this is equivalent to $local\_amount$ in this case since only node A funds the channel).

5. **Line 17:** The newly created transaction input and output objects are used to construct a new funding transaction ($funding\_tx$).

6. **Line 20:** Node A signs the funding transaction using its secret private key.

7. **Line 23:** A new *Channel* object is created.

8. **Line 26:** Node A sends the new *Channel* object as a channel request to node B.

Node B uses the function shown in listing 6.12 to receive (line 3) and verify (line 6) the channel request. If it is able to verify the funding transaction (by ensuring that the input and output of the transaction are correct and that the inputs are validly signed) then it creates a new *Channel* object to represent a payment channel with node A. In the specific example being used, node B will set the local balance to 0 satoshis and the remote balance to 899999 satoshis.

```
1  def listen_for_channel_request(peer):
2
3    request = peer.receive()
4    funding_tx = Tx.parse(request['funding_tx'])
5
6    if(funding_tx.verify()):
7      new_channel = Channel(peer, request['remote_amt'], request['
       local_amt'], funding_tx)
8      return new_channel
9    else:
10     print("Invalid Channel Request")
11     return None
```

Listing 6.12: Function $listen\_for\_channel\_request$

Code listing 6.13 shows the code that node B would use to call the $listen\_for\_channel\_request$ function in order to create $channel\_AB$.

```
1  channel_AB = listen_for_channel_request(peer_A)
```

Listing 6.13: Code used by node B to add a new channel

6.5 Multi-hop payments in exchange for data delivery

To implement the mutli-hop payment process, separate programs have been developed for each different node role (client, relay and gateway). In this section, the design of each of these programs will be described using activity diagrams and pseudo-code. For simplicity, it is assumed that the route chosen is $A - B - C$. Meaning that node A is the client node, node B is a relay node and node C is the destination (or gateway) node.

6.5.1 Client Node

Activity Diagram

The activity diagram in figure 6.5 shows the structure of the program running on a node in client mode (in this case node A). As shown in the diagram, node A first sets up payment channels with nodes B and D and then begins the process of constructing the payload that it will send to the next node on the chosen path, node B. Note that if node D advertises a lower relay fee than node B then node A will choose path $A - D - C$ in which case the steps would be the same as in the diagram shown below except that node D would replace node B.

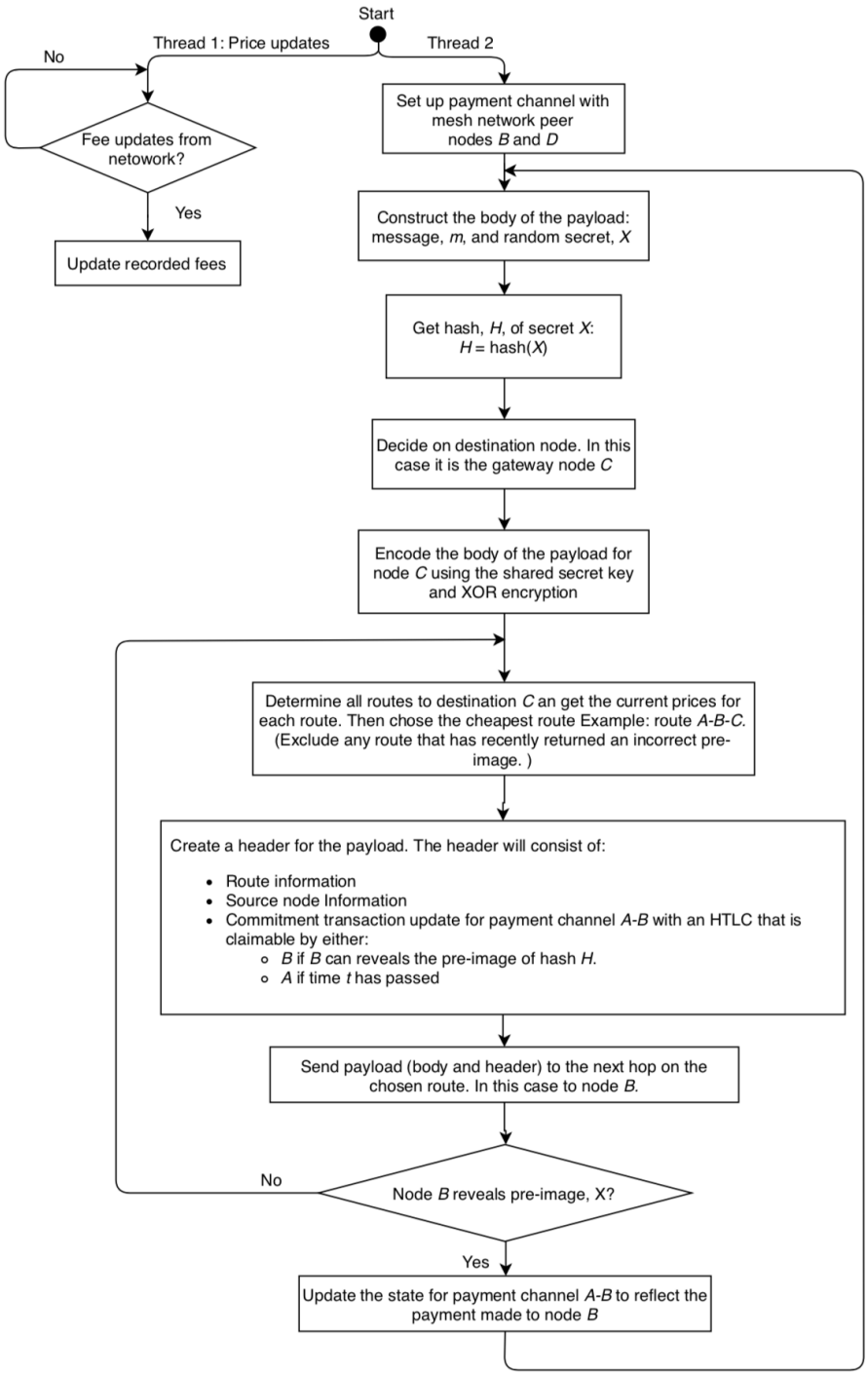


Figure 6.5: Activity diagram for a client node such as node A

Pseudo-Code

The activity diagram in figure 6.5 shows the structure of the program used by node A. Various sections of the code used for this program will be shown in pseudo-code and described.

Listing 6.14 below shows how the body of the payload is constructed. A library called *secrets* is used to create a random string of 16 bytes for secret X. For testing

purposes, the same function is used to determine the payload message, m, and the size of the message can be set by the user. The *sha*256 cryptographic hash function is used in line 3 to create the hash, H, which will later be used in the commitment transactions. The code then goes on to get the secret symmetric key that is shared between node A and node C (lines 9-10). This key is used to encrypt the body in line 13.

```
1  #body: secret and actual message
2  X = secrets.token_urlsafe(16)
3  H = sha265(X)
4  m = secrets.token_urlsafe(packet_size)
5  body = {"secret":X, "message":m}
6
7
8  # get details of node C
9  destination = 'n1weDdde5xXLfPeutESLaG8swr5jLCqz72'
10 sym_key_dest = get_peer(peers, destination).sym_key
11
12 # encrypt the body for node C
13 encrypted_body = encrypt(body, sym_key_dest)
```

Listing 6.14: Payload body construction and encryption

The routes that exist in the mesh-network along with the fees charged by each node will be propagated throughout the network and each node will store the information in a variable called *routes*. The code in listing 6.15 shows how the *routes* variable is used to determine the cheapest route to the given destination. The *cost* of the service will then be calculated based on the chosen route and the size of the message (line 6). Lines 9-10 show how the chosen route is used to identify the *Peer* and *Channel* objects associated with the node that will be the next-hop for the payload on the chosen route. Line 13 shows how node A calls the function *new_commitment_tx* to construct a new commitment transaction for *channel_AB*. This function will be described next.

```
1  # Find cost of each route and choose cheapest
2  cheap_route_index = find_cheapest_route(routes, destination)
3  cheapest_route = routes[cheap_route_index]
4
5  # cost of service given chosen route and message size
6  cost = route_cost(cheapest_route, len(m))
7
8  # get next hop from route and hence get relevent channel
9  next_hop = get_peer(peers, cheapest_route[0])
10 next_hop_channel = get_channel(next_hop, channels)
11
12 # create commitment transaction
13 commitment_tx = new_commitment_tx(node, next_hop_channel, cost, H)
```

Listing 6.15: Getting route and cost information

Listing 6.16 shows the pseudo-code function definition of the *new_commitment_tx* function. This function is used to construct a new commitment transaction given a payment channel, the amount being transacted and a secret hash value. As described

in section 5.6, a commitment transaction for a multi-hop payment requires one input and three outputs. The function consists of the following pieces:

- **Line 6:** The input for the commitment transaction is constructed using the funding transaction of the given payment channel, *channel_AB*.
- **Line 10:** The first output is constructed by sending node A the current amount that it owns minus the cost that it will pay for successful data delivery.
- **Line 14:** The second output is constructed by sending node B the current amount that it already owns (the channel remote balance).
- **Line 18:** The third output is paid to an HTLC script. This script, if it is broadcast to the public blockchain, is either spendable by the remote peer if they provide the secret hash, H, or it will be spendable by the local peer after 1000 block confirmations depending on which event happens first.
- **Line 21:** The different inputs and outputs are used to construct a commitment transaction object.
- **Line 24:** The input of the commitment transaction, the 2-of-2 multi-sig, is signed by the local peer.

```
1  def new_commitment_tx(node, current_channel, cost, H):
2
3    remote_peer = current_channel.peer
4
5    # Create input using the output from the funding tx
6    tx_in = TxIn(current_channel.funding_tx.id(), 0)
7
8    # Output 1 to node A
9    script_1 = p2pkh_script(node.address)
10   tx_out_1 = TxOut(amount = current_channel.local_amt - cost,
      script_pubkey = script_1)
11
12   #Output 2 to node B
13   script_2 = p2pkh_script(remote_peer.btc_addr.decode())
14   tx_out_2 = TxOut(amount = current_channel.remote_amt,
      script_pubkey = script_2)
15
16   #Output 3 to an HTLC
17   script_3 = Script([H, remote_peer.public_key, 1000, node.
      public_key])
18   tx_out_3 = TxOut(amount = cost, script_pubkey = script_3)
19
20   # Construct the commitment tx object
21   commitment_tx = Tx([tx_in], [tx_out_1, tx_out_2, tx_out_3])
22
23   #sign the commitment transaction
24   commitment_tx.sign(node.private_key)
25
26   return commitment_tx
```

Listing 6.16: Function *new_commitment_tx*

Once the commitment transaction has been constructed, the header of the payload and the payload can be created as shown in listing 6.17

```
1 header = {"source":node.address, "route":cheapest_route, "
    commitment_tx":commitment_tx}
2 payload = {"header":header, "encrypted_body": encrypted_body}
```

Listing 6.17: Payload Construction

The payload is sent to the next node and the client node will wait for a reply before proceeding as shown in listing 6.18. Once a reply is received, it is analysed to see if it contains a commitment transaction signed by the next node and if it contains the pre-image of hash H. The *signed* function used in line 5 takes a commitment transaction and a *Peer* object as arguments and returns *True* if the commitment transaction's multi-sig input is signed by the given peer. If the pre-image received is correct and the commitment transaction has been validly signed then the local node will update the relevant payment channel state for *channel_AB* as shown in line 8.

```
1 reply = next_hop.receive()
2 commitment_tx = Tx.parse(reply['commitment_tx'])
3 revealed_secret = reply['secret']
4
5 if(revealed_secret == secret and signed(commitment_tx, next_hop)):
6         print("Successful delivery of message")
7
8         next_hop_channel.pay(commitment_tx.tx_outs[2].amount)
```

Listing 6.18: Client node verifies reply from relay node

6.5.2 Relay Node

Activity Diagram

The activity diagram in figure 6.6 below shows the process of a node in relay mode. This diagram specifically shows the process of node B but node D will undertake a very similar process. As the diagram shows, once the necessary payment channels have been set up, two different program threads will be forked:

- **Thread 1:** Determines if the user of the device has changed the nodes relay fee. If the fee has been changed then the new fee is advertised to the rest of the mesh-network.
- **Thread 2:** Listens for any incoming payloads from node A and handling them accordingly before sending a new payload to node C.

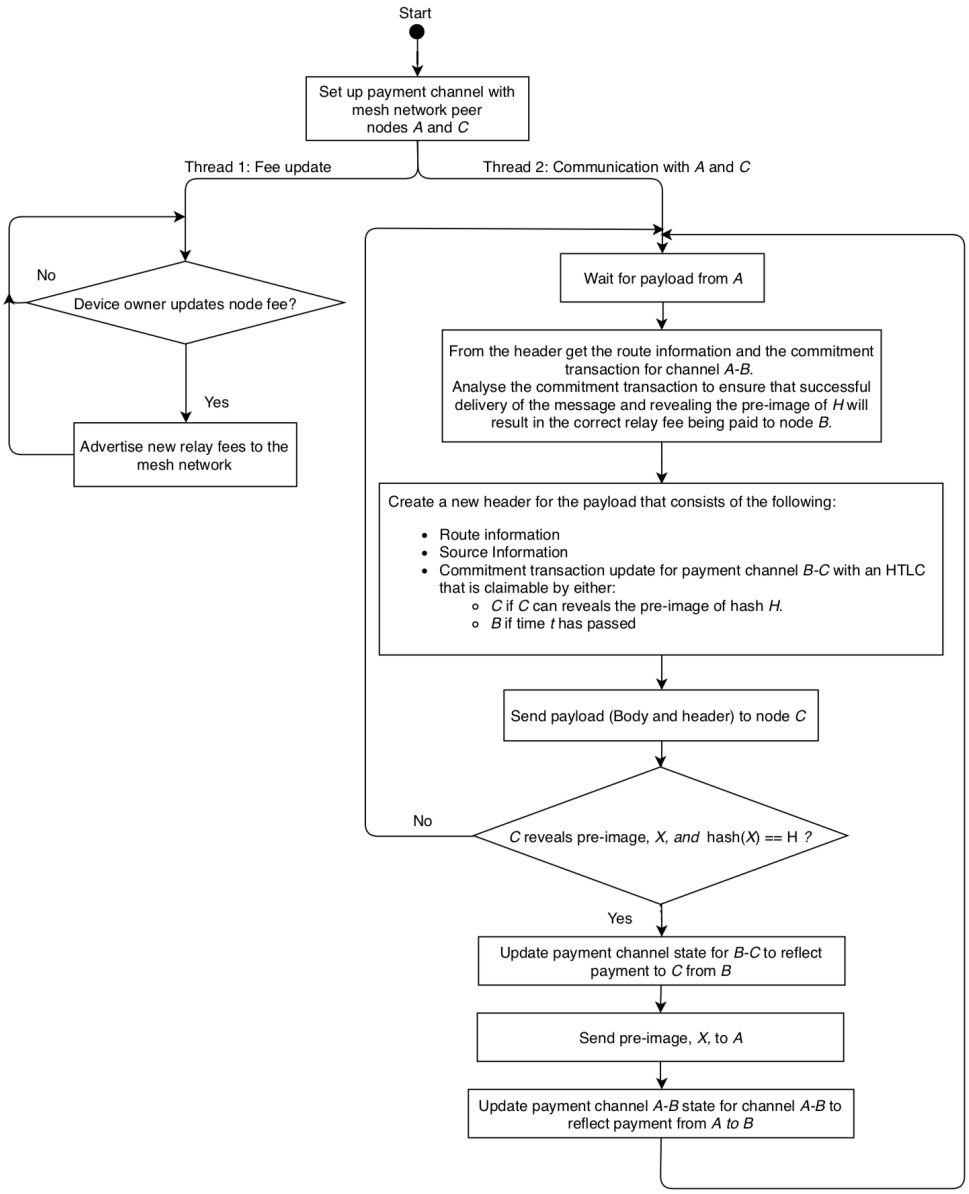


Figure 6.6: Activity diagram for a relay node such as node B

Pseudo-Code

In this section, various pieces of pseudo code used for thread 2 (see the activity diagram in figure 6.6) of the relay node program for node B will be described.

After the initial set-up of the device, it goes into *routing* mode and starts listening for any incoming payloads from the client node, node A. Listing 6.19 shows how the *Peer* and *Channel* objects associated with node A are determined in lines 2 and 3. The rest of the listing shows how node B waits for a payload from node A and then gets the header and encrypted body from the payload.

```
1 # get the Peer and Channel objects for node A
2 prev_hop = get_peer(peers, 'mst8broiaX4PFMFNbjfrBnMSnrVF42Jgd7')
```

```
3 prev_hop_channel = get_channel(prev_hop, channels)
4
5 # receive payload
6 received_payload = prev_hop.receive()
7
8 received_header = received_payload['header']
9 encrypted_body = received_payload['encrypted_body']
```

Listing 6.19: Define *Peer* and *Channel* of previous hop and wait for payload

The code in listing 6.20 shows how node B retrieves and stores the commitment transaction for $channel\_AB$ ($commitment_t x_p rev_h op$) from the received header (line 2) and then uses the *check_htlc_and_get_secret_hash* function (see appendix A.1) to check that the various elements of the commitment transaction are correct and to retrieve the hash value, H. The rest of the listing shows how node B constructs a new commitment transaction for $channel\_BC$ ($commitment_t x_n ext_h op$) and uses this to construct a new header for the payload. Line 19 shows how a new payload is constructed using the newly created header along with the encrypted body received from node A. This payload is then sent to the next hop on the route which is node C (line 22).

```
1 #get header info
2 commitment_tx_prev_hop = Tx.parse(decrypted_header['commitment_tx'
    ])
3 H = check_htlc_and_get_secret_hash(node, commitment_tx_prev_hop,
    prev_hop_channel)
4 cost_paid = route_cost(decrypted_header['route'], len(
    encrypted_body)
5
6 #adapt header
7 new_header = header
8 new_header['route'] = header['route'][1:]
9 cost_to_pay = route_cost(new_header['route'], len(encrypted_body))
10
11 #get info of next hop on the route
12 next_hop = get_peer(peers, new_header['route'][0][0])
13 next_hop_channel = get_channel(next_hop, channels)
14
15 #create a new commitment transaction for the next hop
16 commitment_tx_next_hop = new_commitment_tx(node, next_hop_channel,
    cost_to_pay, H)
17 new_header['commitment_tx'] = commitment_tx_next_hop
18
19 payload = {'header' = new_header, 'encrypted_body'=encrypted_body}
20
21 # send payload to next hop
22 next_hop.send(payload)
```

Listing 6.20: Verify received commitment transaction and construct new commitment transaction

The code in listing 6.21 shows the process of node B waiting for a reply from the next hop (node C). Once it receives a reply, it extracts the commitment transaction along with the revealed secret, X. In line 5, node B checks that the commitment

transaction has been validly signed by node C and that the revealed secret is the correct pre-image of the hash value, H. If these conditions are met, node B proceeds in signing the commitment transaction that it received from node A and then returns this to node A along with the revealed secret. Lines 7 and 18 show respectfully how node B updates the states of *channel_BC* and *channel_AB* to reflect the payments made.

```
1  reply = next_hop.receive()
2  commitment_tx_next_hop = Tx.parse(reply['commitment_tx'])
3  X = reply['secret']
4
5  if((sha256(X) == H) and signed(commitment_tx_next_hop, next_hop)):
6            # update channel state
7            next_hop_channel.pay(cost_to_pay)
8
9            # sign the previous hop commitment tx
10           commitment_tx_prev_hop.sign(node.private_key)
11
12           reply = {"commitment_tx": commitment_tx_prev_hop, "secret
       ": X}
13
14           # send reply to previous hop node
15           prev_hop.send(reply)
16
17           # update channel state
18           prev_hop_channel.paid(cost_paid)
19
20 else:
21           print("Cannot unlock HTLC")
```

Listing 6.21: Verify Reply

6.5.3 Gateway Node

Activity Diagram

The activity diagram in figure 6.7 shows the process of gateway node C. As can be seen in the diagram, the program running on node C will have three threads forked after all necessary payment channels have been set up:

- **Thread 1:** Determinines if the user of the device has changed the nodes relay fee. If the fee has been changed then the new fee is advertised to the rest of the mesh-network.
- **Thread 2:** Waits for incoming payloads from node B and then decrypts the payload to reveal the hash pre-image, X. It will then send this pre-image back to node B along with a signed commitment transaction.
- **Thread 3:** Identical to thread 2 except that it responds to incoming payloads from node D.

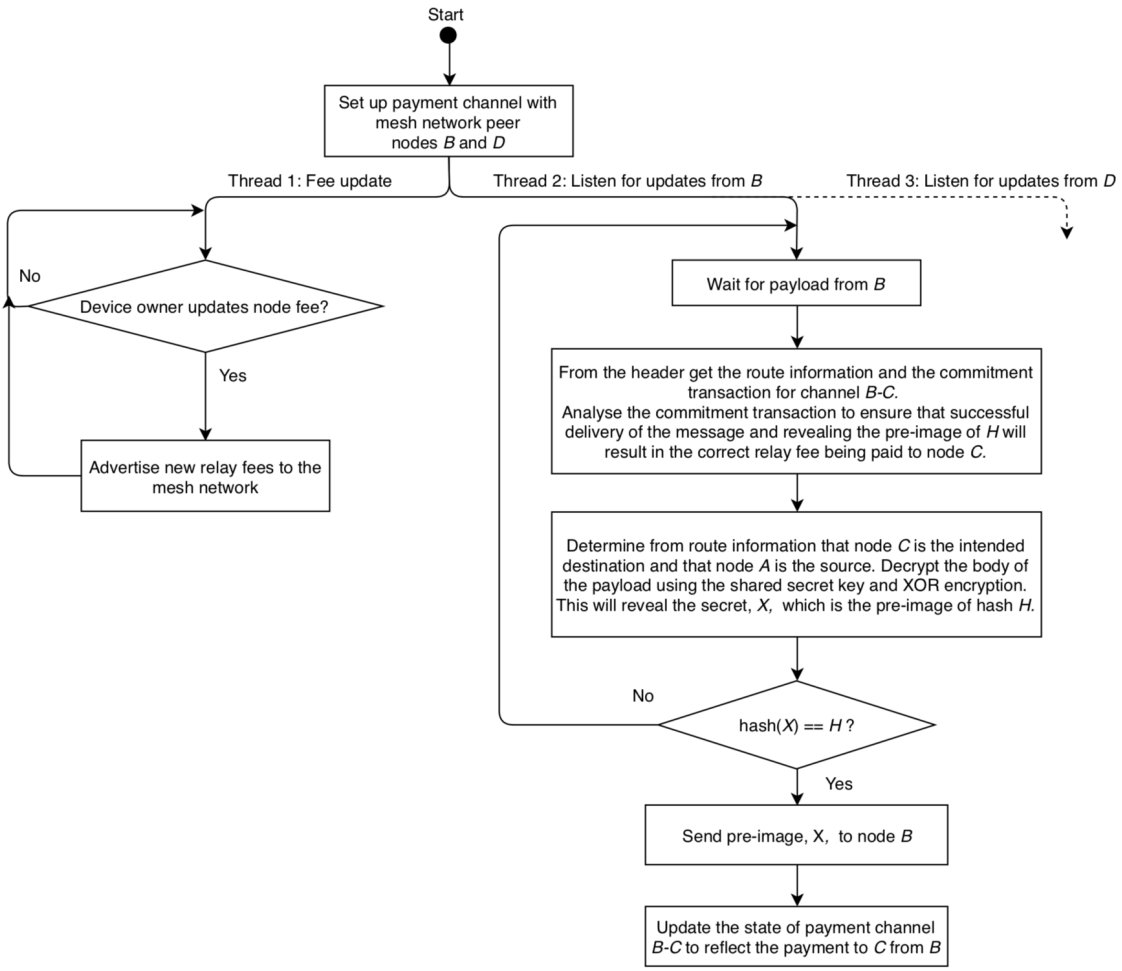


Figure 6.7: Activity diagram for a gateway node such as node C

Pseudo-Code

In this section the pseudo code for thread 2 (see the activity diagram in figure 6.7) of the gateway node program for node C will be described.

Listing 6.22 shows the process carried out by node C in thread 2 which is responsible for listening and responding to node B. The process is similar to that described for relay node B except that node C is able to decrypt the encrypted body of the payload. The process of decryption is shown in lines 16-18 where node C uses the symmetric key that it shares with node A to decrypt the body and reveal the secret, X.

Line 21 shows how node C checks that the revealed secret is in fact the pre-image of the hash value, H, that was obtained from the commitment transaction received by node B. If the condition is satisfied then node C signs the commitment transaction and sends it back to node B along with the revealed secret. Node C then updates its stored state of $channel\_BC$ to reflect a payment from node B to node C (line 31).

```
# get the Peer and Channel objects for node B
prev_hop = get_peer(peers, node_address)
current_channel = get_channel(prev_hop, channels)

# receive payload
received_payload = prev_hop.receive()
header = received_payload['header']
encrypted_body = received_payload['encrypted_body']

# information from header
source = get_peer(peers, header['source'])
commitment_tx = Tx.parse(header['commitment_tx'])
H = check_htlc_and_get_secret_hash(node, commitment_tx,
    current_channel)

# decrypt body and get information
sym_key_source = source.sym_key
decrypted_body = decrypt(encrypted_body, sym_key_source)
X = decrypted_body['secret']

#check that you can suceesfully unlock the htlc output
if(sha256(X) == H):

            # sign the commitment tx
            commitment_tx.sign(node.private_key)

            # construct and send reply
            reply = {"commitment_tx": commitment_tx, "secret": X}
            prev_hop.send(reply)

            # update the appropriate channel state
            current_channel.paid(commitment_tx.tx_outs[2].amount) }

else:
            print("Cannot unlock HTLC")
```

Listing 6.22: Determine previous hop node

6.6 Payment for the transfer for multiple data packets using Shamir Secret Shares

In order to incorporate the use of Shamir Secret Shares into the implementation, only a few adjustments need to be made to the implementation described in section 6.5. These adjustments will be explained in this section.

The client node program used for node A remains as is described in section 6.5.1 with the only two exceptions. The first of these exceptions is shown in listing 6.23. Node A uses the *shamir* Python library to generate a random secret number, X, and n shares all of which will be required to reconstruct X (line 3). The hash value, H, is then generated from X and is included in a commitment transaction. The *shares* variable is an array of elliptic curve points and each of these will be included in a different payload body and encrypted such that only node C will be able to decrypt

them. Instead of sending only one payload including one header and one encrypted body to node B, node A will instead send one header and then n encrypted bodies to node B (as shown in the sequence diagram in figure 5.26).

```
import shamir

X, shares = shamir.make_random_shares(n, n)
H = sha256(X)
```

Listing 6.23: Construction of Shamir Secret Shares

The program used for a relay node such as node B and D only require one adaptation which is that instead of receiving and relaying one encrypted body, they must receive and relay n encrypted bodies to node C.

The gateway node program used for node C is adapted so that it receives n encrypted bodies, decrypts all the bodies and hence reveals all of the shares. Once node C has all n of the Shamir shares, it will be able to reconstruct the original secret, X. This is shown in listing 6.24.

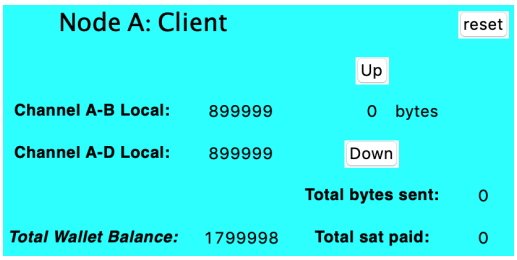
```

X = shamir.recover_secret(shares)
```

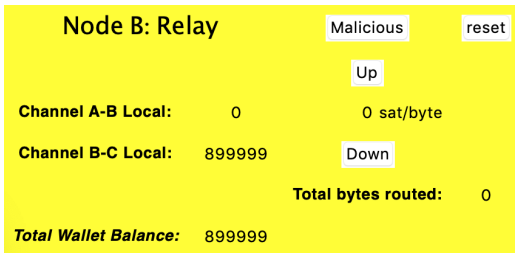
Listing 6.24: Recovering X from the Shamir shares

6.7 User Interfaces

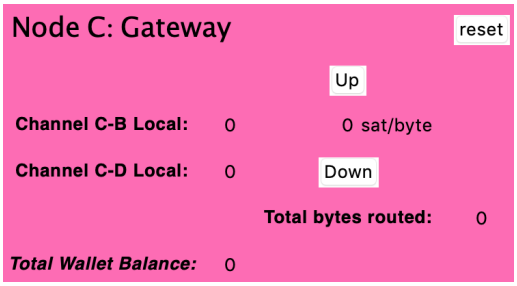
Using the Python library called "Tkinter", user interfaces were designed for each node in the network. The user interface buttons and labels were linked to the main programs of each node. Using the *Up* and *Down* buttons on the relay and destination node immediately changes the fee advertised by the node. And the buttons on the client node will change the size and number of packets that the client node requires to be delivered to the gateway node. All the channel balance labels will show changes in channel states as soon as they occur. Figures 6.8a, 6.8b, 6.8c and 6.8d below show the individual node user interfaces and figure 6.9 shows a picture of the entire Raspberry Pi set-up each with their respective user interfaces.



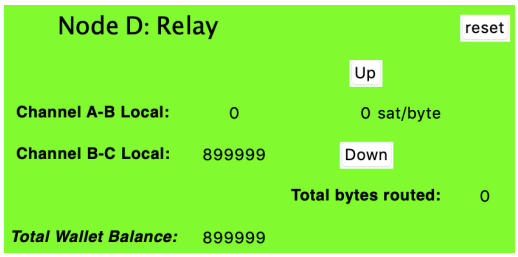
(a) Display for node A



(b) Display for node B



(c) Display for node C



(d) Display for node D

Figure 6.8: User interfaces for the mesh network nodes

Figure 6.9: Complete mesh-network set-up

Chapter 7

Results

In this Chapter, the system will be tested to verify that it works as expected. The system will then be analysed to ensure that all the acceptance tests outlined in Chapter 4 are passed and that all the associated user and functional requirements are met. Lastly, an analysis will be done so as to measure the performance of the system.

7.1 System Tests

Figure 7.1 below shows the initial state of all the nodes when they are activated. From the figure, all the existing payment channels and their current states are shown. For example it can be seen that node A and node B share a payment channel (channel $A - B$) and that currently the entire capacity of the channel belongs to node A with 899999 satoshis assigned to it and 0 satoshis assigned to node B.

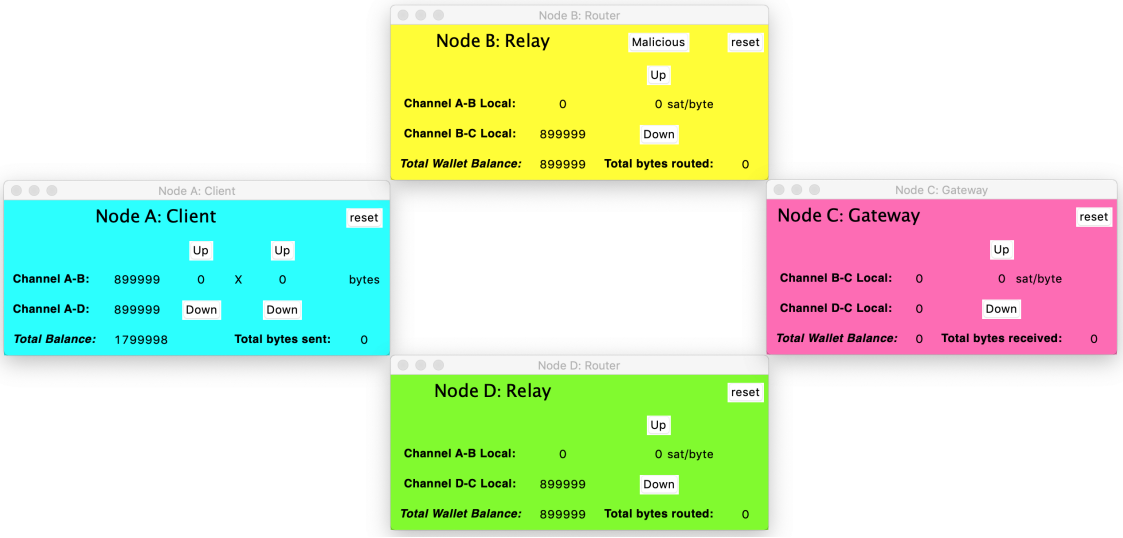


Figure 7.1: Mesh-network initial set-up

To test that the system works as expected, the following steps were taken:

Step 1:

- Node B's fee is set to 1 sat/byte.
- Node D's fee is set to 2 sat/byte.
- Node C's fee is set to 2 sat/byte.
- Node A is instructed to create and send one 10 byte packet.

From the above settings, it is expected that node A will chose the route $A - B - C$ to route its payload since it is the cheaper of the two routes (a total of 3 sat/byte).

Figure 7.2 below shows the results of step 1.

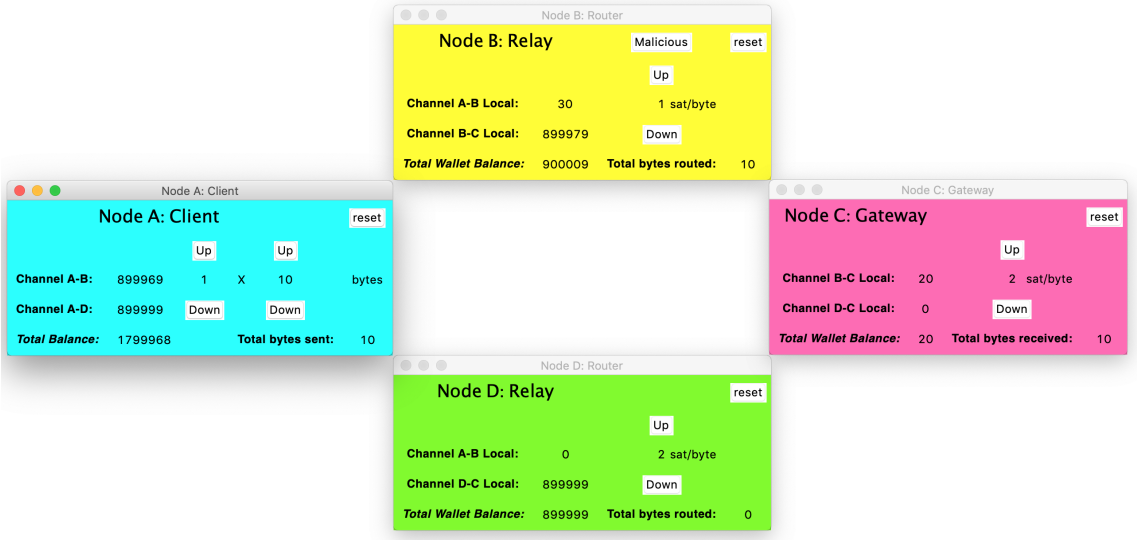


Figure 7.2: Mesh-network test step 1

From the screen shot in figure 7.2, the following can be seen:

- Node A paid a total of 30 satoshis using payment channel $A - B$ in order to pay for the service of routing the 10 byte packet
- Node B received 30 satoshis from node A in the $A - B$ payment channel and paid 20 satoshis to node C using the $B - C$ payment channel. Node B therefore made a profit of 10 satoshis for successfully routing the 10 byte packet as can be seen by the increase in node B's total wallet balance.
- Node C received and routed the 10 bytes sent by A and was paid 20 satoshis in the $B - C$ payment channel.
- Node D shows no changes as it did not assist in routing any data and therefore was not paid.

From the above observations it is clear that the system has behaved as expected.

Step 2:

- Node B's fee increased to 3 sat/byte.
- Node A is instructed to create and send one 20 byte packet.

After node B changes its fee to be higher than node D's fee, it is expected that node A choose route $A - D - C$ to route its 20 byte message.

Figure 7.3 below shows the state of the system after step 2

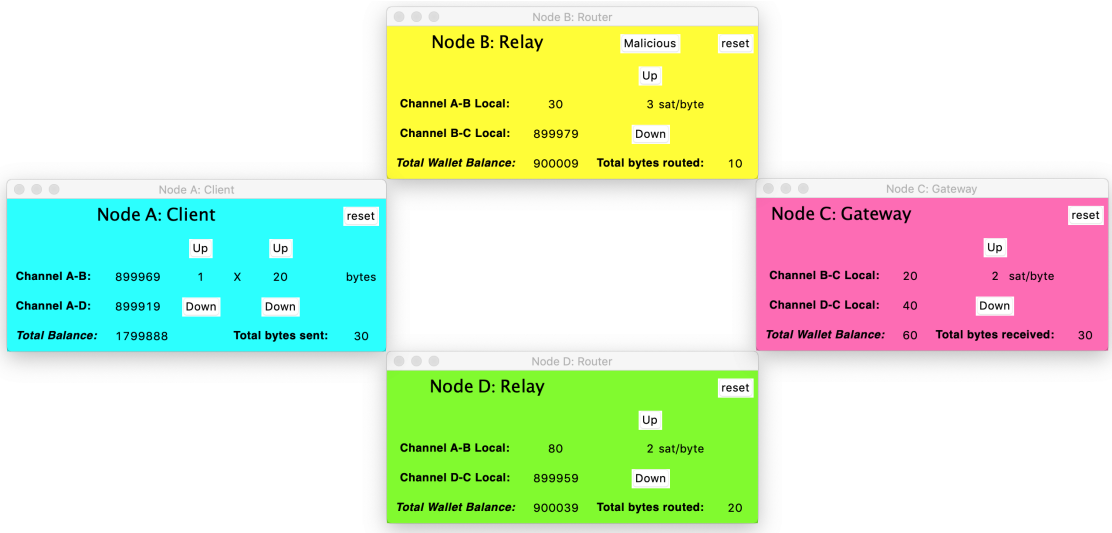


Figure 7.3: Mesh-network test step 2

From the screen shot in figure 7.3, the following can be seen:

- Node A paid 80 satoshis using payment channel $A - D$ in order to pay for the service of routing the 20 byte packet using route $A - D - C$.
- Node D received 80 satoshis from node A in the $A - D$ payment channel and paid 40 satoshis to node C using the $D - C$ payment channel. Node D therefore made a profit of 40 satoshis for successfully routing the 20 byte packet as can be seen by the increase in Node D's total wallet balance.
- Node C received the 20 bytes sent by A and was paid 20 satoshis in the $D - C$ payment channel.
- Node B shows no changes from its state after step 1 as it did not assist in routing any data and therefore was not paid further.

From the observations made above, it is clear that the system behaved as expected in step 2.

Step 3:

- Node B's fee is set to 1 sat/byte
- Node B is altered to be a malicious node and tries to prove to node A that it delivered node A's packet without having done so. Node B tries to do this so that it can be paid by node A without paying node C which would result in a larger overall balance. Node B does this by sending an incorrect pre-image (X) value back to node A.
- Node A is instructed to create and send one 10 byte packet.

It is expected that node A will initially chose route $A - B - C$ due to it being the cheapest route but due to the requirements and workings of the system it is expected that node A will immediately notice when node B returns an incorrect pre-image and switch to the next cheapest route, $A - D - C$. Node B should not be able to steal any funds from node A.

Figure 7.4 below shows the state of the system after step 3:

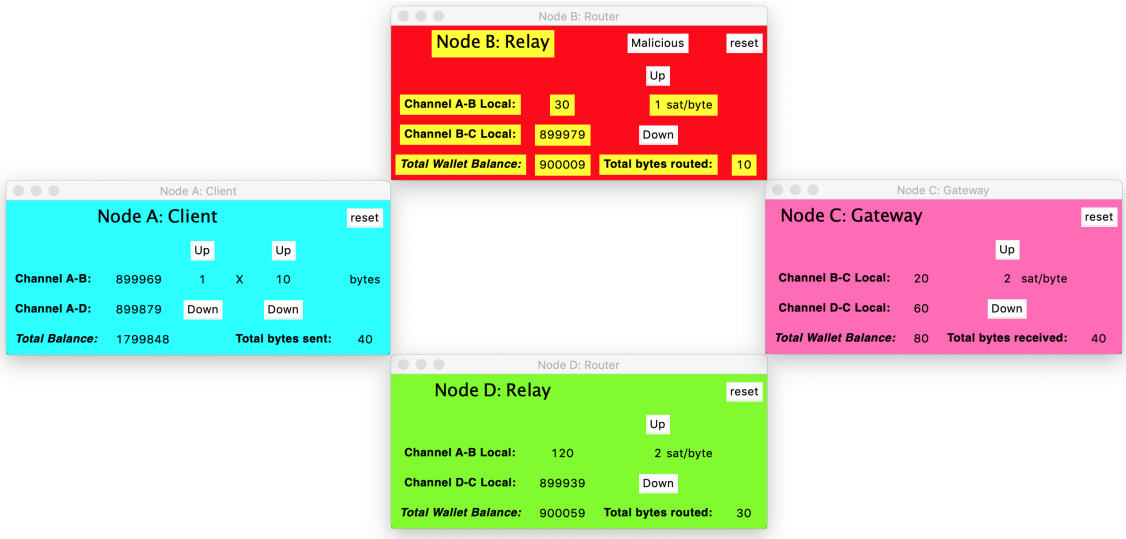


Figure 7.4: Mesh-network test step 3

From the screen shot in figure 7.4, the following can be seen:

- Node B is in malicious mode and so even though it is advertising a smaller fee than node D, node A chose the route $A - D - C$ to route its packet.
- Node B's channel and overall wallet state has not changed from step 2.
- Node D received 40 satoshis from node A in the $A - D$ payment channel and paid node C 20 satoshis in the $D - C$ payment channel.
- Node C received the 10 byte packet and was paid 20 satoshis in the $D - C$ payment channel.

The system behaves as expected for step 3.

7.2 Acceptance Test Procedures

The aim of this section is to ensure that all the given ATPs identified in section 4.1.6 are passed and hence that all the user and functional requirements of the system have been met.

7.2.1 ATP1

From the system tests done in section 7.1, each of the devices are able to connect to the mesh-network and successfully contribute to the network given their assigned roles. The client node, node A, is able to send packets to a gateway node, node C, successfully by routing the data through the mesh-network. This means that in practice, node A is able to connect to the internet by using the mesh-network services. Nodes B and D successfully act as relay nodes for node A. ATP1 is thus passed meaning that FR1 and UR1 have been met.

7.2.2 ATP2

The Bitcoin addresses produces by each node are as follows:

- **Node A:** $mst8broiaX4PFMFNbjfrBnMSnrVF42Jgd7$
- **Node B:** $mfwnjj1Jbd1uwXbj5Q4FUjmkEcGqQQsYDn$
- **Node C:** $n1weDdde5xXLfPeutESLaG8swr5jLCqz72$
- **Node D:** $mmqrZXdvAi8mcjvXGJX2eJdA37kWXmCWjW$

Each address is unique and hence each is derived from a unique Bitcoin private-public key pair. The acceptance test ATP2 has passed and hence FR2 and UR2 have been satisfied.

7.2.3 ATP3

In order to test whether or not the funding and commitment transactions created by the devices are valid or not, the transactions will be serialised into their hexadecimal form and an external Bitcoin transaction decoder will be used to verify that the transactions are of the correct form.

When a payment channel is set up, a funding transaction is constructed. For example when node A funds a channel with node B it is expected that the input will be the transaction shown in figure 6.4 that paid node A's address 899999 satoshis (from a testnet faucet website) and the output is expected to be a 2-of-2 multi-sig transaction

that requires the signatures of both nodes A and B to spend. Figure 7.5 is the output given by a bitcoin transaction decoder when the hexadecimal serialisation of the funding transaction for channel $A - B$ is given as an input:

Figure 7.5: Decoded funding transaction

From the figure above, the following can be seen:

- The input of the transaction (*vin*) has a transaction id (*txid*) identical to that shown in figure 6.4 which means that it is referencing the same transaction.
- The output (*vout*) has a value of 0.00899999 BTC (or 899999 satoshis) as expected and the output also shows that it is of a multi-sig type (*type*), that two signatures are required to spend it (*reqSigs*) and that the signatures must be those associated with the two addresses shown (*addresses*). From the results of ATP2, it can be seen that these addresses belong to nodes A and B.

The contents of the funding transaction is thus as is expected.

When a commitment transaction is created (for example between node A and B), the input must be the funding transaction for the channel and there must be three outputs the details of which are described in section 5.6. Figure 7.6 shows the output given by a Bitcoin transaction decoder when the hexadecimal serialisation of a commitment transaction for channel $A - B$ is passed to it in the case where node A proposes to pay node B 30 satoshis in exchange for revealing secret X as would be done in the step 1 of the system tests in section 7.1.

Figure 7.6: Decoded commitment transaction

From the above figure, the following can be confirmed:

- There is one input that references the funding transaction shown in figure 7.5 (the *txid* values are the same)
- Output 1 pays 899969 satoshis to the address belonging to node A
- Output 2 pays 0 satoshis the address belonging to node B
- Output 3 pays 30 satoshis to an HTLC script.

Both the funding and commitment transactions constructed by the nodes are valid Bitcoin transactions and so ATP3 is passed and hence FR2, FR3 and UR2 are satisfied.

7.2.4 ATP4

From the results of step 1 and 2 of the system tests that were done in section 7.1, it is shown that the client node, node A, immediately switches to using the cheapest possible route to node C based on the advertised prices of nodes B and D. ATP4 is thus passed and hence FR4, FR5, UR3 and UR4 are satisfied.

7.2.5 ATP5

Through the use of shared symmetric key cryptography and xor-encryption, the client node is able to encrypt data for the destination node without having any form of direct communication with that node. Thus ATP5 has been passed and so FR6 and UR5 have been satisfied.

7.2.6 ATP6

The results of step 3 of the system tests in section 7.1 show that when node B starts acting maliciously by sending node A an incorrect pre-image value, node A immediately switches to using a different route. ATP6 is passed and hence FR7 and UR6 are satisfied.

7.2.7 ATP7

The results of all the system tests done in section 7.1 show that all channel states are updated appropriately when data is being transmitted through the network. Thus ATP7 is passed and hence FR1, FR7, UR6 and UR1 are satisfied.

7.3 System Analysis

The requirement of a wireless mesh network is that it transmit data from a client device to a destination device in the shortest possible time, in other words to have the highest possible bandwidth. To determine how the added payment features that make the mesh-network incentivised would affect the performance of the network, various investigations are done.

7.3.1 Comparison between incentivised and un-incentivised mesh-network performance

The first investigation involves comparing the performance of a mesh-network without a payment structure (un-incentivised) to one with a payment structure (incentivised). Figure 7.7 below is a sequence diagram that shows the content and sequence of messages passed between nodes in an un-incentivised mesh network for the case where a single payload is being transmitted. The sequence digram of the incentivised mesh-network can be seen in can be seen in section 5.6.2 figure 5.23.

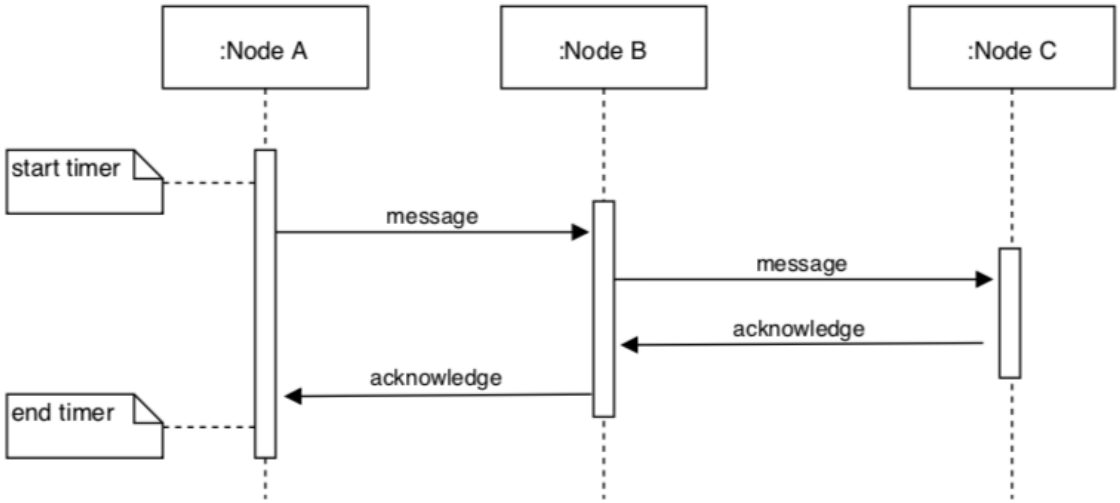


Figure 7.7: Sequence diagram for an un-incentivised mesh-network

A timer is used to test the performance of the two mesh-network set-ups (un-incentivised and incentivised). The timer is started when the client node, node A, starts the process of constructing the message and is stopped when either an acknowledgement alert is received from node B in the case of the un-incentivised system or when node A receives the correct pre-image from node B in the case of the incentivised system. For both tests, the route length (number of node hops) is kept at a constant of 2 hops (as depicted in the sequence diagrams) and the number of packets being sent is kept constant at one. The tests are then repeated for different packet sizes varying between 1 byte and 1000 bytes. 1000 bytes is chosen as the maximum packet size to test due to the fact that the Maximum Transmission Unit (MTU) of the Raspberry Pi is 1500 bytes [13]. Thus 1000 bytes was chosen as the maximum packet size so that 500 bytes could be used for the overhead of transaction and Shamir share data in the case of the incentivised network. The results of these tests can be seen in figure 7.8 below. Note that the y-axis of the graphs uses a logarithmic scale.

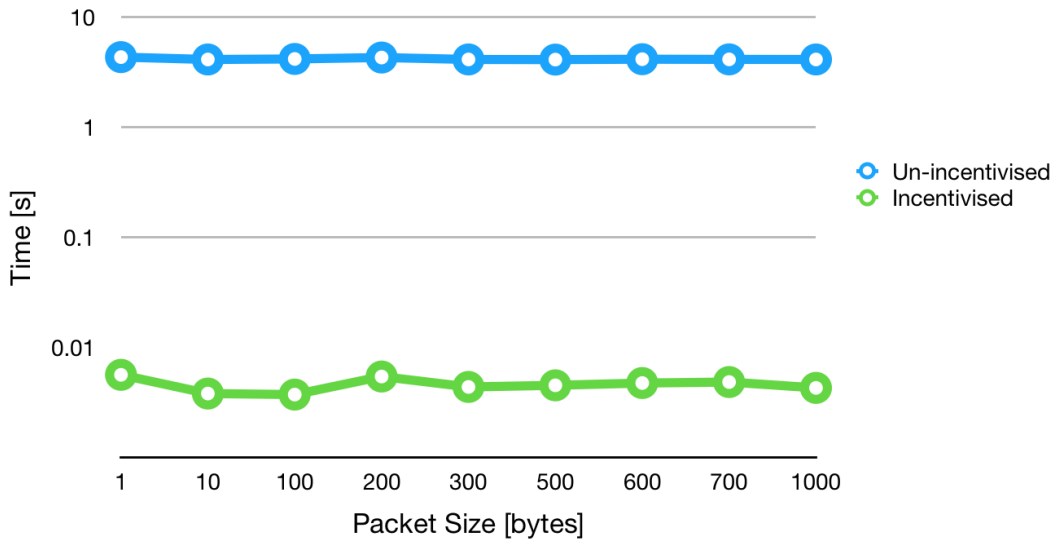


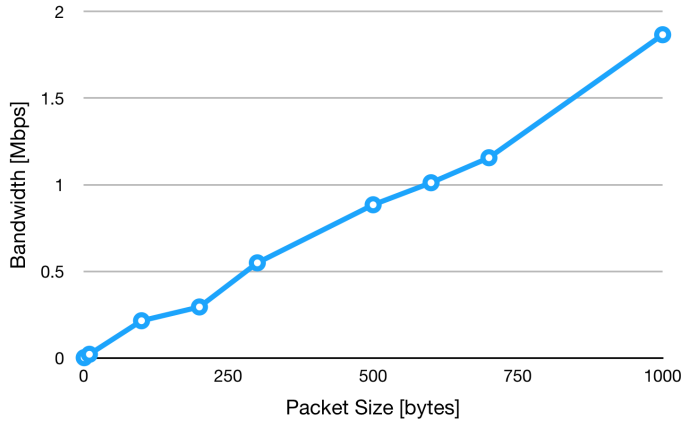
Figure 7.8: Graph showing the time taken to complete the process of delivering packets of different sized in an incentivised and un-incentivised network

From figure 7.8 it can be seen that the time required for the incentivised mesh-network to complete the delivery of a message and complete the associated payments is significantly more than the time required to deliver a message in an un-incentivised mesh network. In fact the incentivised network takes on average 690 times longer than the un-incentivised mesh network when the tests are performed on the Raspberry Pi mesh-network set-up. It is also evident that the time required to complete the process of transmitting a single packet does not vary significantly as the size of the packet increase. Table 7.1 below shows the average times recorded for the two systems.

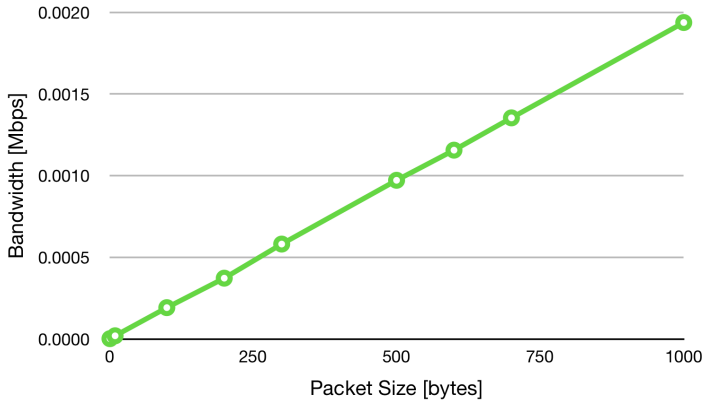
Mesh-Network type	**Time** [s]
Un-incentivised	0.0045
Incentivised	4.1575

Table 7.1: Performance comparison between an incentivised and un-incentivised mesh-network

From the same timing results, figure 7.9a and 7.9b below can be plotted. These graphs show the bandwidth in Mbps (Mega bits per second) that is possible in the un-incentivised system (figure 7.9a) and the incentivised system (figure 7.9b).



(a) Bandwidth against packet size in an un-incentivised mesh-network



(b) Bandwidth against packet size in an incentivised mesh-network

Figure 7.9: Bandwidth and packet size comparisons of an un-incentivised and incentivised network

For a maximum packet size of 1000 bytes, the bandwidth achieved in the incentivised network is approximately 0.002 Mbps compared to 1.87 Mbps in the un-incentivised network. Therefore the un-incentivised mesh-network performs better by a factor of 935. As stated before, these results show the performance of each system sending a single packet across a fixed route length.

7.3.2 Timing each part the incentivised mesh network process

Due to the significant difference in performance between the un-incentivised and incentivised mesh-network systems, a further analysis is done into the workings of the incentivised system in order to determine the cause of its poor performance. In order to do this, the various sections of the program of each node involved in the delivery of a message were timed. Figure 7.10 shows the results of this test. The diagram shows the various sections of each program involved in routing a message

through route $A - B - C$ and the duration of each section of each program given as a percentage of the overall process time.

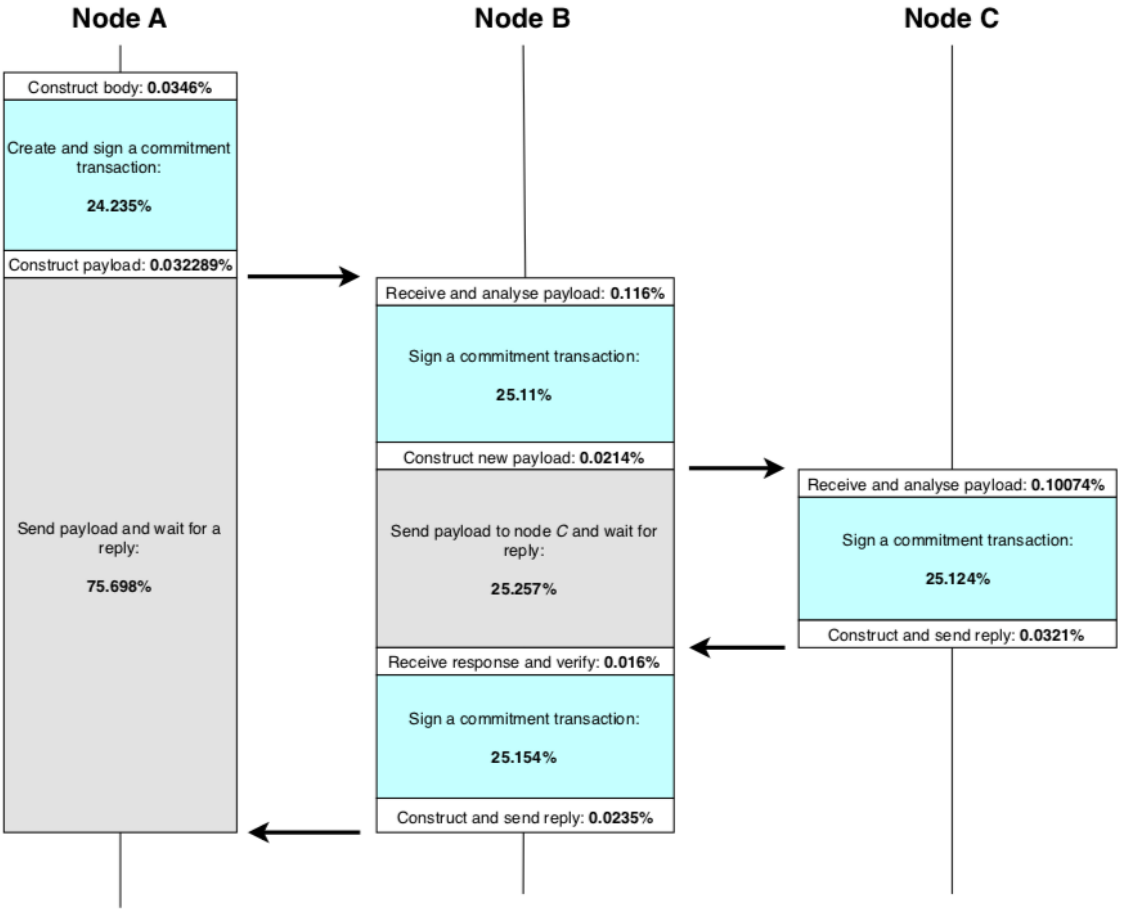


Figure 7.10: Diagram showing the duration of each section of each program as percentages of the overall process time

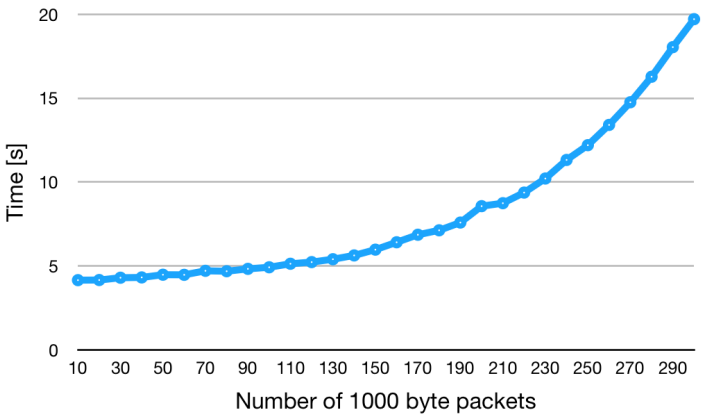
From figure 7.10 above it is evident that the the process of signing commitment transaction occurs four times in the three node (or two hop) example. Altogether 99.623% of the overall process time is spent signing commitment transactions. The process of signing a commitment transaction involves having to do scalar multiplication of a finite-field, elliptic curve point and so it is clear that it is this multiplication operation that causes the poor performance of the incentivised mesh-network.

7.3.3 Analysis of an incentivised mesh-network using Shamir Secret Sharing

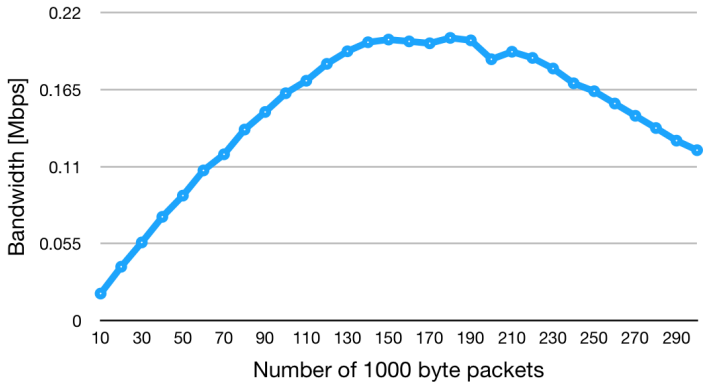
In sections 5.7 and 6.6, the design and implementation of enabling multiple packet to be transmitted while only requiring the associated payment information to be communicated once are outlined respectfully. This was done using Shamir Secret Sharing.

To analyse the performance of the incentivised mesh-network with the added feature, the process can be timed as done in the tests discussed in section 7.3.1. For these tests the route length is kept constant at two hops and the packet size is kept at a constant of 1000 bytes since 1000 bytes will result in the maximum bandwidth in the transmission of a single packet as was determined in section 7.3.1. The number of packets to be transmitted for each payment process will be varied from 1 to 300 packets. In other words the number of bytes being transmitted will vary from 1000 bytes to 300000 bytes.

The results from this test can be seen in figure 7.11a. From these results, the bandwidth of the system is calculated and shown in figure 7.11b.



(a) Time against the number of 1000 byte packets



(b) Bandwidth against the number of 1000 byte packets

Figure 7.11: Graphs showing how time and bandwidth of the system vary with the number of 1000 byte packets

From the results shown in figure 7.11b it is clear that the bandwidth of the network reaches a peak value of 0.202 Mbps when the number of packets, and hence the number of Shamir shares, is 180. The decrease in bandwidth that occurs as the number of packets (and shares) exceeds 180 can be attributed to the computation required by the Shamir algorithm to produce such a high number of shares and then to reconstruct a secret from these shares. From the results in figure 7.11a it is seen that the time taken for the process to complete when the number of packets is 180

is 7.126 seconds. Since it is known that the time required for signatures in a 2 hop Raspberry Pi network is 4.1575 seconds (see table 7.1), it can be estimated that the time required for Shamir share construction and reconstruction of 180 shares is:

$$7.126 - 4.1575 = 3.0096s$$

7.3.4 Performance Prediction

Using the results from the various tests performed, predictions can be made of how the incentivised mesh-network would perform on a larger scale. The following points will be used to estimate the performance of a larger mesh-network:

- The length, L, of a route through the mesh-network can be expressed in terms of the number of hops, h, in the route. The relationship between L and h can be expressed as follows:
$$h = L - 1$$
- Since the time used for the process of signing commitment transaction makes up for almost 100% of the overall process time when only a single packet is transmitted, determining how many signing operations will need to be performed given the length (or number of hops) of the chosen route will give a good estimate of how the system will perform. Given a route of length L, the client and destination nodes will always perform one signature operation each and every relay node (L-2 nodes) will need to perform two signature operations. The number of signatures required can thus be expressed as follows:

$$\begin{aligned} num\_signatures &= 2 + ((L-2) \times 2) \\ &= 2 + ((h-1) \times 2) \\ &= 2 + 2 \times h - 2 \\ &= 2 \times h \end{aligned} \tag{7.1}$$

- The test results displayed in table 7.1 show the performance results of a 2 hop process performed using Raspberry Pis. From the derivation above and from the diagram shown in figure 7.10 it is clear that the times recorded in the table are approximately the times required for four signature operations. Using this approximation, the time for a single signature operation can be calculated:
$$time\_per\_signature \approx 4.1575/4 = 1.0394s$$
- From the results shown in figure 7.11a, it is clear that the time used by the Shamir algorithm starts to become more significant as the number of packet (and hence required shares) increases. Thus if a large number of shares is used then the time taken by the Shamir algorithm must also be taken into account when modeling the performance of the system.

- From the the points made above, approximations can be made regarding how the system will perform as the number of hops on a given route increases. The following equation can be used to express the total time required to complete the process of delivering a single message:

$$\begin{aligned} total\_time &\approx (num\_signatures \times time\_per\_signature) + shamir\_time \\ &= (2 \times h \times time\_per\_signature) + shamir\_time \end{aligned} \tag{7.2}$$

- Furthermore, the possible bandwidth of the network can be calculated using equation 7.2 as follows:

$$\begin{aligned} bandwidth &= \frac{b \times n}{total\_time} \\ &= \frac{b \times n}{(2 \times h \times time\_per\_signature) + shamir\_time} \\ &= \frac{b \times n}{(2 \times h \times time\_per\_signature) + shamir\_time} \times \frac{1}{h} \end{aligned} \tag{7.3}$$

  where b is the number of bytes per packet, n is the number of packets and h is the number of hops on the route.

Using the results from previously discussed tests, the following values will be used in order to predict the best possible performance of the system:

- $b = 1000$ bytes due to the maximum MTU of the RPi
- $n = 180$ packets due to the results from section 7.3.3
- $time\_per\_signature = 1.0394$ seconds
- $shamir\_time = 3.0096$ seconds due to the fact that 180 shares are required.

These variables can be substituted into equation 7.3 and the number of hops (h) can be varied to estimate the bandwidth of the network for different route lengths. The results of this process can be seen in figure 7.12. It is evident that the performance of the network decreases exponentially as the mesh-network route length increases.

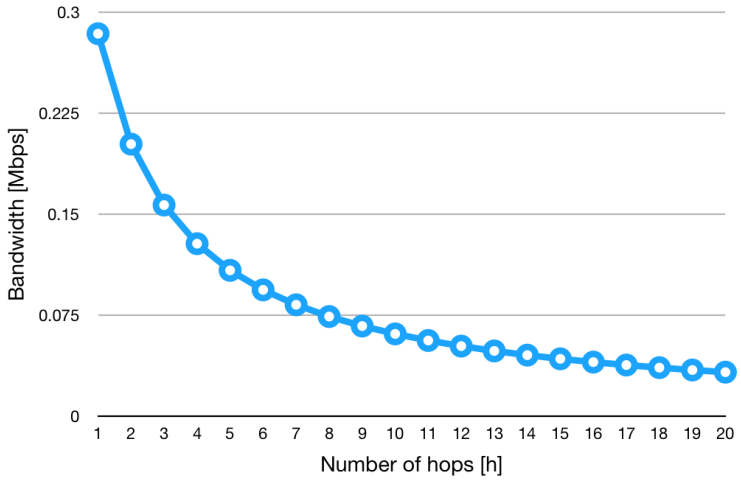
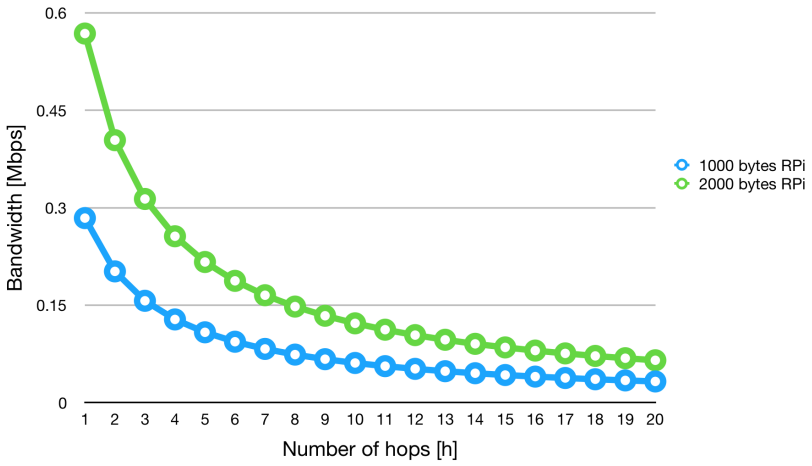
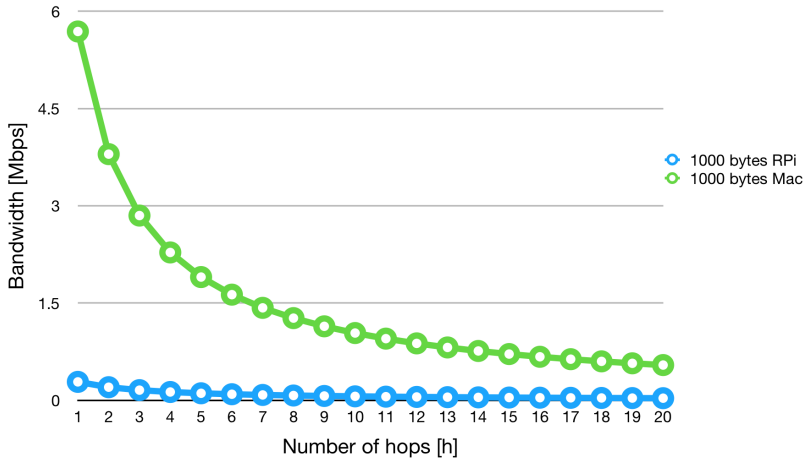


Figure 7.12: Bandwidth of the network as the route length is varied

Figure 7.13a below shows how the performance of the system would improve, relative the the results shown above, if the packet sizes could be increases to 2000 bytes. Figure 7.13b below shows how the performance of the system would improve if a more powerful computer was used. In this case, an Apple Mac computer was used which had a *time_per_signature* value of 0.0664 seconds and a *shamir_time* value of 0.127 seconds when the number of packets is set to 180.



(a) Bandwidth improvement given an packet size increase from 1000 bytes to 2000 bytes



(b) Bandwidth improvement using an Apple Mac computer compared to a Raspberry Pi to do signature calculations

Chapter 8

Discussion

In this chapter, the results in Chapter 7 will be discussed in order to determine if the proposed method of creating an incentivised mesh network is viable.

8.1 Performance of the implemented system

The purpose of the mesh-network is to provide a reliable internet connection for the devices in the network. Therefore, it is important that the bandwidth provided by the network is large enough to support the basic needs of people using the network. Table 8.1 below shows some of the different applications that an internet connection is often used for along with the minimum bandwidth required to make use of each.

Application	**Minimum Required BW [Mbps]**
WhatsApp Voice Call	0.064
Social Media use (e.g. Facebook or WhatsApp messenger excluding any videos)	0.03
Streaming Audio (e.g. Spotify)	0.096
Sending emails	0.08
Streaming videos (e.g. Youtube or Netflix)	0.5

Table 8.1: Minimum internet bandwidth requirements of various popular applications [17] [18] [19]

From the above table and from the bandwidth estimations of the implemented system shown in figure 7.12, it can be seen that the incentivised mesh-network made up of Raspberry Pi devices would be able to provide the minimum bandwidth required by most of the applications listed above as long as the number of hops does

not exceed 6. If the mesh-network is made up of devices organised in a grid shape of $n \times n$ devices and all the devices had payment channels with their direct neighbouring nodes (as is shown in figure 8.1 for a grid of 3x3 devices) then the largest distance would be the diagonal of the grid which would consist of n-1 hops. Since the incentivised system can provide decent bandwidth for routes smaller than or equal to 6 hops, it means that in theory a mesh-network of up to 49 devices (7x7) can be catered for if they are organised in a grid shape and have the required payment channels set up. The requirements for the application of streaming videos is not met by the system.

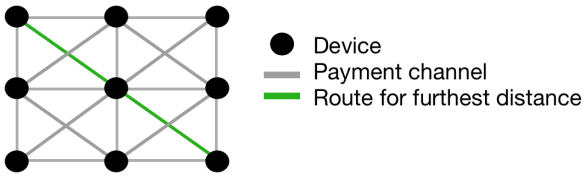


Figure 8.1: Diagram showing the layout and payment channel connections of a perfect 3x3 mesh-network

8.2 Possible performance of the system given certain improvements

From the results it is evident that the minimum requirements for the application of streaming videos would not be met with the implemented system. However, as is evident by the graph in figure 7.13a, if devices with larger MTU sizes were used that would allow 2000 byte packets instead of 1000 byte packets to be transmitted, then the system would be able to cater for the application of streaming videos as long as as the number of hops does not exceed 1. Any reasonably sized mesh network would require more hops and so it can be concluded that the system is not suitable for the application of streaming videos even when the MTU size is doubled.

Figure 7.13b shows the improved performance expected from using a more powerful computer for the mesh network device instead of a Raspberry Pi (1.2 GHz CPU). The improved performance is as a result of the more powerful computer being able to sign transactions and perform Shamir operations faster than the Raspberry Pi. The figure specifically shows the performance improvement when an Apple Mac computer is used and shows that with the Apple Mac processor (2.2 GHz), the bandwidth requirements of all the applications listed in table 8.1 can be met for routes up to 21 hops long. This means that, given a grid shaped mesh-network with all the required payment channels, a theoretical maximum mesh network size of 484 (22x22) nodes can be achieved (see the explanation for a 3x3 grid in section 8.1 above) all while providing each node with the minimum required bandwidth for all the applications listed in table 8.1.

8.3 Trade-off between cost and system performance

As discussed in the literature review in Chapter 3, Lot49 focused on designing a system that enables micropayments to be made between devices in exchange for the delivery of data packets. Ideally these packets are as granular as possible so that new micropayments can be made for the delivery of each individual packet. By breaking data into smaller packets and making micropayments for the routing of each individual packet it will allow a device to make routing decisions more often and therefore may result in an overall cheaper cost for the client. After implementing and then analysing the performance of such a system, it is clear that the possible benefits of making continuous micropayments for the transmission of granular packets comes with great computational overhead and so results in poor performance.

In section 6.6 of this report, the implementation was adjusted by using Shamir Secret Shares to enable payments to be made for the transfer of multiple packets. This alteration allows more data to be transmitted while requiring fewer payments to be made which results in a significant performance improvement as is shown in section 7.3.3. This adjusted method, however, means that the client device will have fewer opportunities to switch routes to the destination device due to the fact that it will not be able to chose a different route for each individual packet that is transmitted. This means that the overall process could be more expensive for the client.

From the above remarks it is evident that the user of an incentivised mesh network must make a trade-off between cost and performance.

Chapter 9

Conclusions

The aim of this research project was to investigate how blockchain and payment channel technology could be used to facilitate device-to-device micropayments and to do so by implementing the chosen use case of an incentivised mesh-network. In an incentivised mesh-network, client nodes are able to pay other nodes on the network for assisting in successfully delivering their packets to a gateway node that has direct internet access.

By using the Bitcoin cryptocurrency along with payment channel technology and a design similar to that used by the Lot49 implementation, it was possible to build an implementation of an incentivised mesh-network where devices could quickly and securely transact micropayments. In this implementation payments were settled between devices for each data packet routed through the network. An analysis of this solution showed that the computational overhead required for correctly making these payments is very large making the initial incentivised mesh-network implementation too slow for its intended purpose.

An investigation was made to try and improve the performance of the system. This was done by using a method called Shamir Secret Sharing to cryptographically link a single payment to multiple data packets. This improved the performance of the system significantly due to the fact that more data could be transmitted through the network without having to increase the computational overhead.

In the results chapter, Chapter 7, interpolation was used to estimate the performance of the final implementation given a mesh-network of a larger size. It was found that bandwidth requirements for popular internet applications such as sending emails or making VoIP calls could be met with the current implantation (using Raspberry Pi devices) for small mesh-networks that would not require routes of longer than 6 hops between the client and gateway nodes. The application of streaming videos, however, requires a minimum bandwidth of 0.5 Mbps which the Raspberry Pi implementation is not able to provide. Estimations were also made for the performance of a network using devices with higher MTU sizes as well as for devices with faster processors. From these estimations it is shown that an increase in MTU size improves the performance of the implementation enough to meet the minimum bandwidth required for video streaming for a small mesh-network where the route length does

not exceed 1 hop. However, using devices with faster processors improved the performance of the system significantly and it was estimated that all the bandwidth requirements for basic internet applications, including video streaming, could be met for a mesh-network where route lengths do not exceed 21 hops.

It can thus be concluded that using blockchain and payment channel technology does provide a viable way of enabling fast and secure device-to-device micropayments.

Chapter 10

Recommendations

The following recommendations are made for future work:

- Future work should investigate the bandwidth requirements necessary for the mesh-network nodes to broadcast funding transactions to the blockchain as well as query the blockchain for funding transactions.
- Further investigations should be done of the storage requirements necessary for the devices on the network to be able to partake in the incentivised mesh-network. For example, the devices will need to store multiple commitment transactions.
- Investigations should be done into ways of reducing the computational overhead of signing transactions as this would significantly improve the performance of the system.

Appendix A

Appendix

A.1 Code

A.1.1 Github Repository

All the code for this project can be found in the following Github repository link: https://github.com/ellemouton/Final_Year_Project/tree/master/Code/lightning/pi_specific. In the repository, folders called *A*, *B*, *C* and *D* can be found. Each folder contains a program for the specific node (e.g. "A.py") as well as a folder called *modules* containing various libraries used.

A.1.2 Code listings

```
1
2 def check_htlc_and_get_secret_hash(node, commitment_tx, channel):
3     tx_in = commitment_tx.tx_ins[0]
4     tx_out_1 = commitment_tx.tx_outs[1]
5     tx_out_2 = commitment_tx.tx_outs[2]
6
7     if(tx_in.prev_tx.hex() ==  channel.funding_tx.id()): #checks
    that it is spending from the correct 2-of-2 multisig tx (the
    funding tx)
8         if(tx_out_1.script_pubkey.cmds[2] == decode_base58(node.
    address) and tx_out_1.amount == channel.local_amt):
9             if(tx_out_2.script_pubkey.cmds[6] == hash160(node.
    public_key.sec())):
10                return tx_out_2.script_pubkey.cmds[2]
11     return None
```

Listing A.1: *check_htlc_and_get_secret_hash* function

Bibliography

[1] S. El-Hage and G. Holst. "Micropayments Between IoT Devices; A Qualitative Study Analyzing the Usability of DLT:s in an IoT Environment". In: (2018), p. 16.

[2] J.M. Nlong J.L.E.K. Fendji. "Rural Wireless Mesh Network: A Design Methodology". In: *International Journal of Communications, Network and System Science* 8 (2015), pp. 1–9.

[3] Kenny Li. *The Blockchain Scalability Problem and the Race for Visa-Like Transaction Speed*. 2019. URL: https://hackernoon.com/the-blockchain-scalability-problem-the-race-for-visa-like-transaction-speed-5cce48f9d44.

[4] Jimmy Song. *Programming Bitcoin*. ISBN 9781492031499. O'Reilly Media, 2017.

[5] Andreas M. Antonopoulos. *Mastering Bitcoin*. ISBN 9781491954386. O'Reilly Media, 2017.

[6] P. B. B. OBE. *The World Has Fallen Head over Heels for Elliptic Curve Cryptography*. 2019. URL: https://medium.com/asecuritysite-when-bob-met-alice/the-world-has-fallen-head-over-heels-for-elliptic-curve-cryptography-9aeb8ce674ce.

[7] *XOR Encryption*. 2019. URL: https://teambi0s.gitlab.io/bi0s-wiki/crypto/xor/.

[8] *Shamir Secret Sharing*. 2019. URL: https://cryptography.fandom.com/wiki/Shamir%5C%27s_Secret_Sharing.

[9] *Wireless mesh network*. 2019. URL: https://en.wikipedia.org/wiki/Wireless_mesh_network.

[10] J. Tremback et al. *Althea Whitepaper*. Tech. rep. Version 1.5. Althea, 2019. URL: https://althea.net/whitepaper.

[11] J. Ernst et al. *RightMesh Technical White paper*. Tech. rep. Version 5. RightMesh, Switzerland, 2018. URL: https://www.rightmesh.io/docs/RightMesh_TWP5.pdf.

[12] R. Myers. *Lot49: A lightweight protocol to incentivize mobile peer-to-peer communication*. Tech. rep. Version 0.8.5. Global Mesh Labs LLC, 2019. URL: https://global-mesh-labs.gitbook.io/lot49/.

[13] *Network Failure - MTU Size. - Raspberry Pi Forums*. 2013. URL: https://www.raspberrypi.org/forums/viewtopic.php?t=52807.

[14] Elle Mouton. *Final_Year_Project*. 2019. URL: https://github.com/ellemouton/Final_Year_Project/tree/master/Code/lightning/pi_specific.

[15] *Yet Another Bitcoin Testnet Faucet! Bech32!* URL: https://testnet-faucet.mempool.co/.

[16] *Blockstream transaction explorer*. URL: https://blockstream.info/testnet/tx/15fccae87a15395af0232ba7e1a5659a6d3ca67c90ebdf900025753fb6a57f3e.

[17] *What Is the Minimum Mbps Needed for Streaming?* URL: https://www.techwalla.com/articles/what-is-the-minimum-mbps-needed-for-streaming.

[18] *How Much Internet Speed do I REALLY need?* URL: https://www.otelco.com/how-much-internet-speed-do-i-really-need/.

[19] *VOIP Bandwidth Consumption*. URL: https://www.top10voiplist.com/bandwidth-consumption/.