Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Mesh networking with Bitcoin

Research Project

Renlord N. Yang

yangr@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Dr. Christian Decker
Prof. Dr. Roger Wattenhofer

February 18, 2016

Abstract

Mesh networks are gaining popularity as a means to provide internet connectivity to the public as the cost of deploying fixed network infrastructure increases. Despite an abundance of wireless devices available in the public, it is difficult to incentivise collaboration between devices due to lack of coordination and absence of trust. We introduce the Biternet network, a proof of concept for mesh networking with Bitcoin. Connected mesh nodes form a mesh network where each node runs the Biternet daemon and the OLSR daemon for routing purposes. Each node operates at a peer-to-peer level with no strict requirement that there must be access to the Internet. Bitcoin micropayment channels are used to facilitate micropayments, allowing mesh nodes to charge end users based on the data usage with a fine granularity.

Contents

Chapter 1

Introduction

Internet bandwidth by design is limited in capacity due to its reliance on fixed infrastructure which may only be deployed by vested interests such as governments, ISPs and large organisations. Fixed infrastructure which supports the Internet backbone is difficult to deploy and often costly due to the associated labour cost, land reclamation and materials involved. Furthermore, additional overhead must also be taken into account with the maintenance of the Internet backbone. With an ever increasing demand for bandwidth to stream high definition video contents and confererences, it is apparent that it is not possible for existing fixed infrastructure to keep up with demand. Current solutions to offer bandwidth to general users are prohibitive as they often require pre-registration, credit card details and trust in the system. For example, a traveller who is visiting Switzerland wanting to use a public WiFi network provided by a well-known internet service provider (ISP) would have to register her credit card details with said ISP. More often than not, this is a major barrier of entry for an end-user to utilize the service as international credit card payments are prohibitively expensive and it is also difficult for a traveller to register his/her personal details to obtain the right credentials to access the public WiFi network.

Therefore alternative means to provide bandwidth in parallel to existing infrastructure is necessary, and such an alternative solution should be trivial to access by end users requiring no more than a few clicks. While the main infrastructure backbone is designed to provide a fixed amount of bandwidth, it is possible to have residents or even businesses with extra bandwidth acting as nodes in a mesh network to cater for the fluctuating demand of network bandwidth by the general public. To allow for safe and anonymous transactions between mesh nodes, it is possible to utilize Bitcoins to remunerate services. Thus, removing the necessity to pre-register for usage as long as users and participating mesh nodes are in possession of Bitcoins.

As a solution to the aforementioned problems, we implemented *Biternet*, a software package consist of a daemon that implements the Biternet networking protocol at the application layer and uses the *OLSR* daemon to perform routing between nodes. The Biternet daemon also implements Bitcoin micropayment channels and a temporary wallet to fund the payment channels.

1.1 Mesh Network

An ad-hoc mesh network is a network that is continuously self-configuring and infrastructure-less. Connected devices may freely participate and leave the network at any time and nodes will subsequently update their routing tables accordingly to reflect the current state of connectivity within the network. Ad-hoc networking devices interact with each other at the Physical Layer in the OSI Architecture. Therefore, two ad-hoc devices may establish a point-to-point connection in the routing layer without an access point. As the topology of an ad hoc network is highly dynamic and highly dependant on the participation of networking devices, it is necessary to utilise an ad-hoc mesh routing algorithm to probe and create routes in an ad-hoc network.

For the purposes of this project, we have chosen to utilised the Optimized Link State Routing (OLSR) [1] protocol. OLSR achieves efficient routing by selecting the optimal path between two nodes. OLSR identifies potential paths by broadcasting “HELLO” messages to a specific port. Other mesh nodes running OLSR will also be able to accept and process the “HELLO” messages. These messages are used to discover one-hop and two-hop neighbours through their responses. Each mesh node then selects a multipoint relay based on the one hop node that offers the best route to adjacent two hop nodes. Furthermore, OLSR also uses topology control messages to disseminate neighbour information throughout the network. OLSRd is the daemon which runs the OLSR routing protocol. OLSRd allows mesh node owners to indicate if they are gateways to other networks. For example, if a mesh node was serving as a gateway to the wide area network (WAN), the mesh node may expose a route to the Internet through Host and Network Association (HNA) messages messages which indicates the subnets available through a particular mesh node. It is also possible to expose additional subnets with the same message to expand the Biternet network.

1.2 Bitcoin

Bitcoin [2] is a digital currency which utilizes a distributed ledger commonly known as a blockchain. Its distributed ledger and its consensus protocol were first introduced by Nakamoto Satoshi. It is a fully distributed system where the creation of new coins, transactions and verification of transactions may be performed collectively by the network without the need of a central authority. That being said, Bitcoin is a distributed system with eventual consistency characteristics as the blockchain may at times experience forks, where nodes would be minting on different branches of the blockchain. These temporary inconsistencies are eventually resolved when Bitcoin nodes receive and mine on the longest chain, thus achieving eventual consistency in the long run.

The Bitcoin blockchain is a distributed data structure which contains a series of blocks chained together like a tree. Every block within the blockchain is a descendent of the *genesis block*, which is the first block created by Satoshi to initialise the Blockchain. All subsequent blocks appended to the blockchain must contain a hash of the previous block thus guaranteeing the chronological order of blocks that are appended to the blockchain. Occasionally a *fork* in the blockchain may occur, a block has more than one block as its child. When this happens, the Blockchain is in an inconsistent state. Each block contains a block header and a series of transactions. The inclusion of a transaction in a particular block can be verified checking its header, therefore there is no need for a client to inspect the entire blockchain for the authenticity of a transaction. To append a block to the blockchain, the Bitcoin consensus protocol requires miners to contribute computational power to "mint" blocks. The act of minting is done by solving a mathematical puzzle which involves hashing a nonce and the merkle tree hash of the previous block until a hash with a specific number of leading zeroes as stipulated by the consensus network is found. The number of leading zeroes that will lead to find a block is the *difficulty* of minting the block as the actual difficulty of finding a hash with an increasing number of leading zeroes would be increasingly difficult just as well. The difficulty of minting is re-set for every 2016 blocks minted and is collectively determined by the entire Bitcoin consensus network. The current targeted minting duration for 2016 blocks is pegged at 14 days, therefore should the network be able to mint 2016 blocks in less than 14 days, the difficulty of minting will be increased. Otherwise, the difficulty will be reduced after all 2016 blocks are found.

Bitcoin transactions contain a list of inputs that consume unspent outputs from previous transactions and a list of new outputs. An *input* is a claim to an unspent output, it includes signatures and a reference to previous transaction. One or more inputs are included in a transaction to fund a transaction. Outputs are then used to spend the inputs included in a transaction. The difference between the sum of inputs and outputs are then paid to the successful miner who includes the transaction into a newly minted block. An *output* is analogous to a payment, it contains an output value and one of two redeem script types. For all intents and purposes, the Bitcoin network only accepts two types of transactions, *Pay-to-PubkeyHash* and *Pay-to-Script-Hash.* Complex transactions, for example smart contracts and time-locked transactions often have their scripts hashed, allowing a complicated transaction to be expressed as a simple *Pay-To-Script-Hash* transaction. When another user attempts to redeem the unspent outputs, the user will have to generate a matching hash which matches the script hash and thus be able to claim the unspent output in a different transaction. When constructing a transaction, a user will have to include a list of inputs which are also unspent outputs from a previous transaction. The combined sum of the inputs should be greater than or equal to the combined sum of the outputs, otherwise the transaction will be considered invalid by the Bitcoin network. Should the combined sum of the inputs be greater than the combined sum of outputs, the difference in sum of inputs and outputs would then be considered as *transaction fees* and will be paid to any successful miner that has minted the block which includes said transaction.

Public key cryptography is used to authenticate Bitcoin transactions. Private keys are stored in a file known as a *Bitcoin Wallet*. Typically, a public key is then derived from the private key. Private keys stored in wallets are used to sign Bitcoin transactions. A Bitcoin user may do this by consuming an unspent output, transforming it into an input of a transaction, then sign it with a private key which was used to derive the corresponding public key used in the redeem script. A *redeem script* is a set of instructions to be undertaken by Bitcoin nodes to validate a transaction that is attempting to consume an unspent output as an input in another transaction. Suppose that a private key and a public key used in an unspent output does not match, typically the signature used to sign the input will not pass validation, thus the Bitcoin network will reject the newly broadcasted transaction.

1.3 Micropayment Channel

Micropayments are typically transactions that involve a small sum that are considered to be infeasible to be paid due to the prohibitive cost in traditional payment methods. Bitcoin may be used to facilitate the processing of micropayments through a series of transactions that involve three phases collectively known Micropayment Channels proposed by Jeremy Spillman. To setup a micropayment channel, a commitment transaction which commits a sum to a shared account jointly held by a service provider and a consumer has to be created. Prior to signing and broadcasting the commitment transaction, a refund transaction which spends the output of the commitment transaction back to the consumer is required to guarantee that, should the provider fail to honour its commitments, the consumer will be able to re-claim her committed funds to the shared account. Typically, the refund transaction will be time-locked so it is essential that the consumer retain a raw signed refund transaction to broadcast in at a later time as the Bitcoin network would typically reject time-locked transactions until the time lock expires. Subsequent payments to the provider are done through payment transactions which spends the outputs of the commitment transaction. Payment transactions are only broadcast when the service has ended and its broadcast typically ends the micropayment channel as any remaining balance in the shared account will be refunded back to the consumer. A significant drawback of Jeremy Spillman micropayment channels is its inability to allow for additional deposits to a shared account. However, recent changes to the Bitcoin protocol which includes the introduction of `OP_CHECKLOCKTIMEVERIFY` [3] allows for subsequent deposits to a shared account as there is no need to setup refund transactions prior to commitment transactions.

1.4 Transaction Malleability

All transactions in Bitcoin prior to broadcast must be signed. While transactions are signed and the integrity of transactions are guaranteed, it is possible to invalidate a transaction by means of manipulating the byte stream containing the transaction hash. This has profound side-effects on the micropayment channels we have implemented for the project as it is possible for a malicious provider to invalidate the hash of the commitment transaction thus rendering the refund transaction unusable as it is reliant on the hash of the commitment transaction. This inherent vulnerability is caused by the fact that a transaction hash can be changed and thus invalidating all future transaction that rely on the current transaction hash. As a consequence of transaction malleability, it is unsafe to accept chains of unconfirmed transactions under any circumstance as later transactions that depend on the hashes of the previous transactions may change until they are confirmed in a block.

There exist proposed solutions to the transaction malleability problem and Segregated Witnesses is one of the solutions proposed that has already been implemented in a test Bitcoin network environment known as *segnet*. Segregated Witnesses [4] solve the problem by having signatures removed from Bitcoin transactions, since signature data is no longer part of the transaction hash, changes to how a transaction was signed would no longer be relevant to transaction identification. In the context of micropayment channels, this means that it will no longer be possible for a malicious service provider to invalidate a refund transaction by changing how a commitment transaction is signed, assuming Jeremy Spillman Micropayment Channels are used.

1.5 Related Work

The Bitcoin [2] protocol was pioneered and introduced by Satoshi Nakamoto in 2009, following rapid improvements by the community and significant research, Bitcoin improvement proposals were proposed to address vulnerabilities within the Bitcoin protocol. The transaction malleability vulnerability was highlighted by the Decker et. al. in their study of its impact on Mt. Gox [5] and several BIPs were proposed to address the transaction malleability vulnerability. Micropayment channels [6] were introduced by Jeremy Spillman and they featured the first trust-less mechanism to setup micropayment channels between two parties using transaction steps as described earlier. The introduction of `OP_CHECKLOCKTIMEVERIFY` by Peter Todd's BIP65 [3] enabled the establishment of Micropayment Channels which are immune to transaction malleability.

The optimized link state protocol [1] was proposed in 2003, it was then implemented by Tonnessen where OLSRd [7] is a daemon which runs the OLSR protocol. Following further development, the OLSR daemon was further extended to feature a full plugin library [8] and third party extensions were customisable and could be added through an API.

Chapter 2

Design & Implementation

In this section of the report, we describe how Biternet was designed and implemented. Biternet is designed to be a software stack which runs on any generic Linux distributions for Biternet for the purposes of this research project. The software stack includes a daemon which executes the Biternet protocol and the OLSR daemon which performs routing independent of the application daemon. By design, every node participating in the Biternet mesh network is required to run the software stack. The *Biternet* top layer daemon is responsible for setting up micropayment channels, seeking adjacent peers and all communication protocols between peers.

As a proof of concept, several wireless pre-configured mesh nodes running *Biternet* were used. Each mesh node was a Raspberry Pi Model B with Raspbian and the network devices were the TP-Link and Edimax EW7811UN wireless dongles. They were used to setup WiFi access points, internet gateways and ad-hoc connections between mesh nodes. Biternet follows a peer-to-peer model where the mesh node is capable of acting as a server, relay or a client at the same time. By default configuration, all Biternet mesh nodes are relay nodes, the provision of client services is left as a choice to the owner of said mesh node.

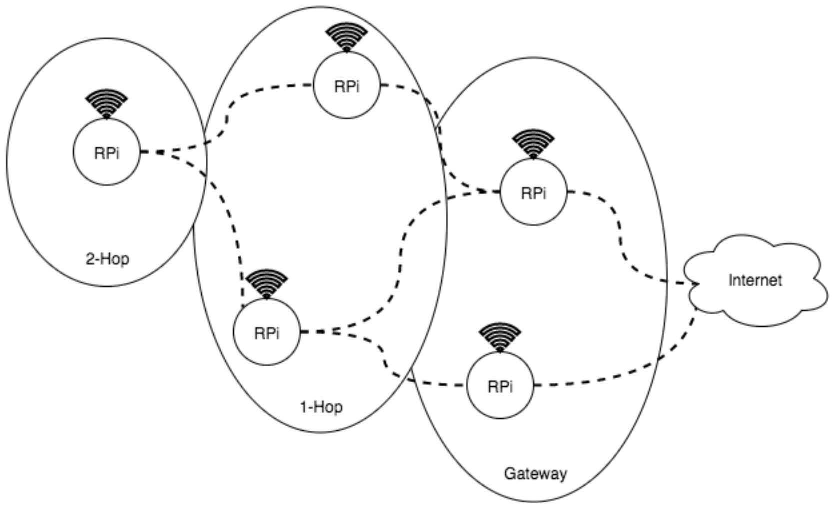


Figure 2.1: Simple Biternet Mesh Network with Wireless Access Points and Internet Gateways

2.1 Software Stack

Biternet features a complete software stack which allows mesh nodes to choose between becoming a full service provider that exposes an endpoint for end users to use the Biternet network or act as a relay within the Biternet network as illustrated in the diagram below. Should a mesh node elect to expose a WiFi Access Point, a mesh node will need to have at least two network devices with one interface perform ad-hoc networking for OLSRd and the other for broadcasting the WiFi Access Point. Similarly, the number of network devices increases according to the number of services a participating node wishes to provide. The sole requirement of participation in Biternet is a network device which perform ad-hoc networking, otherwise every other network interfaces are optional.

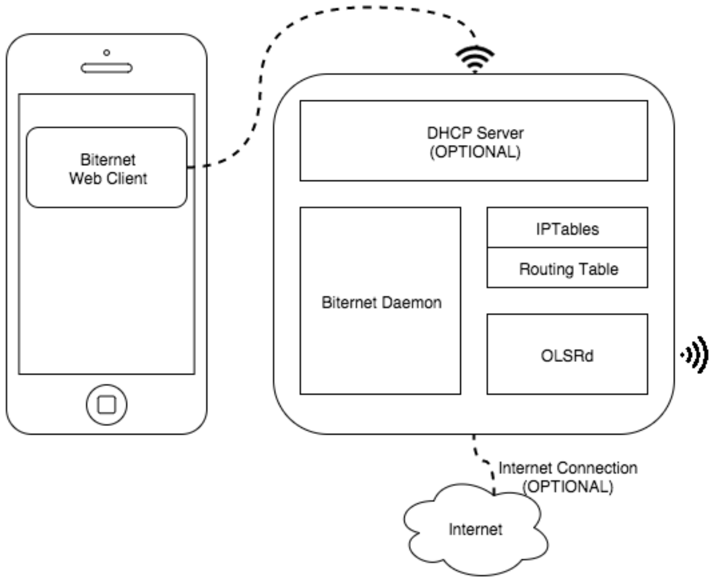


Figure 2.2: Biternet Software Architecture

2.1.1 Biternet Daemon

The Biternet daemon is designed for flexibility. Owners of the mesh nodes run the Biternet daemon and are free to configure as many parameters as possible to encourage competition between mesh nodes to strive to provide the cheapest, optimal links to adjacent nodes. Mesh nodes may also expose subnets by configuring their OLSR daemon to inform adjacent nodes of existing subnets exposed by their mesh nodes. The Biternet daemon supports peer to peer communication through the use of Web Sockets. Mesh nodes that will be shutting down typically provide a grace period of 60 seconds for dependent mesh nodes to finalise and broadcast payment transactions before internet connectivity is disabled. Everytime dependent mesh nodes relying on one particular Internet Gateway node that is shutting down would have to tear down all existing payment channels with its users and attempt to find a new internet gateway. Once a new internet gateway is found, a series of payment channels will be set up with the adjacent nodes again. Each Biternet daemon also contains a temporary wallet which is required to be funded by the owner of the mesh node. Biternet daemons do not choose what nodes they connect to as this is being done by OLSRd instead, the daemon is only responsible for the establishment of micropayment channels, accounting traffic usage and to communicate between mesh each time an event has occured.

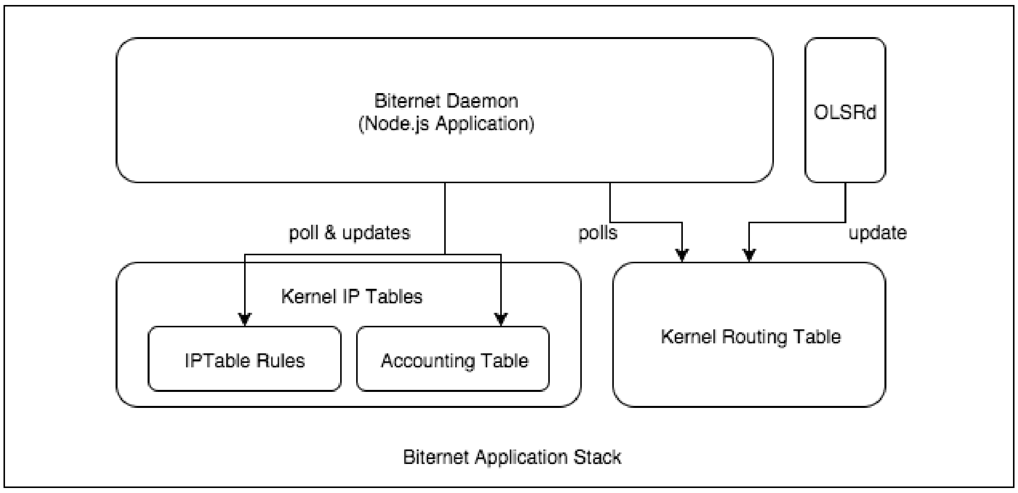


Figure 2.3: Biternet Daemon Application Stack

The Biternet daemon communicates at the application layer with a specially designed protocol which allows mesh nodes to setup micropayment channels between nodes automatically. Typically, a mesh node which exposes a gateway to the Internet would remain dormant and not proactively seek to intiate connections with adjacent Biternet mesh nodes. Mesh nodes that do not have a gateway to the Internet would proactive seek to connect to adjacent Biternet mesh nodes with a route to an Internet gateway. When such an adjacent node is found, it will ask for an advertisement to obtain the terms of service of the adjacent mesh node. Subsequently, the mesh node will establish a micropayment channel with the adjacent mesh node. Once a micropayment channel is established, mesh nodes will forward traffic so as long as the connecting mesh node continues to pay invoice messages that are sent by the mesh node providing a relay service. When a mesh node terminates its service, a shut down message is sent to all adjacent nodes connected to it. If said mesh node is acting as an Internet Gateway, the shutdown message will be propogated to all mesh nodes that are linked in a series of hops to the Internet gateway, this guarantees that all mesh nodes will eventually have an opportunity to end their micropayment channels appropriate by broadcasting the latest payment transaction in their current states.

2.1.2 Discovery and Routing

Prior to running the Biternet client, mesh node users will need to manually configure a static IP address that is unique within the Biternet network. Ideally, we would want all mesh nodes to have self-configuring IP addresses that may be assigned by utilizing IPv6 Stateless Address Autoconfiguration. By doing so, mesh nodes will be able to dynamically obtain an IP Address without a central authority delegating IP addresses.

Expanding the Biternet network is done by means of discovering adjacent nodes who run the Biternet application. Biternet applications by default run a routing daemon called OLSRd - Optimized Link State Routing Daemon. OLSRd periodically updates the IP Routing Tables to choose optimal paths. Optimality in the context of Biternet is calculated as a function of cost to relay a kilobyte of data to the Internet and the link quality to the adjacent node. While it is possible to configure the path cost through OLSRD, mesh node users running Biternet will have to manually synchronize the path cost configured for the OLSRd configurations and the indicative path cost advertised by the Biternet application. Such a limitation exists because both OLSRd and the Biternet application are designed to be run seperately. Although mesh nodes will only connect to adjacent nodes, it is still possible for mesh nodes to connect to other nodes, the connected adjacent nodes simply act as gateways to other mesh nodes within the same subnet.

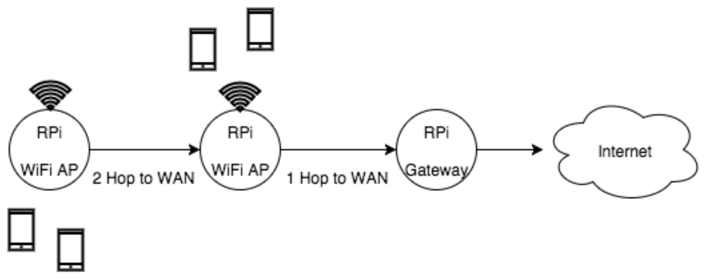


Figure 2.4: Simple example illustrating 1-Hop and 2-Hop mesh topology

2.1.3 Access Control

Running an instance of Biternet will automatically invoke the installation of various IPTables rules which will throttle and limit access by other mesh nodes. IPTables is a firewall utility bundled with most if not all Linux distributions. It is often used to filter internet traffic and protect connected computers from network exploits. For the purposes of this project, IPTables were used as a means to log traffic usage, packet redirection for captive portals and access control.

Each time a client is connected to the Biternet network and has successfully established a payment channel with a provider mesh node, it will be included in a whitelist accounting table where packets will be forwarded without restriction. The Biternet application also allows mesh node users to configure if they would like to charge for both upstream and downstream traffic or uni-directional traffic. By default, the Biternet application only charges for downstream traffic as it is perfectly possible for a malicious node to account for upstream traffic which may not necessarily be forwarded to its destination. As a consequence, a client may unknowingly be retrying requests too often while getting charged for doing so. Clients who are connected to the Biternet network but has yet established a payment channel will be denied relay services as IPTables are used to block all forwarding traffic until payment channels are established. However, to facilliate Bitcoin transactions, well-known bitcoin API services are whitelisted and SPV client ports are also whitelisted to allow for Bitcoin Wallets to fund payment channels and broadcast transactions.

2.1.4 Usage Metering

To be able to accurate charge correct amounts of bitcoins from client mesh nodes and end users utilizing the Biternet network. IPTables are used to measure upstream and downstream data usage based on IP Addresses. The Biternet application allows mesh node owners to configure the periodicity to poll the IPTables for updates and in each polling interval, invoices will be sent out to client mesh nodes for payment. If client mesh nodes or users fail to provide a partially signed payment transaction within the time interval stipulated in the usage agreement, filtering rules will be re-instated for such clients and users so that they may no longer be able to connect to the Internet through the Biternet network.

2.2 Protocols

Biternet has a protocol for connecting and setting up payment channels between mesh nodes. Every Biternet daemon implements a peer-to-peer architecture, therefore they may both initiate and receive contact from an adjacent peer node.

2.2.1 Micropayment Channels

To formally set up a payment channel with an adjacent mesh node, the following payment channel negotiation protocol is used:

1. The consumer node contacts the provider node with a `GET Advertisement` message.
2. The provider node then replies with an `Advertisement` message which includes the terms of the micropayment channel and all necessary information to set up transactions.
3. If the consumer node is satisfied with the terms of the payment channel, the consumer node can then send a `Acceptance` message with includes its public key and all other bitcoin specific transactions for micropayment channels.
4. Otherwise, the consumer node can choose to ignore the `Advertisement`.
5. This concludes the payment channel negotiation protocol. Should the consumer node accepts the terms, then the mesh node proceeds to execute steps to set up a Bitcoin micropayment channel which is further elaborated in the implementation section of this report.

Our micropayment channels were implemented from scratch as there were no existing usable micropayment channel libraries available for the web. A new javascript library that is usable on all development platforms (i.e., Native and Web) was produced as a result of this project known as *btc-micropayment-channels* and is currently hosted in the Node Package Manager which is freely auditable and accessible by the public. The micropayment channel library is built on a wrapper around Bitcoin low level functions Javascript library known as *bitcoinjs-lib*. The micropayment channels are implemented within the Biternet daemon. For the purposes of the project, we implemented Jeremy Spillman's style micropayment channels, thus it would be vulnerable to transaction mallaebility. Due to time constraints, we could not implement micropayment channels utilizing `OP_CHECKLOCKTIMEVERIFY` that would have been invulnerable to transaction mallaebility.

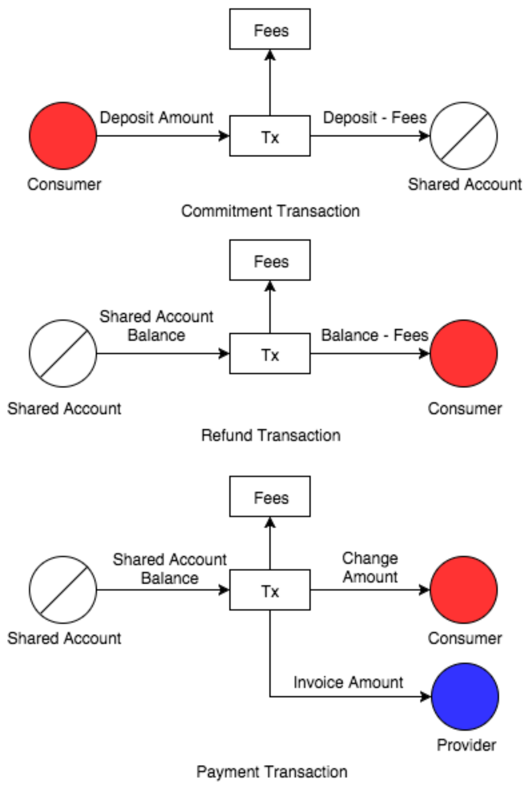


Figure 2.5: Micropayment Channel Transactions

To initiate micropayment channels, we first prepare a commitment transaction which commits a deposit amount to a shared account that spends an output to a 2-of-2 multisig address. To redeem the 2-of-2 multisig unspent output, the transaction spending the committed amount would require both the provider and the consumer's signature. Before the commitment transaction is broadcasted, it is necessary for the provider to sign a prepared refund transaction which spends the deposit which is committed to the shared account. This is to guarantee that should the provider renege on her service, the consumer would still be able to recover committed funds to the shared account without the provider's assistance. In our implementation, once the commitment transaction has been prepared on client side, the deposit amount and the consumer public key is then communicated to the provider, so that the provider may sign a refund transactions which spends the deposited amount back to the consumer. After the signed refund transaction is received from the provider, the consumer then sends a signed commitment transaction which the provider verifies, then broadcasts to the Bitcoin network. Subsequently, the provider will periodically issue invoice messages to the consumer to request for payment. Should the consumer renege on payment requests, the provider can easily terminate the service and tear down the micropayment channel. Otherwise, incremental payments will be made to the provider and once the consumer is done using the service, the provider can then broadcast a final payment transaction to the Bitcoin network which spends the total amount paid to the provider and returns the remaining balance to the consumer.

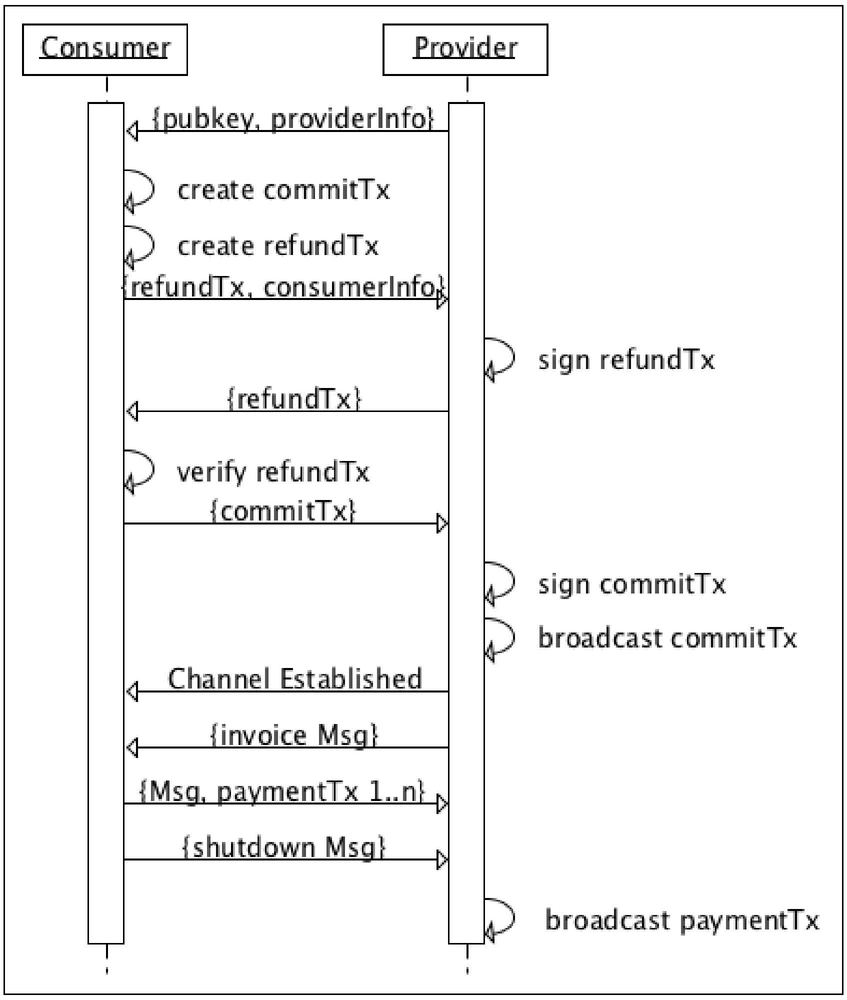


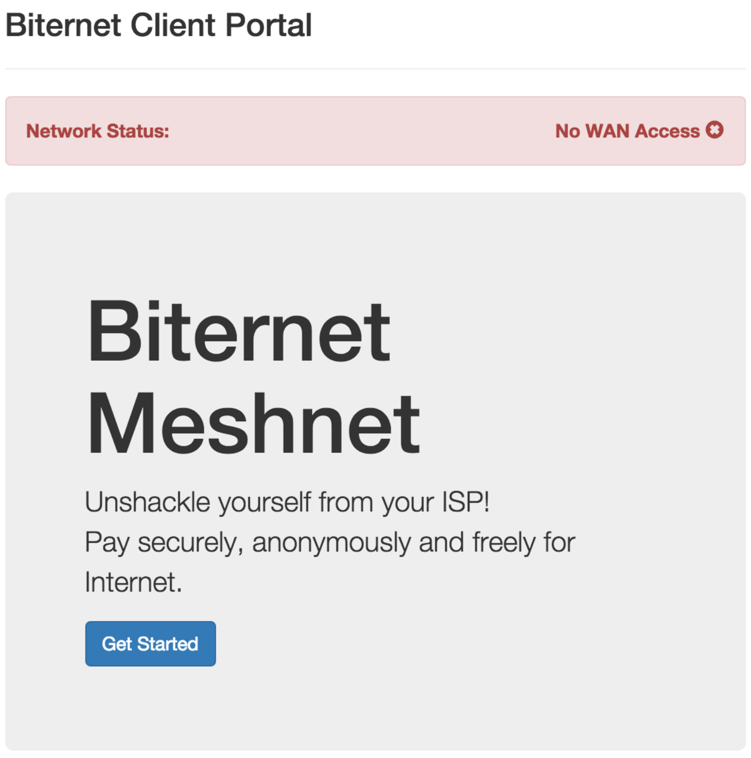
Figure 2.6: Micropayment Channel Sequence Diagram

2.2.2 Application P2P Messaging

As in most other protocol based applications, Biternet also implements peer-to-peer messaging to allow peers within the network to announce their impending shutdown. A mesh node should fully exit the Biternet network after a full 60 seconds to allow other reliant mesh nodes to terminate existing payment channels and broadcast necessary transactions. Currently, the only messages the Biternet network handle are OLSRd routing messages and protocol-specific messages (e.g, shutdown messages and advertisement messages). All protocol-specific messages are sent in Javascript Object Notation (JSON) message format for generic message parsing.

2.3 Web Client

To facillitate ease of access to the Biternet network, each mesh node which provides a WiFi Access Point hosts a web client which serves as a captive portal. When a user connects to a mesh node access point, the user is re-directed to a captive portal which requires her to set up a payment channel. The Biternet web application also implements the exact same peer-to-peer protocol as the one used in the daemon. A simple illustration of the web client in the diagram below and further implementation details may be sought in the implementation section of the report.


Biternet Client Portal

Network Status: No WAN Access

Biternet Meshnet

Unshackle yourself from your ISP!
Pay securely, anonymously and freely for Internet.

Figure 2.7: Biternet Web Client

Steps for an end-user to connect to the Biternet network:

1. User connects to a Biternet WiFi Access Point.
2. User receives an IP address from the DHCP server of the provider node.
3. Captive portal pops up in user's device and prompts for micropayment channel initiation.

4. If user accepts the terms of service, user then funds the micropayment channel and Internet access will be available.

5. Otherwise, user will be denied access to the Internet until user chooses to fund micropayment channel.

2.3.1 Biternet Web Application

The Biternet web application is fully functional without a server and may be run on both mobile devices and desktop. Upon instantiation, the web application will randomly generate a private key pair client-side which can be used as a temporary wallet used to store Bitcoins that the end user would like to pledge to fund a micropayment channel. As no local storage is implemented, a wallet import format hex string is generated when the user is attempting to fund the temporary wallet so that the user may safely recover the wallet should the browser refresh or exit. Once the temporary wallet on the Web Client is funded, the web application then initiates the establishment of a micropayment channel with the DHCP server that is hosting the WiFi network for the client, commiting all funds available on the temporary wallet to the micropayment channel. Once the micropayment channel is established, the user may use Internet service or connect to subnet devices within the Biternet network. In the off chance that connectivity with the DHCP server is lost or WAN connectivity is lost after funds are committed to a shared account, the web application still retains the ability to broadcast a refund transaction as long as the browser tab containing the web application is not refreshed nor terminated. The user may easily move on to a different WiFi hotspot or even utilize mobile internet tethering to broadcast the refund transaction to the Bitcoin network to reclaim funds committed to the shared account as long as no payment to the provider has been made.

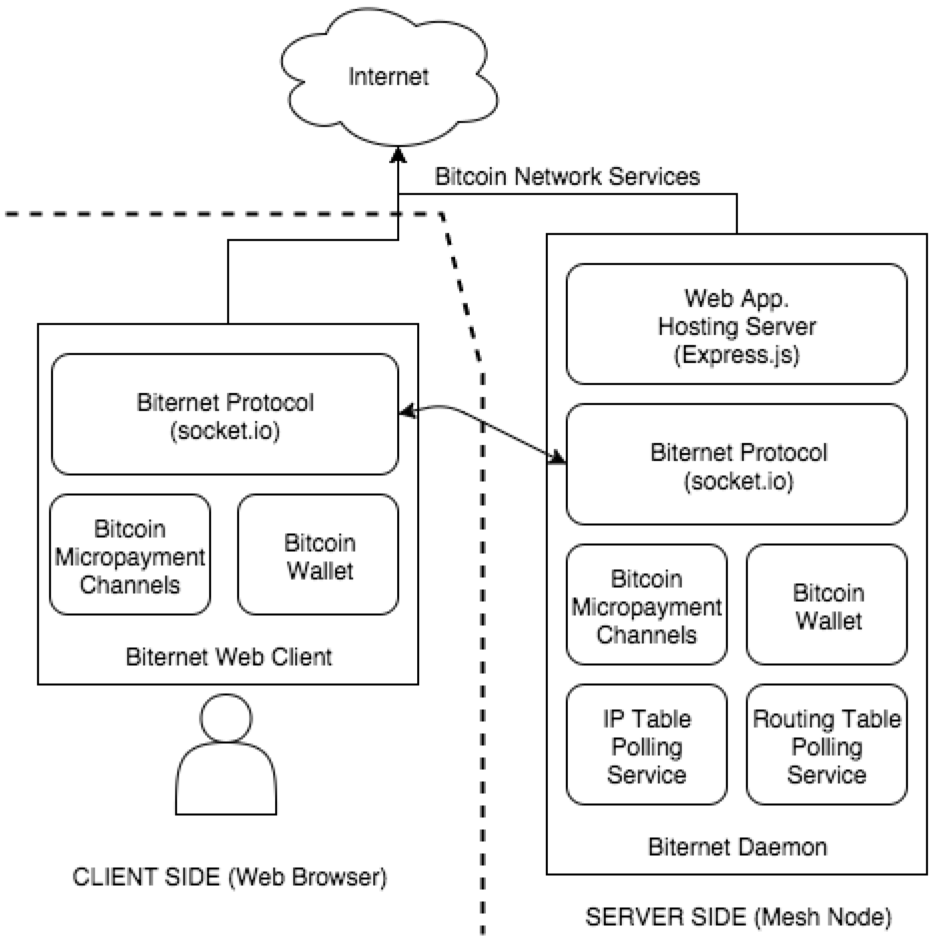

Internet
Bitcoin Network Services
Biternet Protocol (socket.io)
Bitcoin Micropayment Channels
Bitcoin Wallet
Biternet Web Client
CLIENT SIDE (Web Browser)
Web App. Hosting Server (Express.js)
Biternet Protocol (socket.io)
Bitcoin Micropayment Channels
Bitcoin Wallet
IP Table Polling Service
Routing Table Polling Service
Biternet Daemon
SERVER SIDE (Mesh Node)

Figure 2.8: Communication between web app and mesh node

CHAPTER 3

Evaluation

In this section of the report, we evaluate the performance metrics of our implementation for Biternet. In the following sections, we measured the time it takes for a route to be built and the throughput of the Biternet network with respect to an increasing number of hops between mesh nodes to the internet gateway. We also did a thorough analysis on the known problems and potential vulnerabilities of the system.

3.1 Route Build Time

To measure the route build time between mesh nodes, we have setup our experiment to measure the time taken for OLSRd to build a route to the nearest mesh node which provides a gateway to a default route. For N-Hop configurations where N is greater than 1, we measure the time taken to build a route to the adjacent mesh node which acts as a relay to the mesh node providing an internet gateway. To obtain the results for this experiment, we have set up the mesh nodes in the following configuration:

- 1-Hop Configuration. A mesh node that exposes a wireless access point (AP) is linked to a mesh node which provides internet access through wired interface.
- 2-Hop Configuration. A new mesh node that exposes a wireless access point (AP) is linked to a mesh node as described in the 1-hop configuration.
- N-Hop Configuration. For each increasing N-hop configuration, one additional mesh node that exposes a wireless AP is added and is linked to the outer most mesh node that exposes a wireless AP in the $(N-1)$-hop configuration.

Due to hardware and spatial constraints, we were only able to test up to 3-Hop configurations.

The experiment was conducted as follow:

1. We set up a mesh node which provides a gateway to the internet.
2. We placed each mesh node required for the N-hop configuration in a specific physical location where the number of connectible mesh nodes is constrained to just one.
3. For each N-Hop configuration, we first turn on hardware interface which performs ad-hoc networking on the N-th furthest mesh node from the mesh node which provides the internet gateway.
4. A job script is also set up on the reference mesh node (i.e., the mesh node furthest away from the mesh node providing an internet gateway) to time the duration it takes to observe a default route entry in the kernel routing table. As it was not possible to register an event to trigger the stopping of the timing script, the timing script periodically polls the kernel routing table every 100 milliseconds to check if a default route has been inserted into the kernel routing table.
5. Once an entry to a default route is inserted into the kernel routing table, we know that the reference mesh node has built a route to the mesh node providing an internet gateway.
6. We repeat the experiment 10 times for each N-hop configuration to normalize the errors attributed to systematic errors in our experimental design.

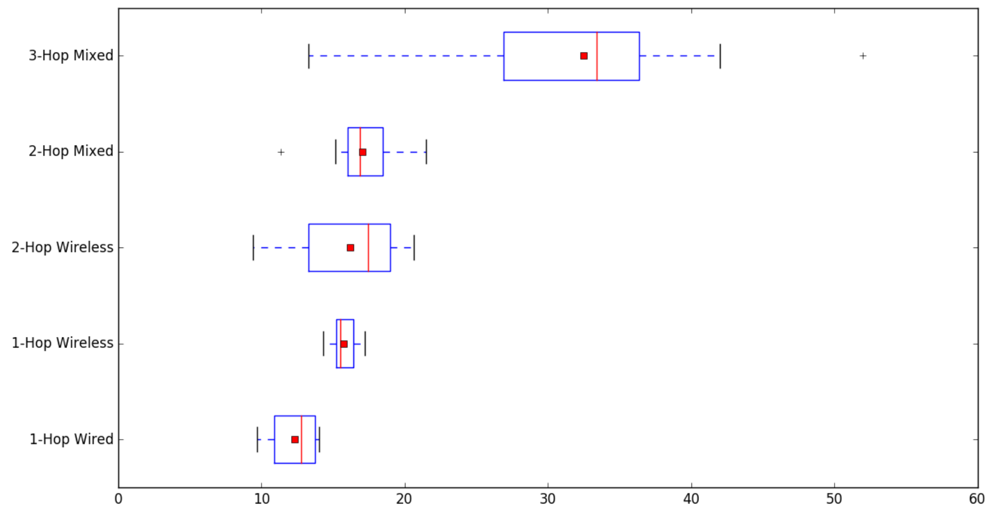


Figure 3.1: Time required to connect N-th mesh node to $N-1$th Relay Node in N-Hop Configuration (in seconds)

Results Based on our findings, we observed that wired configurations would yield more consistent time measurements compared to mixed and wireless configurations. This is apparent through the lower median time requirement compared to both wireless and mixed configurations. It is also apparent that as the number of hops increases, the time required to set up connect the N-th mesh node to a $N-1$th relay node increases significantly as demonstrated in the diagram above with the increasing median and mean time. The standard deviation of the data set also increases as the number of hops in a configuration increases. The data also present some significant outliers in N-Hop configurations where N is greater than 1. Extreme outliers greater than 60 seconds observed in the 3-Hop configuration dataset were removed in order to provide a sound comparison between N-Hop configurations.

Discussion The increasing median time observed in the data set is within expectation as the complexity of the network topology would increase as the number of hops increases. An increase in the complexity of the network topology would significantly increase the necessity of topology control messages. As a consequence of that, OLSRd would have to process more messages and will require more time to process them to select an optimal route to a mesh node providing an internet gateway. The dramatic increment of the standard deviation however was unexpected as the experiment was set in a controlled environment and the physical topology of the mesh nodes were configured in such a way where each mesh node would only have one adjacent mesh node to connect to, therefore we expected that OLSRd should have been able to select the adjacent mesh node consistently. However, given that the experiment was run in a building with many wireless devices, the significant variance in the data set could have been attributed to wireless interference from other devices causing OLSR topology control messages to be dropped resulting in an inconsistent route build time for configurations with more hops.

3.2 Route Swap & Seek Time

Ad-hoc networks are inherently unstable as their topology changes frequently. In this section of the experiment, we measure the time it takes for OLSRd to swap between routes which enables it to connect to a mesh node which provides an internet gateway. We also measure the time it takes for OLSRd to discover a new internet gateway should an existing internet gateway go offline. Due to hardware limitations, we were only able to test this in a 3-Hop configuration where there are two intermediary relay nodes which provides a link to the mesh node with an internet gateway. The procedures for the experiment are as follow:

1. Two mesh nodes which provides a gateway to the internet was set up.
2. Two mesh nodes acting as an intermediary relay between a mesh node with a Wireless AP and the mesh node with an internet gateway were set up. Both intermediary relays do not share the same mesh node which provides an internet gateway.
3. Finally, a mesh node with a Wireless AP which exposes the Biternet network to end users was set up.
4. Measurements were taken in a SSH session in the mesh node with the Wireless AP to measure the time taken to swap between intermediary relays which provides connectivity to the mesh node with an internet gateway.
5. The experiment was repeated 10 times to normalize errors attributed to systematic errors in our experimental design.

To measure the seek time to connect to a mesh node with an internet gateway when connection with an existing mesh node provider an internet gateway is severed. The following configuration was set up for the seek time experiment:

1. One mesh node acting as the reference mesh node was set up with a Wireless AP. This is the mesh node used to measure the time it takes to regain internet connectivity.

2. Another mesh node acting as an intermediary relay between the mesh nodes providing internet service was set up. The intermediary relay was connected to the reference mesh node with wired interface.

3. Two mesh nodes with internet connectivity were set up.

4. To run the experiment, the mesh node providing an existing internet service was switched off by a test runner script on the reference mesh node. The job script then polls the kernel IP routing table until an entry with default destination is entered. The time taken for the entry is then logged.

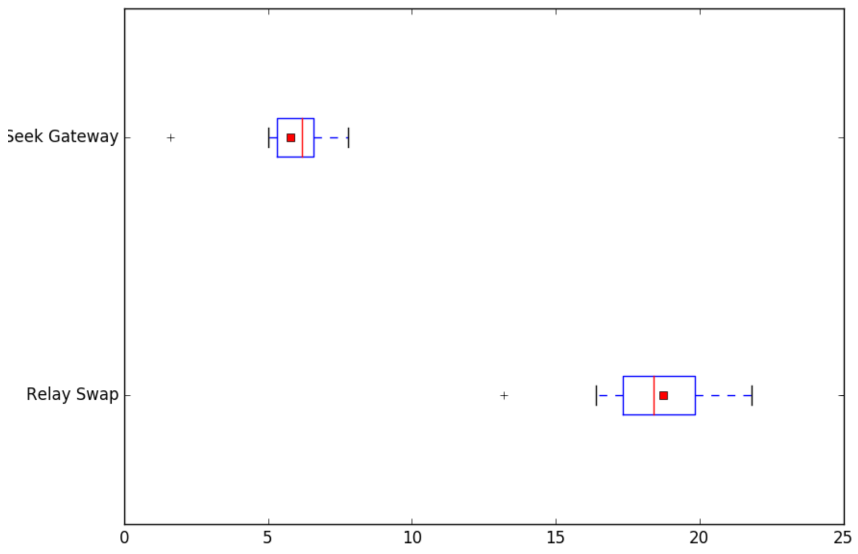


Figure 3.2: Swap & seek time (seconds)

Results Our results show that the cost of changing intemediary relay nodes to a common mesh node with an internet gateway is significantly higher than the cost of directly connecting to a mesh node with an internet gateway. We also observed that the required time to create a route to an adjacent mesh node with an internet gateway is signficantly more consistent than creating a route to the internet through an intermediary relay node.

Discussion By default, OLSRd configures the neighbour set holding time to 6 seconds, therefore should a link be broken down, it would take OLSRd at least 6 seconds to realise that a link is completely lost. This is consistent with the measurements taken for the time taken for the reference mesh node to connect to a gateway node through an intermediary relay node. Since the intermediary node has to update its 1-hop neighbour time, it will take a minimum of 6 seconds for OLSRd to realise an existing link is lost, then update to a new gateway node. Once this is done, the intermediary relay node can then broadcast a Topology Control message which allows the reference node to update its 2-hop neighbour table immediately. Thus, a new route to the gateway node may be found relatively quickly in less than 10 seconds. While the measurements do deviate quite significantly from the route build time measurements, this maybe attributed to the difference in the background interferrence as the measurements for this particular experiment was done during off-peak hours where there would be significantly less signal interference.

Building a route to a gateway node through a newly connected relay node inherently cost more time due to the fact that it would take more time to update neighbour sets across two nodes along the path. This is caused by the fact that the relay node would have to been selected as a multipoint relay by the gateway node prior to it being able to forward OLSR messages from the gateway node through to the reference node. The inherent communication overhead for this is estimated to approximately 12 seconds as an election for an MPR would have to take place when the new relay node was chosen as the relay between the reference node and the new gateway node. An *Midpoint Relay (MPR)* is a node selected by OLSRd to serve as a midpoint relay to reduce message flooding in the mesh network. While it is quick for an asymmetric route to be built between the reference node and the gateway node as the relay node would have already been selected as a MPR in the OLSR daemon for the gateway node. The newly connected relay node would have to undergo a selection process in the reference node and this is the cause of the time overhead compared to a direct connection to a gateway node. Once the relay node is selected as a MPR by the reference node, it will require more time for the relay node to forward OLSR messages to the gateway node so that a symmetric link could be established, which is a pre-requisite for entry in the kernel routing tables.

3.3 Known Problems and Vulnerabilities

As of the current build version, Biternet will throw an error and exit when its wallet has run out of funds. As a consequence of that, should other mesh nodes be reliant on a mesh node that has run out of funds, all micropayment channels and connections to the mesh node that has run out of funds would be severed instantaneously. As there are no means of topping up a micropayment channel without running the entire establishment procedure from the start, it is currently not possible to top up an established micropayment channel. Jeremy Spillman micropayment channels are also inherently vulnerable to transaction mallaebility as discussed in the introduction section of this report due to the fact that the refund transaction is reliant on the hash of the commitment transaction.

Suppose mesh nodes lose connectivity to the internet whilst providing service to end users who are connected to the Biternet network. The nodes will lose all ability to commit transactions to the Bitcoin network. Whilst this is not a fatal problem that would terminate the Biternet daemon, this would prevent any Biternet daemon from allowing end users to connect to the Biternet network as it is necessary to establish a micropayment channel before gaining access to the Biternet network. Existing users who have already established micropayment channels would not be affected by the lack of internet connnectivity and would be able to use the Biternet network continuously until the micropayment channel runs out of funds.

The Biternet web client currently does not implement stateful memory nor does it cache its state. Therefore should the user refresh or exit the web application by accident, it is not possible to continue the existing established micropayment channel and it would be necessary to establish a new micropayment channel. Furthermore, the Biternet package currently relies on third party Bitcoin block explorer services, therefore there it is necessary to trust a third party service and the availability of the Biternet service is also reliant on the availability of the third party block explorer service. Ultimately, it is possible to substitute the reliance on third party block explorer services with an implementation of a SPV client, thus allowing mesh nodes and web client is independently verify transactions on the blockchain without relying on third party services. Regardless, the usage of SPV clients is still conditional on the provision of internet service by the Biternet network, if there is no internet gateway, there is no way a SPV client could verify and broadcast transactions.

Biternet currently requires mesh node owners to statically configure non conflicting IP addresses for each mesh node. Ideally, Biternet mesh nodes should run IPv6 stateless auto-configuration to automatically gain a non-conflicting IPv6 address within the network. However, due to significant time constraints and the incompatibility of OLSRd to work with IPv6 addresses, this has been omitted from the design of Biternet. As of the current implement, Biternet will experience conflicting IP addresses within the same network should nodes be configured incorrectly. As a consequence, the ad-hoc mesh networking may fail occasionally as the network interface device may broadcast on different subnets.

Chapter 4

Conclusion

We implemented a proof of concept (Biternet) where several mesh nodes were able to to collectively provide end users with internet connectivity in exchange for payments in bitcoins. Biternet is simple and easy to configure for mesh node owners and may easily scale depending on the hardware used to provide internet service. Furthermore, it is also possible for Biternet to supply content from subnets within the network.

There are many avenues for future improvement. Future iterations of Biternet will remove the necessity to configure two separate applications (i.e., Biternet and OLSRd). Followed by the introduction of the usage of OP_CHECKLOCKTIMEVERIFY in commitment transactions to remove the necessity of pre-signed refund transactions that are reliant on commitment transactions, thus eliminating the possibility of transaction mallaebility. Finally, followed by an implementation of a proper plugin for OLSRd to factor in link cost in terms of monetary value.

Bibliography

[1] Clausen, T., Jacquet, P.: Optimized link state routing protocol (OLSR). Technical report (2003)

[2] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. (2008)

[3] Todd, P.: OP_CHECKLOCKTIMEVERIFY. https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki (Oct 2014)

[4] Eric Lombrozo, Johnson Lau, P.W.: Segregated Witness (Consensus layer). https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki (Dec 2015)

[5] Decker, C., Wattenhofer, R.: Bitcoin transaction malleability and MtGox. In: Computer Security - ESORICS 2014. Springer (2014) 313–326

[6] Spillman, J.: Micropayment channels. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html (Apr 2013)

[7] Tonnesen, A.: OLSRd: Implementation code of the OLSR. Available on line at http://www. olsr. org

[8] Hafslund, A., Tønnesen, A., Rotvik, R.B., Andersson, J., Kure, Ø.: Secure extension to the OLSR protocol. In: Proceedings of the OLSR Interop and Workshop, San Diego. (2004)