Muhammad Rameez

**Raiden Reaserch**

# 

Table of Contents

[1. 1](#_Toc497263533)

[2. Introduction 3](#_Toc497263534)

[3. References 3](#_Toc497263535)

# Introduction

* Getting started with raiden extremely hard
* Syncing with ropsten test net faucet
* Always on pc really required. Can always copy chain data off of it.

# Goals

* ~~See how to write ERC20 tokens and register them with raiden~~
* ~~Write a small application using and transferring ERC20 tokens between two raiden machines~~
* Also check how to route payments through a third machine i.e. node discovery, routing etc functions need to be checked
* Write wrapper contract to treat ETH like an ERC20 token so it can be used in raiden
* Also provide prepackaged function calls for creating ERC20 tokens in ur IoT interface
* Search telehash and MQTT

# Things to do tomorrow

* Read the following
  + <https://www.youtube.com/watch?v=1wAbCnD-M_I>
  + <https://raiden.network/faq.html>
  + <https://raiden.network/101.html>
  + <https://github.com/raiden-network/raiden/wiki/Raiden-PoC%E2%80%900>

# Micro Raiden & Raidos

“**What is μRaiden?**

μRaiden (Micro Raiden) shares some properties with the Raiden Network. It can provide trustless, instant and free transfers between two parties. It is intended for many-to-one payment setups, like users interacting with a Dapp. However, it is not suitable for many-to-many payment setups as it requires users to lock up tokens upfront for every potential payee. This limitation comes with reduced technological complexity, allowing µRaiden to be used on the mainnet today.”

* Check out if microraiden can be used in our IoT interface or if we need normal raiden.
* Microraiden is many to one platform

# Work done so far

* ~~Deploying raiden, was also very hard lots of implicit requirements and broken dependencies.~~
* ~~Lots of bug fixes with help of raiden developers and testing with them to proof everything is working~~
* ~~Faced a lot of problems with channel opening due to bugs in raiden code~~
* ~~Fixed the bugs and tested channel opening and raiden API function e.g close , settle, swap etc~~
* ~~Wrote multiple ERC20 Token and tested them, Two broad categories minteable and fixed supply tokens~~
* ~~Wrote ERC mintable tokens~~
* ~~Fixed channel opening problems~~
* ~~Researched difference between settle and close~~
* ~~Tried sending tokens from counter party doesn’t work too~~

# Problems with token transfer

* In order to make token transfers the two nodes need to see each other I,e, ping and acks should reach each other. Issue created

<https://github.com/raiden-network/raiden/issues/1020#issuecomment-329534169>

<https://github.com/raiden-network/raiden/issues/1086#issuecomment-337491680>

<https://github.com/raiden-network/raiden/issues/1133>

* Raidens Nat punching module is not perfect has bugs and doesn’t work well with all networks or firewalls.
* For now whoever got it to work have been by turning off every firewall in the entire network and or allowing all types of udp traffic through routers

# Fixes tried for token transfer

* Tried several solutions, worked very closely with Raiden developers
* Also raiden should be run under its own virtualenv
* Several sleepless nights spent and many possible causes eliminated

# Future work

* Order raspberry pi’s as it will have problems too, imp to start early
* Also check how to route payments through a third machine i.e. node discovery, routing etc functions need to be checked
* Write Raiden research document with the following
  + What is channel manager contract
  + What is netting channel
  + i.e. architecture settle ,close, fund etc
  + Document problems faced in the raiden document
  + Bouncy castle research

# Telehash vs MQTT

## MQTT

### Introduction

MQTT is a very light weight messaging protocol for IoT and machine to machine communication. It is designed as a publish/subscribe protocol to operate over constrained devices, and unreliable networks. The protocol uses client server architecture described as broker and client in official MQTT documentation. Some key feature of the broker and clients are described below.

* MQTT brokers are light weight and can be run on a raspberry pi.
* Client IoT devices subscribe to Topics and publish data to topics.
* All IoT devices connected to a single broker can share data.

pub

Client

Broker

oker

Sub

Figure 1

### Basic Model

MQTT clients or devices can register for topics they are interested in with a broker. Devices which publish data publish them into logically separated segmented called topics. This way any devices which are interested in some data can subscribe to a topic and receive updates. A basic model with two clients and a broker is shown in the figure 2 below. In this model client A publishes data for living room temperature and client B subscribed for living room temperature topic. Topics are represented by strings separated by slashes e.g living room/temperature

pub LV/temp pub

Client B

Broker

oker

Client B

Sub LV/temp

Figure 2

### Security

Security is not an inherent feature of MQTT however since version 3 we can pass usernames and passwords with MQTT packets. Security can be handled with SSL independently of the protocol. I have found some examples where mqtt is operating with bouncy castle API. Please refer to [8], [9] and [10] for further details.

### Pros and Cons

### Pros

* It is extremely light weight.
* It has a robust community, tools and several mature open source implementations.
* Works well over unstable networks.
* Has very good third party support.
* Seems very simple to use and get started with.

**Cons**

* Broker can become a single point of failure.
* Broker also means that we sacrifice some decentralization.
* Security is not built in to the protocol from ground up
* It is achieved using third party SSL APIs which add significant overhead.
* Could not find any example of it being used with any block chain technology.

## Telehash

Telehash is secure mesh networking technology with these design principles according to their official documentation:

* Full end-end encryption all the time.
* Strict privacy: no content, identity or metadata is revealed to third parties
* Is suitable for embedded, mobile and web usage
* Flexible transport protocols for compatibility with existing devices
* Official native implementations for variety of platforms/languages including javascript, c#, objective-c, c, go

### Basic Architecture and High level properties

The basic architecture contains device instances which contains a logical mesh which is just a collection of secure links to other instances. A link between two instances is private only to them and not shared across the mesh. The current version has the following high level properties.

* Each end point has a unique verifiable finger print in the form of a hash name.
* This hash name acts as a secure universal mac.
* Provide native tunneling capabilities e.g. for HTTP, object streams and TCP/UDP sockets.
* Supports asynchronous and synchronous communication.
* Supports automatic peer discovery.
* Specifies a URI format for facilitating links out of band.
* Supports bridging and routing privately by default and optionally via public DHT.
* Supports JSON object signing and encryption **(JOSE)** and OpenID connect.

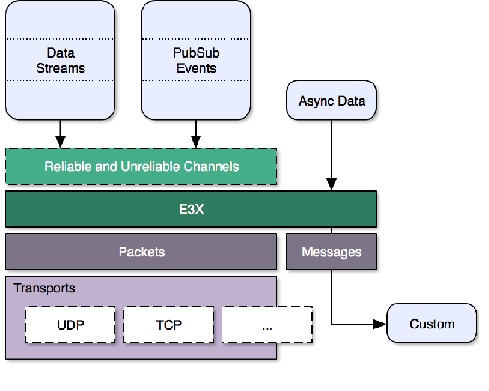


Figure 2 (figure taken from [11])

### Pros and Cons

**Pros**

* Works over reliable and unreliable networks.
* Is architecture agnostic i.e. can use centralized, pub/sub, P2P, REST etc.
* Built from ground up with strong security in mind.
* Works well with variety of platforms including embedded devices.
* Standalone Chunking mechanism of telehash is compatible with several RF transport protocols such as CoAP, zigbee, atmel mesh etc.
* High-level guide for encrypting connections to bitcoin networks available [12].
* Several block chain based projects have selected telehash for their use case scenarios.
* Can be configured to work in a completely decentralized or peer to peer manner.
* Previous versions of telehash were based on modified version of bouncy castle api called

**Cons**

* Compared to MQTT telehash does not have such a robust community.
* Most online help I found is based on the official GitHub repository and documentation.
* Telehash seems more complex and has a steeper learning curve.
* Was not able to find any example of latest Telehash being used with bouncy castle, however java implementation of an earlier version uses spongy castle which is a modified version of bouncy castle.

# Raiden Contracts and Modules

Raiden is a solution for off-chain payments or asset transfers by employing payment channels or state channels. In order to protect against double spending attacks off-chain transfers must be backed up by assets on the blockchain. The payment channel on block chain is represented by a smart contract which:

* Provides shared rules agreed up front by both parties.
* Holds the token value in escrow to back the off-chain payments.
* Resolves disputes using already agreed upon rules.

If a dispute arises smart contract can be used to settle and withdraw tokens.

## Netting Channel Smart contract

The Netting channel smart contract carries implicit shared rules agreed upon by the participants of the channel. It acts an on-chain payment channel between two raiden nodes. It allows for the following.

* Arbitrary number of token/asset transfers between channel participants.
* Conditional value transfers that have an expiration and predefined rule to withdraw.
* Rules to determine order of transfers.

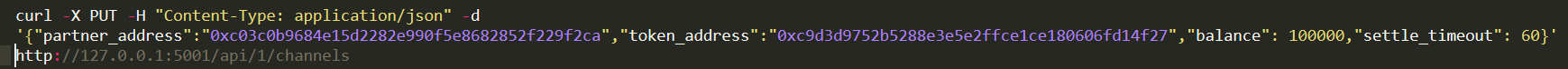
Each netting channel is associated with one bi-directional payment channel. Each channel deals with an ERC20 token and has its own settlement period measured in terms of blocks. Any of the two participants can deposit any number of times any amount of a specific ERC20 token associated with channels Netting contract. The token transfers are represented by hash lock structures that contain token amount, expiration and hash lock. The set of pending transfers are encoding into a Markel tree and represented in each transfer by its root. The total channel capacity is equal to total amount of tokens deposited by both participants. Any participant can increase the capacity at any time by depositing more tokens into the channel. The capacity is divided into available and locked balance to each participant/direction.

### Channel Life cycle

* **Deployment**
* **Funding**
* **Token Transfer**
* **Close**
* **Settle**

**Deployment:** A channel is deployed by calling the /channel endpoint of raiden rest API with the correct ERC20 token address and the partner with which we want to use the channel with.

**Example:**



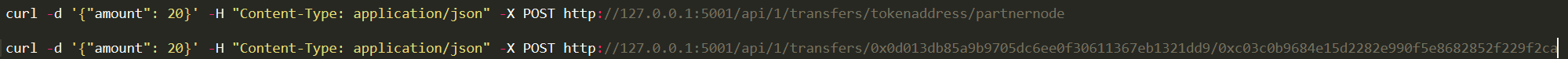
**Funding:** A channel can be funded by either or both parties by transferring and locking tokens in to the channel, once tokens have been transferred each participant can perform an arbitrary number of transfers back and forth provided they have the necessary funds to perform the transfer.

**Example Funding from counter party:** It uses the channel address obtained as output when the channel is first deployed.



**Token Transfer:** Once the channel has been deployed and funded either party can perform multiple transfers back and forth. The token transfer uses the channel address endpoint.

**Example Token transfer**



**Close**

Once either party wants to withdraw tokens or a dispute arises the channel must be closed. This is done by calling the close function. After the close function is called the settlement window opens. Within the settlement window both parties must update the counterparty state and withdraw the unlocked locks.

**Settle**

Any participant can close the channel at any point and from that point no more transfers can be made. Once the channel enters the settlement window the partner state can be updated using the updateTransfer function. After partner state is updated by participant’s locks may be withdrawn. A **withdraw** checks the locks and updates the participants current transferred amount more details are given on [13]. Once withdraw is called or settlement timeout expires the final tokens locked in the channel are distributed to the partners according

## Advanced Topics related to Raiden

I am including some advanced topics here for my own reminder I will put them in detail in the final thesis report for now they are out of the scope of this document.

**UpdateTransfer()**

This is an advanced topic and I will write it up in more detail in my final thesis report details can be found in [13].

**Balance Proof**

More on [13]

Channel Manager Contract:

### Raiden Transfers

**Types**

* Direct Transfers
* Mediated Transfer
* Refund Transfers

Details about these and more including mechanisms of transfer can be found on [13]. All of these should go in the final report.

### Network protocol [13]

### Markel Trees [13]

Might not need to add full merkel tree section if already discussed in an earlier topic in thesis report.

# Raiden API Documentation

Raiden has Restful API with url endpoints corresponding to user facing interactions allowed by raiden node. The endpoints accept and return JSON encoded objects. The API path has this format /api/<version>/. All API calls contain api version in order to differentiate calls between different api versions.

## Endpoints

* **Deploying / Registering Token**

**Format:** /api/<version>/tokens/<token\_address>

**Example:** PUT /api/1/tokens/0xea674fdde714fd979de3edf0f56aa9716b898ec8

* **Querying Information About Channels and Tokens**

**Format:** /api/<version>/channels/<channel\_address>

**Example:** GET /api/1/channels/0x2a65aca4d5fc5b5c859090a6c34d164135398226

* **Querying all channels**

**Example**: GET /api/1/channels

## Further Details:

Further details about these and other raiden endpoints can be found on [14]. In the final report give all API endpoints with more details including outputs.

# References

[1] <https://filament.com/assets/downloads/Filament%20Security.pdf>

[2] <https://filament.com/assets/downloads/Filament%20Foundations.pdf>

[3] <http://www.libelium.com/products/waspmote/overview/>

[4] <https://www.advanticsys.com/shop/mtmcm5000msp-p-14.html>

[5] <https://telosbsensors.wordpress.com/>

[6] <https://tutorials-raspberrypi.com/raspberry-pi-sensors-overview-50-important-components/#wireless>

[7] <https://www.linux.com/news/21-open-source-projects-IoT>

[8] <https://github.com/Hill30/mqtt-client/wiki/How-to-make-MQTT-connection-work-over-SSL>

[9] <https://gist.github.com/sharonbn/4104301>

[10] <https://blog.doubleslash.de/25928-2/>

[11] <https://github.com/telehash/telehash.github.io/blob/master/v3/spec/v3.0.0-stable.pdf>

[12] <https://github.com/telehash/telehash.github.io/blob/master/v3/guides/bitcoin.md>

[13] <http://raiden-network.readthedocs.io/en/stable/spec.html>

[14] <http://raiden-network.readthedocs.io/en/stable/rest_api.html>