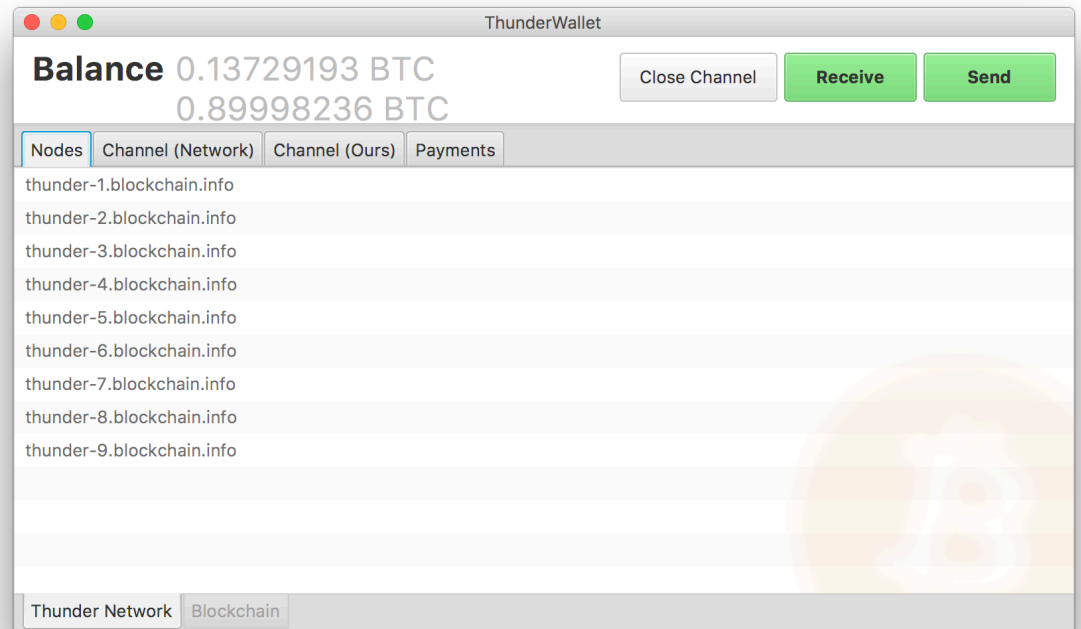

thundernetwork



build unknown

Wallet / Node implementation of the lightning.network P2P protocol. The lightning.network enables Off-Chain Bitcoin Payment Channels using smart contracts.

This is software in alpha status, don't even think about using it in production with real bitcoin. Current release is meant for testing, review, and building real world experience with the general technology.

To learn more, visit www.blockchain.com/thunder

Feature List

- ☒ Encryption
- ☒ Authentication
- ☒ Channel Opening Process
- ☒ Payment Debate
- ☒ Relaying Payment
- ☒ Settling Payment
- ☒ Peer Seeding
- ☒ Providing new Peers with Topology
- ☒ Basic Gossip Protocol

-
- ☒ Interface for Connecting into Wallet Software
 - ☒ Basic Blockchain Watching Capability
 - ☒ Closing a Channel
 - ☐ Hardening against various DDoS attacks
 - ☐ Backing Database Implementation (currently only in memory)
 - ☐ Restoring state after restart - cheking old TX for cheating
 - ☐ Claiming funds after counterparty cheated

Building

Prerequisites

You need

- ```
1 JDK 1.8+
2 Maven
```

to build both the node and the wallet software.

### Installation

Install Java 8 JDK (Unix machines: and OpenJFX) before proceeding.

Executing

- ```
1 ./build.sh
```

will run the tests and create the executables.

Running

Node software and Wallet software are completely independent projects. Nodes are generally passive participants of the network that only relay payments, whereas the Wallet software is only used to make and receive payments.

Node Running

- ```
1 java -jar thunder-node.jar
```

---

will start up an autonomous node that will connect to the thunder.network and build channels with other random nodes. On first startup it will ask the user for some basic configuration. Afterwards it will write this configuration to disk for the next start. To be an active part of the network, please configure your firewall to allow incoming connections on port 2204. You can change the default port in the config file.

### **Wallet** Running

```
1 java -jar thunder-wallet.jar
```

will start up the wallet. It will ask for known nodes and get a topology of the network. The user can then chose a node to form a channel with and make and receive payments.

### **Architecture**

thunder.network uses netty as the underlying networking library. There are several layers for encryption, establishing a channel and making payments.

Additional features will generally live inside their own layer, decoupled from the other layers.

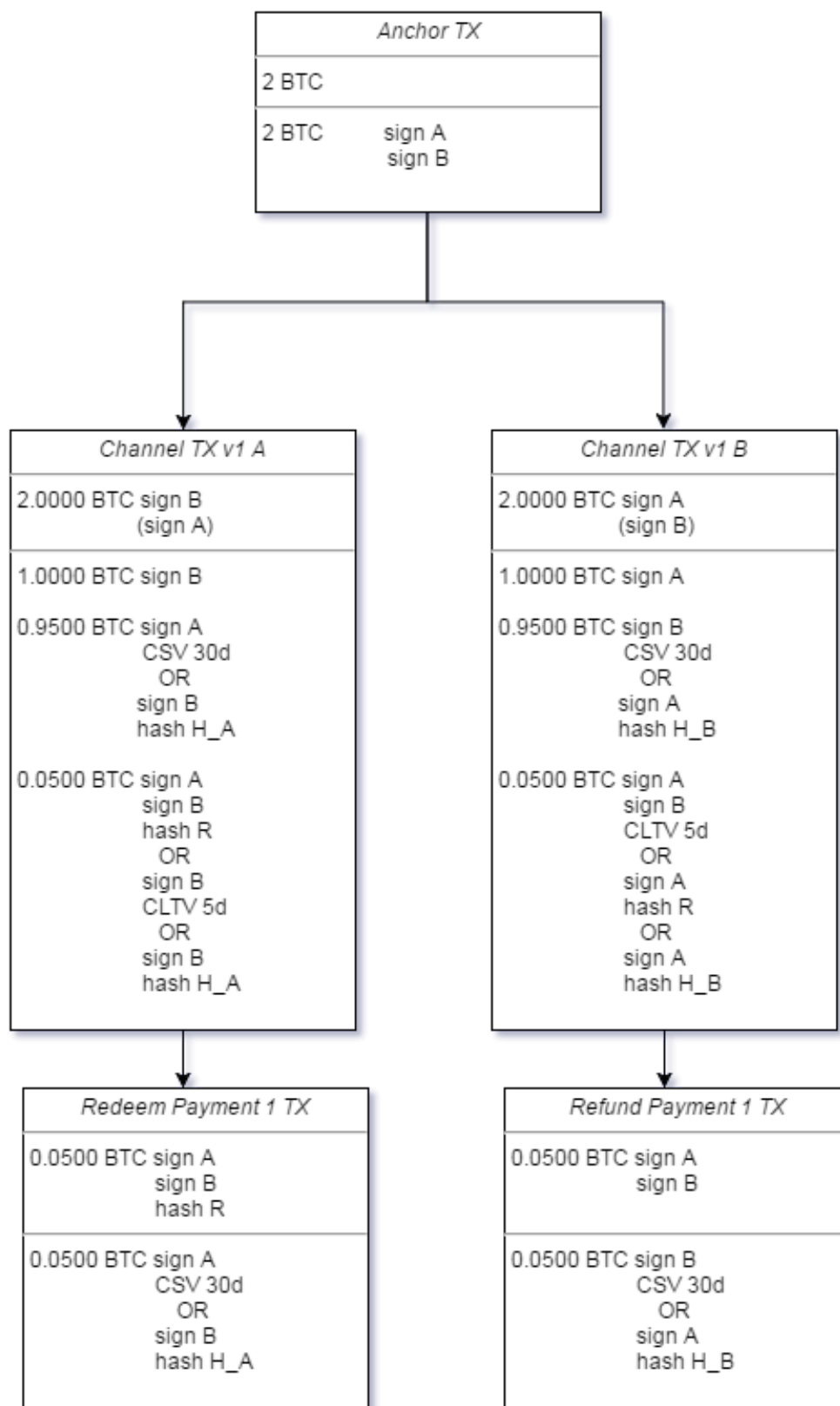
### **Outlook**

thunder.network uses a commitment-transaction design that needs both CSV and Segregated Witness to be completed. Otherwise the payments are not enforceable on the blockchain and are bad promises at best.

### **Dual-TX Approach**

thunder.network implements a commit-transaction design where each payment pays to a 2-of-2 multisig + R || TIMEOUT first. Both parties then create an additional revocable transaction paying to the correct receiver. While this adds additional complexity and makes on-chain resolution more expensive, it allows for decoupling the revocation delay from the refund timespan, which otherwise would not be possible.

In the diagram below, A is receiving a payment from B. If he wants to accept the payment, he has to produce R within 5 days, otherwise the payment is not enforceable. Regardless of that, B can always claim even that payment for another 30d if A cheated and broadcasted a revoked transaction.



---

## Routing

Currently we are using a completely source-oriented onion-encrypted routing algorithm. This means that the receiver gives his public key and the sender creates the complete route to the receiver (*source-oriented*). To make this reliable, each `ChannelPubkeyObject` and other topology data has to be broadcast and gossiped through the network. This can be problematic in case of bad network conditions, where the `ChannelPubkeyObject` of a new node might now propagate to the sender. In this case the sender is not able to find a route and therefore the payment fails.

It is therefore an open TODO to allow for Rendezvous-Point routing (*RP*). In *RP*-Routing the sender will send the public key of a nearby node and the encrypted route from this node to himself. This will make addresses much longer (as they now contain an encrypted route), but will greatly improve privacy (the sender does not know the final receiver) and reliability of path finding (nodes have a much higher probability to be in the senders topology than the receiver's wallet).

Furthermore all routes are onion-encrypted, meaning that for each node in the route we will add one layer encrypting all subsequent layers. When decrypting, each node is only able to read the next node it should pass the payment to. Doing so makes disincentivizing wrong behaviour more difficult later on, but makes active analysis of payments less feasible.

## Anchor

Payment channels are using a 2-of-2 multisig transaction as an anchor. They do so by handing back and forth an unsigned transaction, such that they both can add in- and outputs to fund the channel. They then create the first channel transaction without any included payments that also serves as a refund. Only after exchanging and checking the signatures for the channel transaction they exchange the signatures for the actual anchor and broadcast it to the network.

For this to work out, Segregated Witness is mandatory, as it is impossible to create the channel transaction without knowing the signatures of the parent transaction with plain transactions. Furthermore the risk of malleability means that one party can hold the other parties funds hostage.

## Optimizations

As this is still a prototype, various optimizations are left open for now, as they would hinder active development. For example, JSON was chosen to serialize messages, as Gson allows for very prototype-friendly development, even though it increases message size 3-5 fold.

---

## Contributing

There is always something to do! Feel free to look through the open Issues or look for TODOS in the code. They are well maintained and usually still open. If you want to create a PR, it would be great if you could respect the code style convention of this project. It's easiest if you just use IntelliJ and import the code style settings and follow this tutorial to setup automatic reformatting on save.

## Resources

- [1] LN-Implementation by Rusty Russel, <https://github.com/ElementsProject/lightning>