Bitcoin Regtest and Lightning Network Implementation

Prepared by **Prabal Das** under guidance of **Dr. Anisur R. Molla**

December 16, 2024

Contents

0.1 Overview

This project demonstrates the setup and usage of a Bitcoin **regtest** network integrated with the **Lightning Network Daemon (LND)**. Key features include:

- Channel creation and management.
- Multi-hop payments.
- Atomic Multi-Path Payments (AMP).
- Channel closure operations.

Project Explanation: There will be **four** lightning nodes, say, the nodes are A, B, C and E and we need to create channel between them. There will be channel between A to B, A to E, B to C and E to C and we need to generate a payment request at C and the payment will be Multihop payment, Multipath payment and Atomic multipath payment.

0.2 Environment Setup

0.2.1 Requirements

Before starting we need to ensure that the following software is installed:

- Bitcoin Core -version 24 or higher(I have implemented in version v27.0.0).
- Lightning Network Daemon (LND) with Go environment.
- Ubuntu 22.04.4 LTS or above (or any compatible Linux OS).

0.2.2 Installation

1. Install Bitcoin Core: `https://bitcoin.org/en/download`

```
#Download the latest release using wget from the terminal
$ wget https://bitcoincore.org/bin/bitcoin-core-27.0/bitcoin
   -27.0-x86_64-linux-gnu.tar.gz
#Extract the downloaded file:
$ tar -xvzf bitcoin-27.0-x86_64-linux-gnu.tar.gz
#Move the binaries to a directory in your PATH:
$ sudo mv bitcoin-27.0/bin/* /usr/local/bin/
#Verify installation:
$ bitcoind --version
```

Listing 1: Bitcoin Core Download

```
    Bitcoin Core version v27.0.0
```

Listing 2: Output Bitcoin Core Download

2. Install Go: `https://go.dev/dl/`

```
#lnd is written in Go, with a minimum version of 1.23.4, thus
    install GO first.
$ wget https://dl.google.com/go/go1.23.4.linux-amd64.tar.gz
#Install Go:
$ sudo rm -rf /usr/local/go && sudo tar -C /usr/local -xzf
   go1.23.4.linux-amd64.tar.gz

#This ensures that your shell will be able to detect the
   binaries you install.
$ export PATH=$PATH:/usr/local/go/bin
$ export GOPATH=~/go
$ export PATH=$PATH:$GOPATH/bin
```

Listing 3: Installing Go Environment

3. Install LND: `https://github.com/lightningnetwork/lnd`

```
#Installing lnd from source
$ git clone https://github.com/lightningnetwork/lnd
$ cd lnd
$ make install
```

Listing 4: Installing lnd

0.3 Regtest Network Setup

0.3.1 Configure bitcoin.conf

Once Bitcoin Core is installed, configure it to use regtest and initialize the Bitcoin regtest environment. Note that, /**.bitcoin** directory is a hidden folder, run `ls -a` to see it.

```
#Create or edit bitcoin.conf:
$ nano ~/.bitcoin/bitcoin.conf
#Add these lines to .bitcoin.conf:
regtest=1        # enable regtest
server=1 # enable bitcoin core server for bitcoin-cli command
fallbackfee=0.0002      # default txn fee if not set
rpcuser=prabaldas       # Remote Procedure Call
rpcpassword=password@567
```

```
zmqpubrawblock=tcp://127.0.0.1:28332    #Publishes raw block data
    through ZeroMQ on the specified address and port.
zmqpubrawtx=tcp://127.0.0.1:28333       # ZeroMQ is a messaging
    protocol used for real-time data updates
```

Listing 5: Edit Bitcoin conf file

0.3.2 Launching Bitcoin Regtest

```
#Start Bitcoin Core in regtest mode:
$ bitcoind -regtest -daemon
#To monitor the synchronization progress, check the logs:
$ tail -f ~/.bitcoin/regtest/debug.log
```

Listing 6: Run Bitcoin core

This log file will be look like(below for mine case!)

```
2024-12-16T12:26:00Z Flushed fee estimates to fee_estimates.dat.
2024-12-16T12:32:46Z Potential stale tip detected, will try using
    extra outbound peer (last tip update: 12183 seconds ago)
2024-12-16T12:43:16Z Potential stale tip detected, will try using
    extra outbound peer (last tip update: 12813 seconds ago)
2024-12-16T12:53:46Z Potential stale tip detected, will try using
    extra outbound peer (last tip update: 13443 seconds ago)
2024-12-16T13:04:16Z Potential stale tip detected, will try using
    extra outbound peer (last tip update: 14073 seconds ago)
2024-12-16T13:14:46Z Potential stale tip detected, will try using
    extra outbound peer (last tip update: 14703 seconds ago)
2024-12-16T13:25:16Z Potential stale tip detected, will try using
    extra outbound peer (last tip update: 15333 seconds ago)
2024-12-16T13:26:00Z Flushed fee estimates to fee_estimates.dat.
...
```

Listing 7: Demo log output

0.3.3 Creating Genesis block and Chain

```
#Check that it's running properly:
$ bitcoin-cli -regtest getblockchaininfo
```

Listing 8: Get Block Info

Output should be look like:

```
{
  "chain": "regtest",           # the node is operating in regtest
      mode.
  "blocks": 0,                      # the current number of blocks in
     the blockchain
  "headers": 0,                     # the number of block headers the
     node has received.
  "bestblockhash":
"0f9188f13cb7b2c71f2a335e3a4fc328bf5beb436012afca590b1a11466e2206"
    ,     # hash of the best block (the most recent block) on the
    blockchain. In regtest, since the chain is likely empty or just
     starting, it will show the hash of the genesis (first) block.
  "difficulty": 4.656542373906925e-10,  # the difficulty of mining
      a new block(which is very low, close to zero)
  "time": 1296688602,    # the current time in Unix timestamp
     format, representing the number of seconds since January 1,
     1970 (the Unix epoch).
  "mediantime": 1296688602,               # The median time past is
     the median of the block's timestamps over the past 11 blocks.
      In this case, it's the same as time because you're likely in
      a newly initialized regtest environment.
  "verificationprogress": 1,    # the progress of blockchain
     verification, ranging from 0 (not verified) to 1 (fully
     verified).
  "initialblockdownload": true,
  "chainwork":
"0000000000000000000000000000000000000000000000000000000000000002"
    ,     #represents the total accumulated work of the blockchain,
     measured as the number of computational steps taken to build
    the chain.
  "size_on_disk": 293,  # size of the blockchain data on disk, in
     bytes.
  "pruned": false,       # Pruning refers to discarding old block
     data to save disk space. If pruned is true, the node discards
      old blocks that are not needed for validation.
  "warnings": ""
}
```

Listing 9: output of Block Info

```
#Create a wallet:(It creates a new wallet named "Prabal" in your
    Bitcoin Core node)
$ bitcoin-cli -named createwallet wallet_name="Prabal"
```

Listing 10: Wallet Create

```
{
  "name": "Prabal"
}
```

Listing 11: output

Now, we need to generate the wallet address.

```
#Create a Bitcoin Address:
$ bitcoin-cli -regtest getnewaddress
```

Listing 12: Wallet Address

Note down the address for future use.

```
bcrt1qqpa2n2srqan7als93yu8hka58cff5pc3f42847
```

Listing 13: Demo Wallet Address

Now, we need bitcoin and for that we will mine blocks (this generates 6 blocks and sent to the specified address, but you need atleast 101 many blocks to generate).

```
$ bitcoin-cli -regtest generatetoaddress 101
    bcrt1qqpa2n2srqan7als93yu8hka58cff5pc3f42847
```

Listing 14: Block generation

```
[
"41f66d31c7e8e9fa53c53411e0f40034a8ca726c0e44e9e5c75b2f79ea3d8d85"
    ,
"544029b97516aa2905e69108e5736e1ba87c3474215957b083d8fd4d870610ac"
    ,
"47f85703dff01596b5ac5fff2b37c65dda97cd4cd3f0f2d28f754e9f0c451635"
    ,
"2868a8e26c7be6509ae4ae83a56251ada2a95995193c146d5419e141967b1cb0"
    ,
"78281df137ba375155381d310038fe91f5c26c749dba175146c3924728c24917"
    , "1
    c7a5b846aa77973709237a707b9ceef8a4cc340021041e2970e441bc28ff975
    "
]
```

Listing 15: Demo Output

You shoud have 50btc in your wallet.

```
#Check Your Balance:
$ bitcoin-cli -regtest getbalance
```

Listing 16: Check Balance

For wallet load

```
$ bitcoin-cli -regtest loadwallet "Prabal"
```

Listing 17: Check Balance

0.3.4 Setting Up LND Nodes

To create LND nodes and configure those:

```
#Create Separate Data Directories
$ mkdir -p ~/node-A ~/node-B ~/node-C ~/node-E
#Create a configuration file for each node (e.g., lnd.conf) in
    their respective directories:
$ touch ~/node-A/lnd.conf ~/node-B/lnd.conf ~/node-C/lnd.conf ~/
    node-E/lnd.conf
```

Listing 18: Node Creation

```
#Here's an example configuration for node-A:
[Application Options]
alias=A
listen=127.0.0.1:9735
restlisten=127.0.0.1:8081
debuglevel=info
rpclisten=127.0.0.1:10009

[Bitcoin]
bitcoin.active=1
bitcoin.regtest=1
bitcoin.node=bitcoind

[Bitcoind]
bitcoind.rpcuser=prabaldas
bitcoind.rpcpass=password@567
bitcoind.zmqpubrawblock=tcp://127.0.0.1:28332
bitcoind.zmqpubrawtx=tcp://127.0.0.1:28333
bitcoind.rpchost=127.0.0.1
```

Listing 19: Node Configuration

Repeat for other nodes (Node B, Node C, etc.). Remember, only alias, listen port, restlisten port and rpclisten should be different(in my case, those are one incremented for the nodes i.e. for listen port of B, it is 36, for C it is 37 and for E it is 38 and so on).

0.3.5 Run lnd instances and Wallet setup

Now we will run all four lnd instances parallaly in four terminals and there will be one main terminal where bitcore core will be running.

```
#Run each lnd instance with its corresponding configuration and
    data directory.
$ lnd --lnddir=~/node-A --configfile=~/node-A/lnd.conf
$ lnd --lnddir=~/node-B --configfile=~/node-B/lnd.conf
$ lnd --lnddir=~/node-C --configfile=~/node-C/lnd.conf
$ lnd --lnddir=~/node-E --configfile=~/node-E/lnd.conf
```

Listing 20: Run Node Configuration

The above will be run in four terminals. Now we will create wallet for each nodes. The following will be run in our main terminal one by one(Note that now onwards we will run all command in our main terminal)

```
#Initialize Wallets
$ lncli --lnddir=~/node-A --rpcserver=localhost:10009 create
$ lncli --lnddir=~/node-B --rpcserver=localhost:10010 create
$ lncli --lnddir=~/node-C --rpcserver=localhost:10011 create
$ lncli --lnddir=~/node-E --rpcserver=localhost:10012 create
```

Listing 21: Wallet Creation

After running the above command, it will ask for a wallet password. Note down this for future wallet unlocking. After wallet creation, it will give a fresh seed, which will be helpful for password recovery.

```
Generating fresh cipher seed...

!!!YOU MUST WRITE DOWN THIS SEED TO BE ABLE TO RESTORE THE WALLET
   !!!

---------------BEGIN LND CIPHER SEED---------------
 1. absent    2. chalk      3. team      4. father
 5. twelve    6. dove       7. today     8. carry
 9. under    10. hungry    11. pipe     12. success
13. gather   14. fold      15. dry      16. sun
17. school   18. fiber     19. soda     20. evoke
21. wall     22. monster   23. cradle   24. lawsuit
---------------END LND CIPHER SEED-----------------
```

Listing 22: Demo Output

The following command is for unlocking wallet.

```
# Wallet unlock
$ lncli --lnddir=~/node-A --rpcserver=localhost:10009 unlock
$ lncli --lnddir=~/node-B --rpcserver=localhost:10010 unlock
$ lncli --lnddir=~/node-C --rpcserver=localhost:10011 unlock
$ lncli --lnddir=~/node-E --rpcserver=localhost:10012 unlock
```

Listing 23: Wallet Creation

The following command is for getting the public key of each nodes.

```
# Get the Public Key of a Lightning Node
```

```
$ lncli --lnddir=~/node-A --rpcserver=localhost:10009 --network=
    regtest getinfo
$ lncli --lnddir=~/node-B --rpcserver=localhost:10010 --network=
    regtest getinfo
$ lncli --lnddir=~/node-C --rpcserver=localhost:10011 --network=
    regtest getinfo
$ lncli --lnddir=~/node-E --rpcserver=localhost:10012 --network=
    regtest getinfo
```

Listing 24: PubKey generation

```
{
    "version": "0.18.99-beta␣commit=fn/v2.0.5-2-gbb9c680a4",
    "commit_hash": "bb9c680a48cd1075d793cfc97b85f3676b2812c2",
"identity_pubkey":"03f05cfe8f4d48074c766574e7e8246f7
a2eaf06e74f2c21447d5fc1d85829608d",
    "alias": "A",
    "color": "#3399ff",
    "num_pending_channels": 0,
    "num_active_channels": 0,
    "num_inactive_channels": 0,
    "num_peers": 0,
    "block_height": 202,
    "block_hash":
"5fcc83ef1d287546f977b1549ea5db9dc49a8d53e4022683ff74e85bd0a9200c"
    ,
    "best_header_timestamp": "1734204010",
    "synced_to_chain": true,
    "synced_to_graph": false,
    "testnet": false,
    "chains": [
        {
            "chain": "bitcoin",
            "network": "regtest"
        }
    ],
...
```

Listing 25: Demo Output

Now, we will generate the wallet addresses for each nodes.

```
$ lncli --lnddir=~/node-A --network=regtest  --rpcserver=localhost
    :10009 newaddress p2wkh
$ lncli --lnddir=~/node-B --network=regtest  --rpcserver=localhost
    :10010 newaddress p2wkh
$ lncli --lnddir=~/node-C --network=regtest  --rpcserver=localhost
    :10011 newaddress p2wkh
$ lncli --lnddir=~/node-E --network=regtest  --rpcserver=localhost
    :10012 newaddress p2wkh
```

Listing 26: generating Wallet Address

Note down the `Identity_pubkey` and `address` for future references. In my case, the corresponding pubkeys and wallet addresses are:

```
A:  "identity_pubkey":  "02a2
3391ddf8f2f53ec5f7332a46ae6b17dda60676d1637eb6984c71bfe0a72e93"
"address":  "bcrt1q64jk96xz5p4txllw56mwmcxg9543edhjthp5ve"
B:"identity_pubkey":  "03ff
a538d84fecd3bcd6fa7ac9d8067f477eded1b39d4fc1a3503b718033ec5efb"
"address":  "bcrt1qtwz6tjqdafeqehane7l4chs0g2lc066wtme7fj"
C:"identity_pubkey":  "0326
    b7bc8d7c5d87356e86ce45c2586127715e7f215b0dcb0cdfc4b05763a4ee10"
"address":  "bcrt1qlheu9devx6q5pknxmt98hmczpyzrzgeuqgsv72"
E: "identity_pubkey":  "0206
    d7e629d517983592184f9633685631ee699f33b3eab15e69839074e6edbabe"
"address":  "bcrt1q32vzvxwgwkac6djhf3h39y2vs8c7qfnpurmy0g"
```

Listing 27: demo pubkey and wallet address

Now, we will send 10 btc each of the nodes from our core wallet.

```
$ bitcoin-cli -regtest sendtoaddress
    bcrt1q64jk96xz5p4txllw56mwmcxg9543edhjthp5ve 10
$ bitcoin-cli -regtest sendtoaddress
    bcrt1qtwz6tjqdafeqehane7l4chs0g2lc066wtme7fj 10
$ bitcoin-cli -regtest sendtoaddress
    bcrt1qlheu9devx6q5pknxmt98hmczpyzrzgeuqgsv72 10
$ bitcoin-cli -regtest sendtoaddress
    bcrt1q32vzvxwgwkac6djhf3h39y2vs8c7qfnpurmy0g 10
```

Listing 28: Sending btc

It will give some txn id,note down those ids.

```
471129d78ff4c233815d2bed8ba59d83aab51851d26da79453ecf3559d4c78d3
```

Listing 29: Demo txn id

Since, we have sent 10 btc to each of nodes, it is not confirmed yet. For that we need to mine 6 more blocks.

```
$ bitcoin-cli -regtest generatetoaddress 6
    bcrt1qqpa2n2srqan7als93yu8hka58cff5pc3f42847
```

Listing 30: Block Mining

Now we can see each nodes wallet balance. For that we need to run the following command one by one in our main terminal:

```
$ lncli --lnddir=~/node-A --rpcserver=localhost:10009 --network=
    regtest walletbalance
$ lncli --lnddir=~/node-B --rpcserver=localhost:10010 --network=
    regtest walletbalance
$ lncli --lnddir=~/node-C --rpcserver=localhost:10011 --network=
    regtest walletbalance
$ lncli --lnddir=~/node-E --rpcserver=localhost:10012 --network=
    regtest walletbalance
```

Listing 31: Wallet Balance check

```
{
    "total_balance":  "1000000000",
    "confirmed_balance":  "1000000000",
    "unconfirmed_balance":  "0",
    "locked_balance":  "0",
    "reserved_balance_anchor_chan":  "0",
    "account_balance":  {
        "default":  {
            "confirmed_balance":  "1000000000",
            "unconfirmed_balance":  "0"
        }
    }
}
```

Listing 32: Demo Output for node-A

So, finally we have created nodes and corresponding wallets with some btc. Now we will create the channels.

0.4 Channel Management

0.4.1 Connecting Nodes

First we will connect the nodes before creating the channels. We need to connect A-B, A-E, E-C and B-C.

```
#Connect A to B:
$ lncli --lnddir=~/node-A --network=regtest -rpcserver=localhost
    :10009 connect 03ffa538d84fecd3bcd6fa7ac9d8067
f477eded1b39d4fc1a3503b718033ec5efb@127.0.0.1:9736
#Connect B to C:
$ lncli --lnddir=~/node-B --network=regtest -rpcserver=localhost
    :10010 connect 0326b7bc8d7c5d87356e86ce45c258
6127715e7f215b0dcb0cdfc4b05763a4ee10@127.0.0.1:9737
#Connect A to E:
```

```
$ lncli --lnddir=~/node-A --network=regtest -rpcserver=localhost
    :10009 connect 0206d7e629d517983592184f963368
5631ee699f33b3eab15e69839074e6edbabe@127.0.0.1:9738
#Connect E to C:
$ lncli --lnddir=~/node-E --network=regtest -rpcserver=localhost
    :10012 connect 0326b7bc8d7c5d87356e86ce45c2586
127715e7f215b0dcb0cdfc4b05763a4ee10@127.0.0.1:9737
```

Listing 33: Connecting peers

```
{
    "status": "connection␣to␣03
        ffa538d84fecd3bcd6fa7ac9d8067f477eded1b39d4
fc1a3503b718033ec5efb@127.0.0.1:9736␣initiated"
}
```

Listing 34: Demo Output

we can also verify the peers of a node by running the following command.

```
#Check Connections:
$ lncli --lnddir=~/node-A --network=regtest -rpcserver=localhost
    :10009 listpeers
$ lncli --lnddir=~/node-B --network=regtest -rpcserver=localhost
    :10010 listpeers
$ lncli --lnddir=~/node-C --network=regtest -rpcserver=localhost
    :10011 listpeers
$ lncli --lnddir=~/node-E --network=regtest -rpcserver=localhost
    :10012 listpeers
```

Listing 35: verifying peers

```
{
    "peers": [
        {
            "pub_key": "03ffa538d84fecd3bcd6fa7ac9d8067f477e
ded1b39d4fc1a3503b718033ec5efb",
            "address": "127.0.0.1:9736",
            "bytes_sent": "394",
            "bytes_recv": "394",
            "sat_sent": "0",
            "sat_recv": "0",
            "inbound": false,
            "ping_time": "-1",
            ...
        }
        {
            "pub_key": "0206d7e629d517983592184f9633685631ee
699f33b3eab15e69839074e6edbabe",
            "address": "127.0.0.1:9738",
            "bytes_sent": "349",
            "bytes_recv": "348",
            "sat_sent": "0",
            "sat_recv": "0",
            "inbound": false,
            "ping_time": "-1",
            ...
        }
    ]
}
```

Listing 36: Demo Output

0.4.2 Opening a Channel

Open a channel between nodes:

```
$ lncli --lnddir=~/node-A --network=regtest -rpcserver=localhost:10009 openchannel --node_key=03ffa538d84fecd3bcd6fa7ac9d8067f477
eded1b39d4fc1a3503b718033ec5efb --local_amt=10000000
$ lncli --lnddir=~/node-B --network=regtest -rpcserver=localhost:10010 openchannel --node_key=0326b7bc8d7c5d87356e86ce45c25861277
15e7f215b0dcb0cdfc4b05763a4ee10 --local_amt=10000000
$ lncli --lnddir=~/node-A --network=regtest -rpcserver=localhost:10009 openchannel --node_key=0206d7e629d517983592184f9633685631ee
699f33b3eab15e69839074e6edbabe --local_amt=10000000
$ lncli --lnddir=~/node-E --network=regtest -rpcserver=localhost:10012 openchannel --node_key=0326b7bc8d7c5d87356e86ce45c25861277
15e7f215b0dcb0cdfc4b05763a4ee10 --local_amt=10000000
```

Listing 37: Opening Channels

Note that, after channel creation, we will get `funding_txid`, which will be needed when we will close the channels.

```
A->B:"funding_txid": "98a0682f3ba22707faaadf8802d56ec79d57dc8ce116f8863160ac976772628c"
B->C:"funding_txid": "e2
e9e521f4f5d3e7ed16b4ce6e0fe062d8a0626bb35d30928e0bf5a2d3d86dcd"
A->E:"funding_txid": "ef
c2a4a4391aeb18c2ab83b09e2c9f84d895030cd70c111e33b829a791a16e27"
E->C:"funding_txid": "3f
9a75d9fb9ab0aaa309c514387aa1df8188bb8424ec1b55692f63da54d98af7"
```

Listing 38: Demo Funding txn ids

The above is the funding txn ids in my implementation. Now, we can verify the channels by the following commands;

```
#Check Channels:
$ lncli --lnddir=~/node-A --network=regtest -rpcserver=localhost
    :10009 listchannels
$ lncli --lnddir=~/node-B --network=regtest -rpcserver=localhost
    :10010 listchannels
$ lncli --lnddir=~/node-C --network=regtest -rpcserver=localhost
    :10011 listchannels
$ lncli --lnddir=~/node-E --network=regtest -rpcserver=localhost
    :10012 listchannels
```

Listing 39: verify channels

0.5 Payments

0.5.1 Multi-Hop and Multi-path Payments

Till now we have created channels, now we will start our payment. In lnd node payments, multi-hop and multi-path payments are same i.e. during payments, if a node do not have sufficient fund in a channel, it will automatically split the payment and find different paths to happen the payment. For that, first we need to generate payment invoice in node C. we have funded 10000000sat in each channels and we will generate a invoice of 15000000sat, so that the payment happens in multi-path.

```
$ lncli --lnddir=~/node-C --network=regtest -rpcserver=localhost
    :10011 addinvoice --memo="Test␣Payment" --amt=15000000
```

Listing 40: Invoice generation

This will give a `payment_request` id.

```
    "r_hash":  "fc122367ee595ae599e3ac8b9fb01f05d197f3e4
82a5c54d1b85916c9fd30739",
"payment_request":"lnbcrt150m1pn47pegpp5lsfzxelwt9dw
tx0r4j9elvqlqhge0ulys2ju2ngmskgke87nquusdq523jhxapq2pshjmt9de6
qcqzzsxqyz5vqsp5lz6rlpz0a4ekps99vw579925zevx24kzg6c53y0nh0leh7gl7
vsq9qxpqysgq5hy6mmrmu5vznxlje49qv2uyrdqhh7spr00eyl9yqqdky28fegur6z
6cwjat6uh28n5fgyqaruxezvj
sm2y49ctlmns727tesec60hgq9xn7tf",
    "add_index":  "1",
    "payment_addr":  "f8b43f844fed7360c0a563a9e2955416586556c24
6b14891f3bbff9bf91ff320"
```

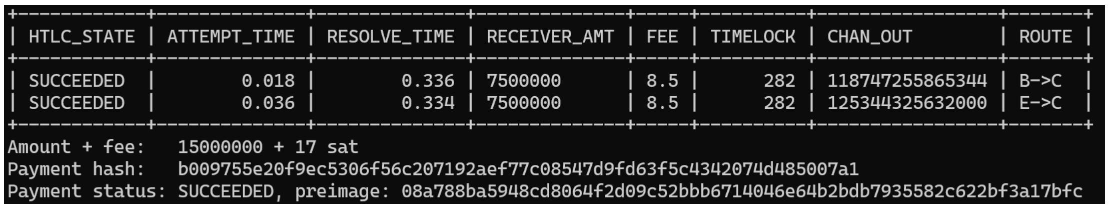
Listing 41: Invoice generation

Now, node A will pay the amount to node C using the invoice.

```
$ lncli --lnddir=~/node-A --network=regtest -rpcserver=localhost:10009 payinvoice
lnbcrt150m1pn47pegpp5lsfzxelwt9dw
tx0r4j9elvqlqhge0ulys2ju2ngmskgke87nquusdq523jhxapq2pshjmt9de6
qcqzzsxqyz5vqsp5lz6rlpz0a4ekps99vw579925zevx24kzg6c53y0nh0leh7gl7
vsq9qxpqysgq5hy6mmrmu5vznxlje49qv2uyrdqhh7spr00eyl9yqqdky28fegur6z
6cwjat6uh28n5fgyqaruxezvj
sm2y49ctlmns727tesec60hgq9xn7tf
```

Listing 42: Pay to C

So, we have initiated the payment and the nodes will find the paths to reach C. The output will be look like: In our case, 15000000sat is splitted into 7500000sat



```
+------------+--------------+--------------+--------------+-----+----------+-----------------+-------+
| HTLC_STATE | ATTEMPT_TIME | RESOLVE_TIME | RECEIVER_AMT | FEE | TIMELOCK | CHAN_OUT        | ROUTE |
+------------+--------------+--------------+--------------+-----+----------+-----------------+-------+
| SUCCEEDED  |        0.018 |        0.336 | 7500000      | 8.5 |      282 | 118747255865344 | B->C  |
| SUCCEEDED  |        0.036 |        0.334 | 7500000      | 8.5 |      282 | 125344325632000 | E->C  |
+------------+--------------+--------------+--------------+-----+----------+-----------------+-------+
Amount + fee:   15000000 + 17 sat
Payment hash:   b009755e20f9ec5306f56c207192aef77c08547d9fd63f5c4342074d485007a1
Payment status: SUCCEEDED, preimage: 08a788ba5948cd8064f2d09c52bbb6714046e64b2bdb7935582c622bf3a17bfc
```

Figure 1: Demo Output

in $A- > B- > C$ and 7500000sat in $A- > E- > C$ path.

0.5.2 Atomic Multi-Path Payments (AMP)

For AMP use flag –amp in the end while generating invoice and paying invoice. For, monitoring payment status and list of txn, run the following commands:

```
#Monitor Payment Status:
$ lncli --lnddir=~/node-C --network=regtest -rpcserver=localhost:10011 lookupinvoice <r_hash>
#List payments:
lncli --lnddir=~/node-A --network=regtest listpayments
```

Listing 43: Some Notes

0.6 Channel Closure

0.6.1 Cooperative Closure

To close a channel cooperatively, we need the corresponding funding txn ids.

```
$ lncli --lnddir=~/node-A --network=regtest -rpcserver=localhost
    :10009 closechannel --<funding_txid>
```

Listing 44: Cooperative Closure

After running this command for each channel, we will get closing txn ids. After closure, generate some more blocks to check whether the funds have been return to the Bitcoin regtest wallet and reflect the txn that you made off-chain.

0.7 Troubleshooting

0.7.1 Common Issues

- **Wallet Locked:** Ensure the wallet is unlocked before proceeding:

```
$ lncli --lnddir=~/node-A --rpcserver=localhost:10009 unlock
```

- **Insufficient Funds:** Generate additional blocks to fund the wallet:

```
$ bitcoin-cli -regtest generatetoaddress 6 <address>
```

0.8 References

- LND Documentation: Link
- Bitcoin Core Documentation: Link
- Learning-Bitcoin-from-the-Command-Line: Link
- Some materials from **G o o g l e**