Indian Institute of Technology Bombay

Dual Degree Stage 1 Report

Flare Routing in Lightning Network

Author:
Shruti Hiray

Guide:
Prof. Saravanan Vijayakumaran

A report submitted in fulfillment of the requirements for DDP Stage 1 Project

Department of Electrical Engineering
Indian Institute of Technology Bombay

October 2018

Acknowledgements

I would like to take this opportunity to express deepest gratitude towards my guide Prof. Saravanan Vijayakumaran for his motivating support, guidance, and valuable suggestions. I would also like to thank Vibha Ma'am of Bharti Communication Centre.

Abstract

The primary goal of this project is to study Flare routing in Lightning Network. Lightning Network is a layer built upon a blockchain (most commonly Bitcoin) to address scalability in blockchain. Lightning network enables fast and small value transactions. Flare routing protocol is used to route payments using multihop in the network. It is a hybrid algorithm that achieves reachability to any node in the network with high probability by storing minimum routing state.

Contents

Chapter 1

Blockchain

1.1 Introduction

Blockchain is a form of database. As inferred from the name, blockchain is a chain of blocks each of which stores records. Blocks are secured using principles of cryptography. Each block contains the cryptographic hash of the previous block. This results in linking of blocks forming the blockchain. Tampering with contents in any block requires to modify hash of all the subsequent blocks in the chain. This along with mining makes blockchain secure. The blockchain is replicated on all the nodes in the network. Thus, blockchain is a distributed database. Bitcoin is a cryptocurrency that uses blockchain as the underlying basis to store transactions.

1.2 Implementation

A **block** in Bitcoin consists of header and the transactions that are included in that block. Figure 1.1 [1] shows the contents of a block used in Bitcoin. A **block header** consists of various fields such as the timestamp, nonce, the cryptographic hash of previous block as shown in Figure 1.2 [1]. The cryptographic function used to calculate the hash is SHA-256.

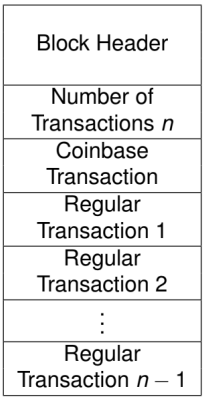

FIGURE 1.1: Block

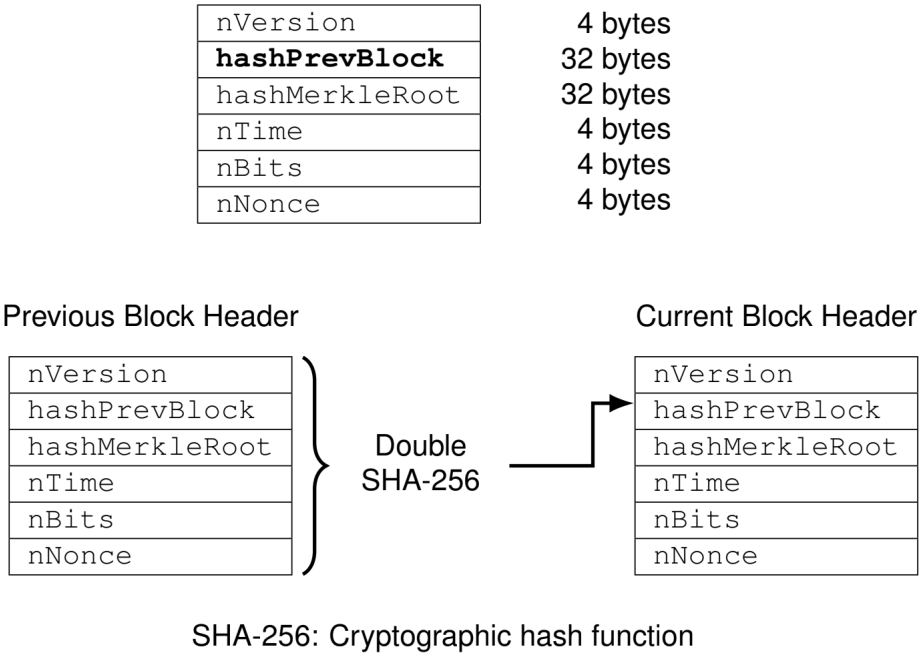

FIGURE 1.2: Block Header

Miners are the nodes that collect some of the valid broadcasted transactions in a block. The miners have to solve a Proof of Work puzzle to add the block to the blockchain. The Proof of Work puzzle is computationally hard but easily verifiable. The regular transactions have a small fee to miner's address for including the transaction in the block. The regular transactions includes sender, receiver addresses and the number of coins to each address. The miner who can solve the puzzle has his block included in the blockchain. If multiple miners solve a puzzle simultaneously, the miner who has his/her block in the chain which is eventually longer wins in long term period. The blocks in other chains join the longer chain eventually. Each block also contain a coinbase transactions in which 12.5 bitcoins (present value) are awarded to miner's address.

A block chain is as shown in the Figure 1.3 [1]. The various blocks are linked by the `hashPrevBlock` field as shown in Figure 1.2.

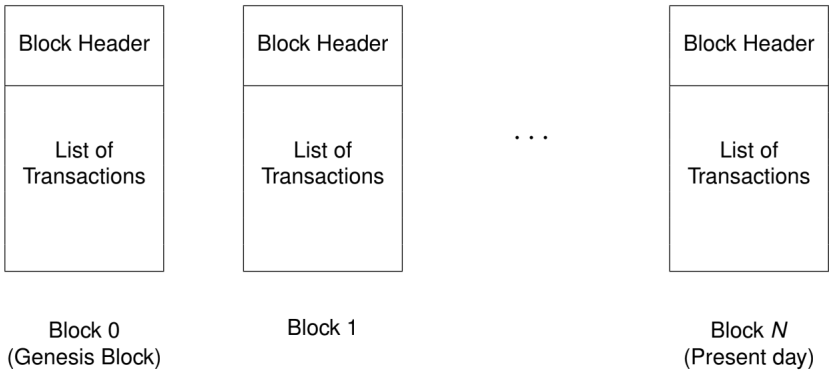

FIGURE 1.3: Block Chain

1.3 Shortcomings

The current shortcomings of Bitcoin are as follows:

- Every node has to agree upon all transactions. As a result, millions of nodes cannot be supported.
- Average time of a transaction to be included in blockchain is large. It varies from 10 minutes to several days making it infeasible for some business interactions.
- Small value transactions do not take place. Thus, a large number of transactions are excluded from the scope of Bitcoin.

For Bitcoin to emerge as one of the widely used payment methods, these shortcomings need to be addressed.

Chapter 2

Lightning Network

2.1 Need for Lightning Network

As seen in Chapter 1, there are shortcomings in the existing Bitcoin design preventing from becoming one of the widely used payment methods.

Solutions to shortcomings of Bitcoin are as follows:

Brute-Force Scaling

Increasing the global rate will lead to increase in number of transactions. The linear speed up will scale the rate by 2x, 4x etc. But it cannot meet the scalability requirement. It also leads to increase in bandwith requirement and may lead to congestion. Every transaction has to be agreed on every node, and hence has to pass through every node. Thus, more rate leads to more bandwith and more congestion.

Layered Architecture

Second solution can be splitting of the network similar to the layered architecture of Internet. In the layered architecture a packet does not reach every node, as a result maintaining consensus poses a challenge.

2.2 Implementation

Lightning network implements layered architecture for Blockchain along with maintaining consensus with the use of payment channels. [2]

2.2.1 Payment Channels

Say Alice and Bob want to make payments to each other.

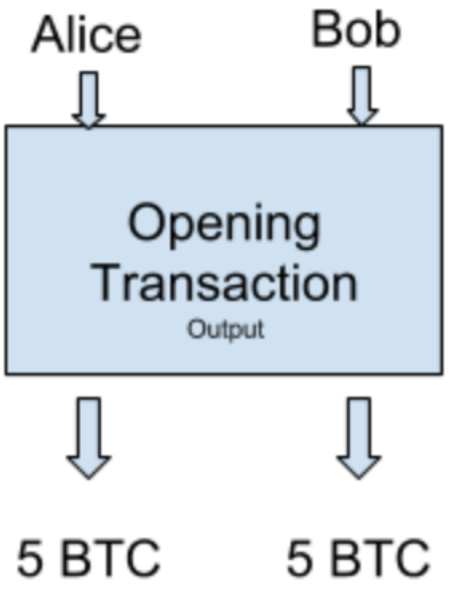

FIGURE 2.1: Funding Transaction

Alice and Bob open a 2 of 2 multisig payment channel. Both send money to a shared address and this address is a 2 of 2 multisig. This transaction is called as the *Funding transaction* as shown in Figure 2.1. 2 of 2 multisig implies both parties need to sign to spend from it. Funding transaction is recorded on blockchain. Commitment transactions are transactions made on Funding transaction. Any party can close the channel at any point of time without needing to inform the other party. Thus, 2 transactions (open and close) are needed for any payment channel.

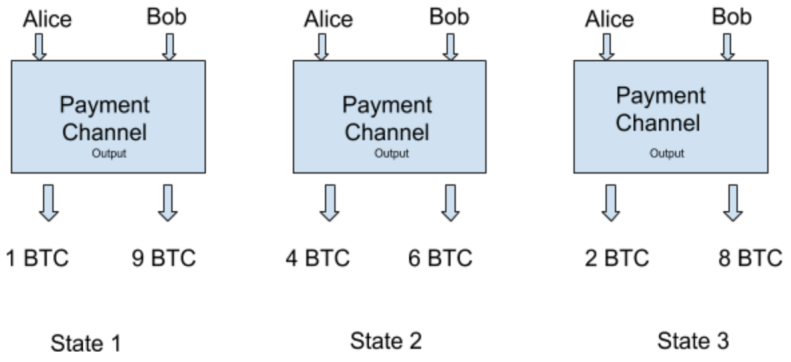

FIGURE 2.2: Various States present in Payment Channel

Many number of transactions (Figure 2.2) can take place over time without broadcasting to blockchain. Say initially Alice and Bob have funded 5 BTC each in the channel. Then Alice transfers 4 BTC to Bob (State 1). After some time Bob transfers 3 BTC to Alice (State 2). Then, Alice transfers 2 BTC to Bob (State 3). All these transactions are not broadcasted on the blockchain.

Any party should broadcast most recent state while closing the channel. A number of off-chain transactions happening instantly leads to increase in rate of transactions taking place.

2.2.2 Revoking Transactions

State 2 in the Figure 2.2 is a valid state and this transaction has signatures of both Alice and Bob. Therefore, Alice can broadcast State 2 instead of 3 and closes the channel. Alice can thus get a profit of 2 BTC by broadcasting State 2 than when State 3 is broadcasted on the blockchain.

To prevent revoking of transactions, timelocks are implemented for the state as shown in Figure 2.3

Alice has to share secret private key of previous state to Bob on creation of a new state (and vice versa). She has to wait for 1000 blocks confirmation time before receiving the output of 4 BTC. If Bob provides the secret key for the concerned state, he can get 4 BTC along with his share of 6 BTC. Thus, Bob can get all the 10 BTC immediately upon broadcast of any older state by Alice. Bob will have similar state 2 with 1000 blocks waiting time for his output. This prevents both the parties from revoking transactions.

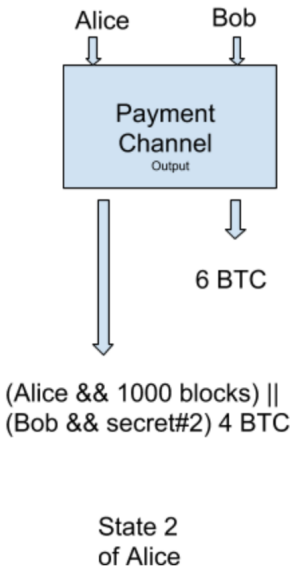

FIGURE 2.3: Prevention of revoking of transactions

The payment channel script for the above scenario is as follows [3]:

Input Script

`0 <SIG_A> <SIG_B> 0 {<FUNDINGTX_SCRIPT>}`

Zero after the signature indicates the secret is not revealed. The input script acts on the `FUNDINGTX_SCRIPT` which is the script for funding transaction.

Output Redeemscript

There are 2 redeem scripts, one for each output of the payment channel

1st Redeemscript:

```
OP_HASH160 <REVOCATION_HASH> OP_EQUAL
OP_IF
      <PubKeyB>
OP_ELSE
      <Timeout> OP_CHECKLOCKTIMEVERIFY OP_DROP <PubKeyA>
OP_ENDIF
OP_CHECKSIG
```

The above script indicates that if hash of Revocation pre-image supplied by Bob equals the required Revocation Hash, then the output funds goes to Bob else to Alice after a timeout.

2nd Redeemscript:

`<PubKeyB> OP_CHECKSIG`

The above script pays funds to Bob normally.

Redeeming Output using Revocation Image

`<SIG_B> <REVOCATION_KEY> <REDEEM_SCRIPT>`

If Bob wants to claim a output, he specifies the above script where the redeemscript is the one corresponding to the required output. The above script acts on the redeemsript. If the Revocation pre-image is correct, Bob can get the funds.

Claiming Timed-Out Output

`<SIG_A> 0 <REDEEM_SCRIPT>`

Alice claims the funds after delay if Bob cannot provide the key.

`OP_CHECKSIG` in the Redeem Script verifies the signatures of Alice `<SIG_A>` and Bob `<SIG_B>`.

2.2.3 Multihop Network

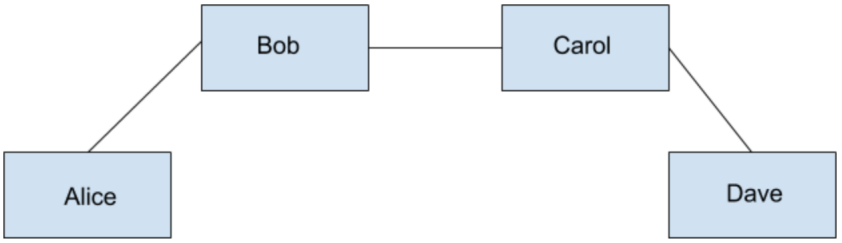

FIGURE 2.4: Multihop Network

Sometimes 1-off payments are needed i.e. need to transact only once. For example: buying pizza from Pizza Express. In such cases opening a payment channel only for one transaction is expensive. Multihop makes use of network of payment channels.

Say Alice wants to transact with Dave in the network of channels as shown in Figure 2.4. Alice has a channel with Bob, Bob with Carol and Carol with Dave. Alice can open a channel with Dave or transact on the Bitcoin blockchain. But if there is a path in the network of payment channels, Alice can transact without sending any data to actual Bitcoin network. Alice can thus avoid paying Bitcoin fees on the Bitcoin blockchain and pay smaller amount of fees to Bob and Carol.

The **Multihop** network is implemented as follows:

Alice lets Dave know that she wants to do a transaction with him. Dave will output a random number R, take hash of it H and sends H to Alice. Alice and Dave are not

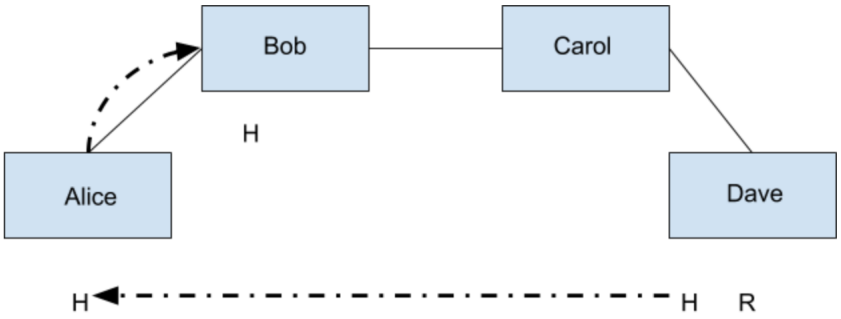

FIGURE 2.5: Transaction in Multihop - I

connected on the payment channel network but are connected on the TCP/IP network which is authenticated and encrypted. Hence, can send securely H over the TCP/IP network. Alice makes a transaction to Bob using H as shown in Figure 2.5

2.2.4 Hashed Timelock Contract

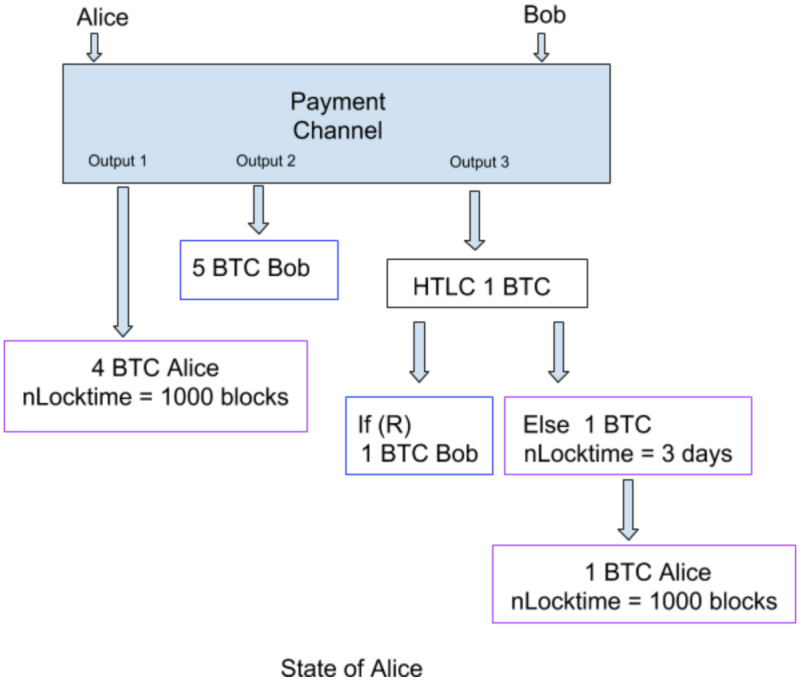

FIGURE 2.6: Hashed Timelock Contract of Alice

Hashed Timelock contracts (HTLC) are different from normal transactions over payment channel. There are three possible outputs unlike two in normal transactions as shown in Figures 2.6 and 2.7. The first two outputs are the same as the two outputs of normal transactions, one each for Alice and Bob. The third output is spendable by Bob

if he signs it along with giving the preimage R, or it can be spendable by Alice after lock time period of 3 days. The condition of 1000 block locktime at the output of Alice in Figure 2.6 (of Bob in Figure 2.7) is to protect revoking of transactions as explained in the Section 2.2.2. This prevents Alice from broadcasting this state if there is a new transaction after this.

HTLC Sender (Alice) Redeemscript

```
OP_HASH160 OP_DUP <R_HASH> OP_EQUAL
OP_SWAP <REVOCATION_HASH> OP_EQUAL OP_ADD
OP_IF
      <PubKeyB>
OP_ELSE
      <HTLC_Timeout> OP_CHECKLOCKTIMEVERIFY
     <Delay> OP_CHECKSEQUENCEVERIFY
      <OP_2DROP>
      <PubKeyA>
OP_ENDIF
OP_CHECKSIG
```

If any of the 2 hash among Revocation Hash or hash of the R value matches, then output is paid to Bob. Else, Alice has to wait for a timeout to claim for the output. Additionally Alice has to wait for delay, which ensures Bob can use revocation if applicable. Lastly, the script verifies the signature.

HTLC Receiver (Bob) Redeemscript

```
OP_HASH160 OP_DUP <R_HASH> OP_EQUAL
OP_IF
     <Delay> OP_CHECKSEQUENCEVERIFY
      <OP_2DROP>
      <PubKeyB>
OP_ELSE
      <REVOCATION_HASH> OP_EQUAL
      <OP_NOTIF>
           <HTLC_Timeout> OP_CHECKLOCKTIMEVERIFY OP_DROP
     <OP_ENDIF
      <PubKeyA>
OP_ENDIF
OP_CHECKSIG
```

This is similar to Alice's script. Here, delay at the output of Bob gives enough time to Alice to use revocation if applicable.

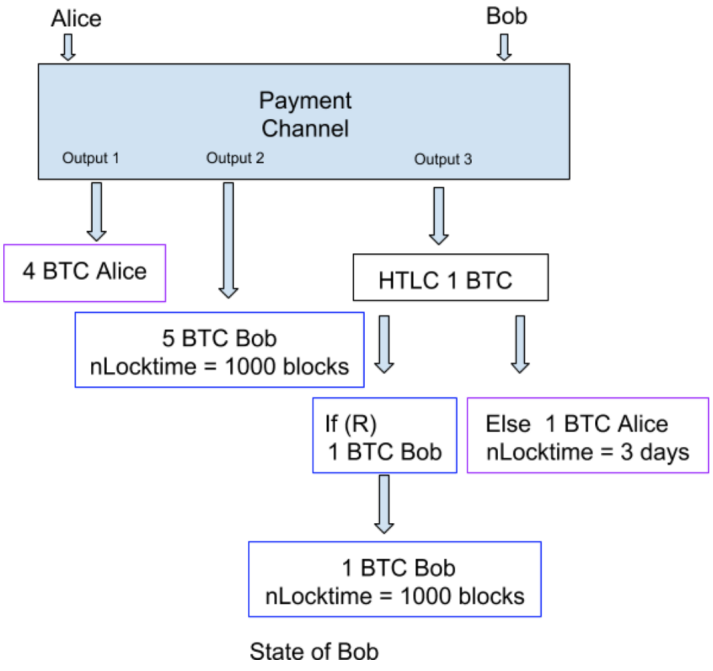

FIGURE 2.7: Hashed Timelock Contract of Bob

Redeeming HTLC Output

`<SIG_B> <HTLC_R_VALUE> <HTLC_REDEEM_SCRIPT>`
Claiming the output by specifying the R value.

Claiming Timed-Out HTLC Output

`<SIG_A> 0 <HTLC_REDEEM_SCRIPT>`
Claiming the output after time-out. 0 is used to specify no key.

Claiming Revoked HTLC Output

`<SIG_A> <REVOCATION_KEY> <HTLC_REDEEM_SCRIPT>`
This is used to claim a revoked output using revocation key.

After decrypting the message from Alice, Bob transmits next to Carol. Bob is not aware whether Carol is the intended receiver. Similarly, if Carol knows R, she can get the coins, else transmits to Dave as shown in Figures 2.8 and 2.9.

Dave knows R and hence can get the coins. He can close the channel to broadcast immediately or keep channel open and tell Carol R. Carol gets back her coins paid to Dave by telling the information of R to Bob. Similarly Bob gets back his coins from Alice by giving R. Finally Alice gets R which is a confirmation that Dave received the

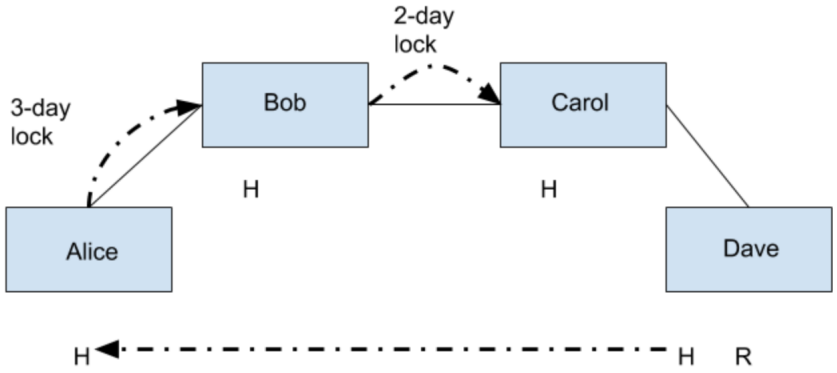

FIGURE 2.8: Transaction in Multihop - II

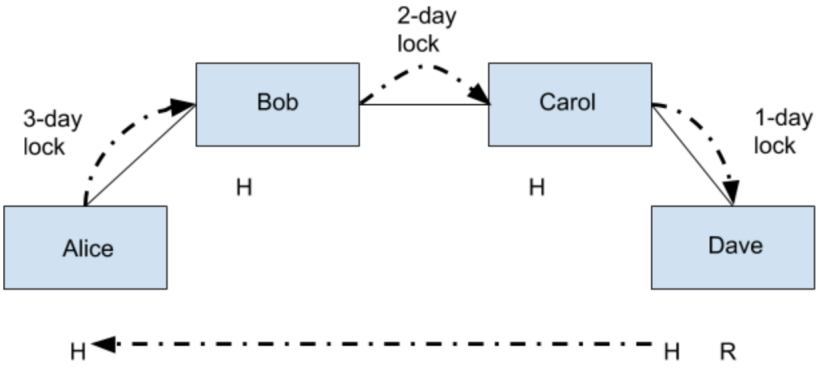

FIGURE 2.9: Transaction in Multihop - III

intended money. The contingent payments can be removed and balances updated, i.e. the states do not have to depend on R anymore.

The states are updated on receiving R and do not depend on R anymore as shown in Figure 2.10.

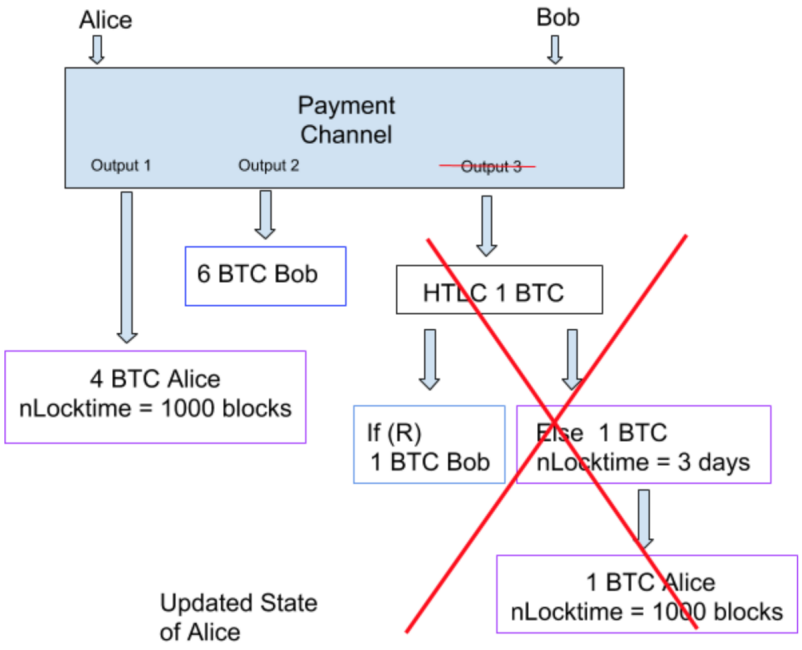

FIGURE 2.10: Updated State of Alice after successful payment to Dave

In such manner Alice can transact with Dave using multihop using Hashed Timelock Contract (HTLC). Similarly anyone can route payment to anyone else in the network if the sender knows a path to the destination. The routing of payments is explored in Chapter 3 using a routing procotol: Flare routing.

Chapter 3

Flare Routing

3.1 Introduction

As seen in Chapter 2, the funding transaction of each payment channel of lightning network is broadcasted on the blockchain as shown in Figure 3.1. Lightning network is used for subsequent off-chain transactions. Payments can be routed in the network using multihop (Section 2.2.3).

Flare Routing [4] is a type of routing algorithm that can be used to route payments

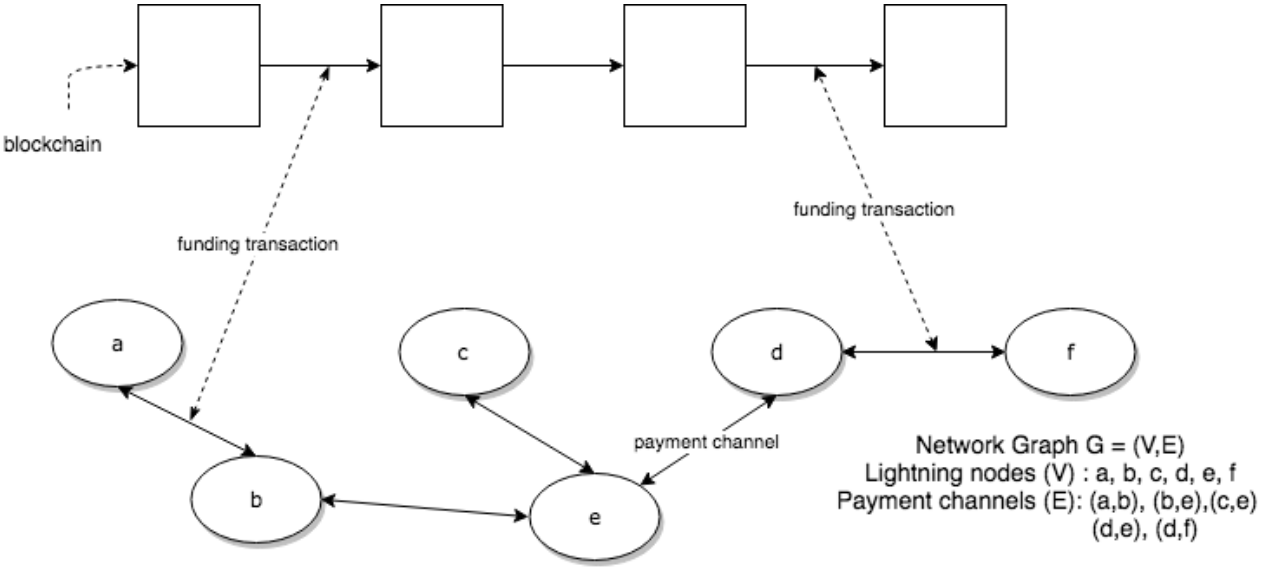

FIGURE 3.1: Lightning Network

within the lightning network. It is a trustless source routing protocol, wherein the source node has the power of decision of choosing the route. The intermediate nodes on path as well as the destination node have no knowledge of the route. The algorithm is a hybrid of proactive and reactive algorithms.

Need for Hybrid Algorithm:

There is a tradeoff between scalability and reliability to choose an algorithm. Proactive (static) algorithms are reliable but not scalable wheras reactive algorithms (dynamic) are scalable but not reliable. Hybrid algorithm achieves the balance between reliability and scalability.

Flare routing algorithm proactively gathers information about network topology in the form of payment channels and beacons and dynamically ranks routes on the basis of dynamic characteristics such as channel state, capcacity of channel and the time required in polling the message on the route.

Flare routing is scalable to a size of several hundred thousand nodes. It also achieves finding routes to any node with high probability.

Hence, flare routing algorithm is scalable as well as reliable.

3.2 Mathematical Model

Lightning network in Figure 3.1 can be modelled as a graph $G = (V, E)$ with nodes as vertices V of graph and payment channels as edges $E \subseteq V^2$ of the graph as shown in Figure 3.2. Each payment channel $e \in E$ has total capacity $Cap(e)$

where $Cap : V^2 \to [0, +\infty)$ is a binary symmetric function.

such that $\forall v_1, v_2 \in V (v_1, v_2) \notin E \iff Cap(v_1, v_2) = 0$ that is, the total capacity of non-existent channels equals 0.

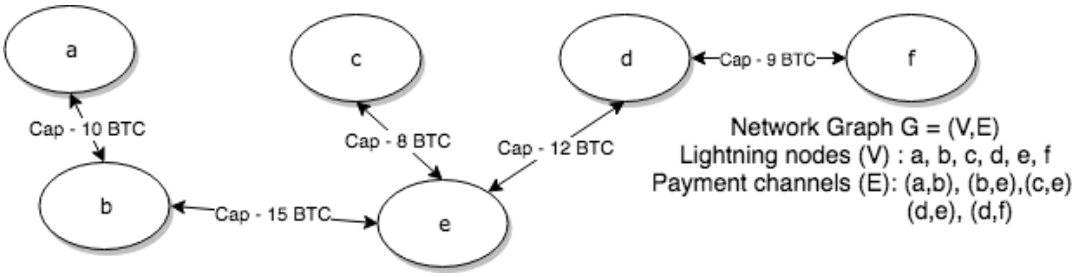

FIGURE 3.2: Network Graph

Definitions 1. Hop Distance:
Hop distance between two nodes is the minimum number of channels connecting the two nodes.

$d_{node}(x, y) = min(n \in \mathbb{N} : \exists v_1, ..., v_{n+1} \in V; x = v_1, y = v_{n+1};$
$\forall i \in 1, ..., n$ such that $(v_i, v_{i+1}) \in E)$

Hop distance between a node and a channel is the minimum of hop distances of the node with the nodes of channel.
$d_{channel}(x, e) \equiv d_{channel}(x, (y, z)) = min(d_{node}(x, y), d_{node}(x, z))$

2. Adjacent nodes:
Adjacent nodes $Adj(v)$ for $v \in V$ are nodes at hop distance 1. That is the nodes with which v has opened a payment channel.
$Adj(v) = \{z \in V : d_{node}(v, z)\}$

3. Node Address:
Node Address is a 256-bit integer computed as SHA-256 of the public key of the node.

4. Address Distance:
Address Distance is the bitwise XOR of node addresses.
$\rho(x, y) = address(x) \oplus address(y)$

5. Routing Table:
Routing table of each source node is the subgraph of network which is the view of the network of source node. It contains a combination of neighboring channels and beacon nodes. Beacon nodes are randomly distributed faraway nodes that helps in expansion of the view of network of source node.

6. Neighborhood Radius:
Neighborhood Radius r_{nb} is the maximum hop radius to neighbor nodes in its routing table. Source node stores all the neighborhood channels between all node neighbors in its routing table.

3.3 Route discovery (Proactive)

The proactive part of routing algorithm involves filling up of routing tables of all nodes that consist of 2 parts: Neighborhood Discovery and Beacon Discovery.

3.3.1 Neighborhood Discovery

Neighborhood discovery is the procedure of identifying and adding payment channels in local neighborhood to the routing table.

Neighborhood discovery makes use of the following messages:

- `NEIGHBOR_HELLO` transmits full routing table of a lightning network node.
- `NEIGHBOR_UPD` transmits incremental change of routing table since last update message sent.
- `NEIGHBOR_RST` request for complete routing table. It is sent when message sender wishes to receive complete routing table from message receiver if the update information was supposedly lost or if sender's routing table gets corrupted.
- `NEIGHBOR_ACK` acknowledgement to `NEIGHBOR_HELLO` or `NEIGHBOR_UPD` inorder to signal that the node has properly received the message.

A node accepts `NEIGHBOR_*` messages only from adjacent nodes. After receiving a `NEIGHBOR_HELLO` or `NEIGHBOR_UPD`, the node adds those channels that are in the neighborhood radius (r_{nb}) and ignores the update to other channels. The algorithm is outlined below:

Algorithm on receiving `NEIGHBOR_HELLO` or `NEIGHBOR_UPD` is:

Input

- Message recepient u
- Message sender v
- Set of new channels in the message $M \subset E$
- Set of closed channels in the message $M_r \subset E$
- Neighborhood radius r_{nb}

Output

- Updated routing table $u.RT$

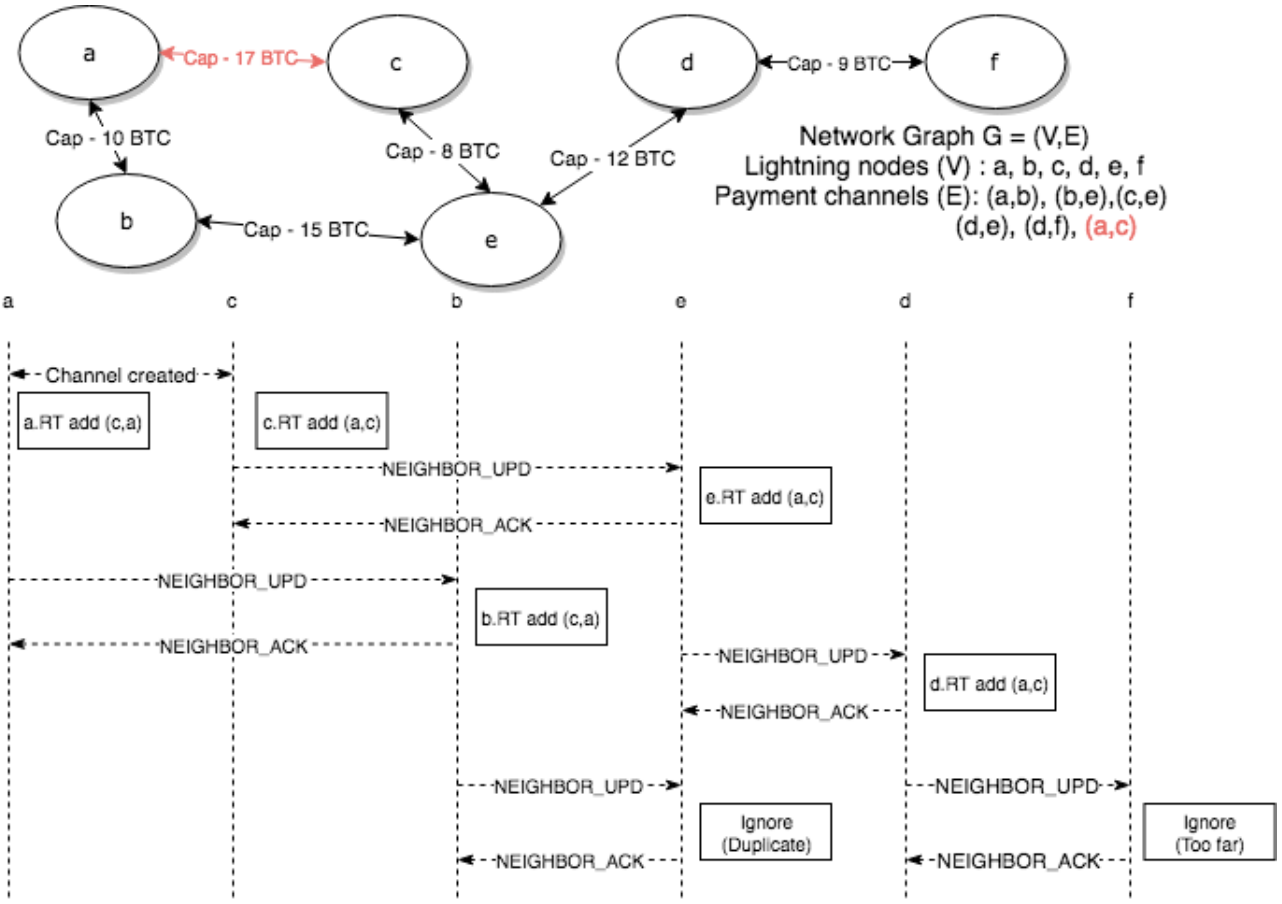

FIGURE 3.3: NEIGHBORHOOD DISCOVERY process upon introducing channel (a,c)

$RT_{pre} := u.RT \cup M$

$G_{pre} := (Nodes(RT_{pre}), RT_{pre})$

$\forall e \in M \setminus u.RT$

Compute min hop distance $d_{channel}$ from u to e

If $d_{channel} \leq r_{nb}$

Get $Cap(e)$

If e is open

$u.RT := u.RT \cup \{e\}$

$\forall e \in M_r \cap u.RT$

Verify if e is closed

If e is closed

$u.RT := u.RT \setminus \{e\}$

In the Figure 3.3, we see that a new payment channel **(a,c)** is opened in the network. The network has a neighborhood radius r_{nb} equal to 2. Nodes **a** and **c** informs about this update to their adjacent nodes. Node **a** inform to node **b** and node **c** informs to node **e**. Since node **a** is in the neighborhood radius of node **b** it adds the update to its routing table, similarly node **e** adds update to its routing table. Now, node **b** sends the update to node **e**, but since it has already received the update, node **e** ignores this message. Node **e** sends the update to node **d**, as node **c** is in the neighborhood radius of node **d**, it adds the update to its routing table. Node **d** sends the update to its adjacent neighbor node **f**, but since neither nodes **a** and **c** lie in the neighborhood radius of node **f**, this update is ignored. The update thus stops getting propogated beyond node **d** in the example graph. In such manner, any update gets propogated in the network with neighborhood nodes adding the update to their routing table.

3.3.2 Beacon Discovery

Beacon discovery is the process of finding nodes close in address space to source node and adding the beacons and paths to the beacons in the routing table of source node. These beacons are randomly distributed faraway nodes that helps to expand the view of the network known to source node.

Beacon discovery involves the use of the following messages:

- `BEACON_REQ` request to become a beacon candidate.
- `BEACON_ACK` contains
  - acknowledgement of node v confirming beacon acceptance.
  - set of possible candidates C_v.
  - Mapping $M_v : C_v \rightarrow v.RT^+$; $M_v(z)$ are known route(s) from v to z.
- `BEACON_SET` message to convey node selected as beacon.

In beacon discovery, the source node looks for closest nodes in address space in its routing table and sends request to them. If these nodes find in turn better beacon candidates which are closer to source node, they inform the source. The source then sends request to these candidates, this happens till no further closer nodes can be found. The source node sets the closest candidates as beacons. The algorithm is as follows:

Algorithm for selection of beacons is:

Input

- Current node u
- Number of beacons N_{bc}

Output

- Updated set of beacons $u.bc$

Initial variables

- Initialize N_{bc} unprocessed beacons B with minimum address distance.
- Initialize processed nodes $U := \emptyset$
- Initialize responsive nodes $R := \emptyset$

while $|B| > 0$

$v :=$ arg min$\rho(z, u)$ where $z \in B$

$U := U \cup \{v\}$; $B := B \setminus \{v\}$

Send `BEACON_REQ` to v and wait for `BEACON_ACK`

If `BEACON_REQ` times out

continue to next node

Let `BEACON_ACK` response is (C_v, M_v); $R := R \cup \{v\}$

$\forall z \in C_v \setminus (B \cup U)$

Verify $\rho(z, u) < \rho(v, u)$

Verify $M_v(z)$ contain path(s) from v to z.

$B := B \cup \{z\}$

while $|B| > N_{bc}$

$z^* =$ arg min$d(u, z)$ where $z \in B$; $B := B \setminus \{z^*\}$

$\forall v \in R$ sort by increasing $\rho(v, u)$

Send `BEACON_SET` to v

$u.bc := u.bc \cup v$

Add paths from u to v to $u.RT$

if $|u.bc| := N_{bc}$

exit

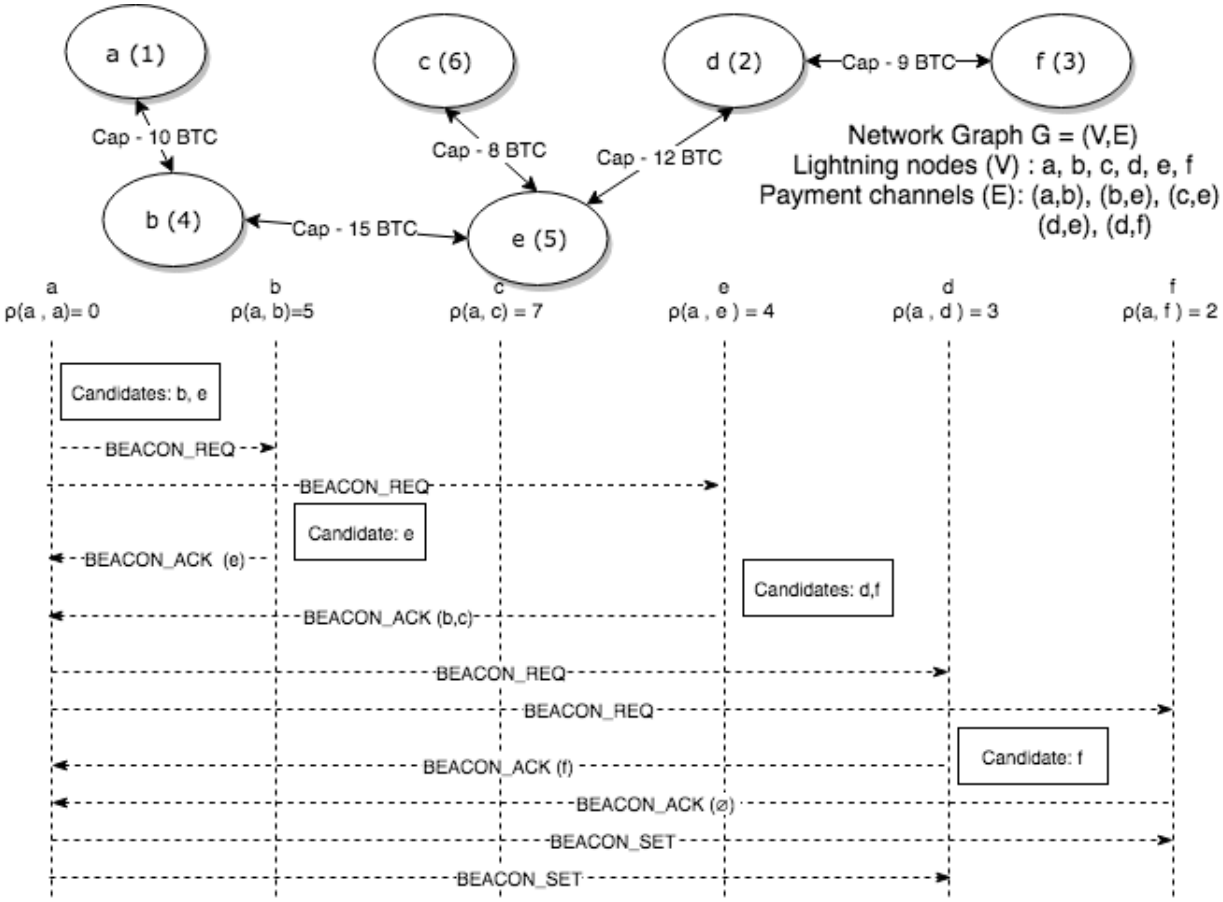

FIGURE 3.4: Beacon Discovery

Figure 3.4 describes the process of finding beacons for node **a** when number of beacons $N_{bc} = 2$. Node **a** sends `BEACON_REQ` to nodes **b** and **e** (the closest nodes in its routing table). Node **b** finds a closer node than itself **e** in its routing table and conveys it to node **a**. Similarly node **e** finds better candidate nodes **d** and **f** in its routing table. Node **a** sends `BEACON_REQ` to nodes **d** and **f**. After receiving `BEACON_ACK` from the two, node **a** sends a final message `BEACON_SET` confirming beacon selection as nodes **d** and **f** are the closest nodes it has found to itself during the beacon search algorithm. In this manner, beacon discovery takes place.

3.4 Route Selection

This part of routing algorithm occurs when a node wants to route payment to another node in the network using multihop. If source node cannot find route to the destination, it queries for the destination's routing table and if that is also unsuccessful it queries for routing tables from nodes which are close to destination address and to which it already knows a path. The algorithm of this process is outlined below:
Types of messages used are:

- `TABLE_REQ` request for routing table
- `TABLE_RESP` response containing full routing table

Algorithm to find routes from source to destination:

Input

- Payment sender s
- Payment recepient r
- Maximum number of paths $k \geq 1$
- Maximum number of queried nodes $N_{tab} \geq 1$

Output

- Set of routes from s to r P ($P = \emptyset$ means refusal for routing)

Initial variables

- Initialize combined routing table $RT_{co} := s.RT$
- Initialize found routes $P := \emptyset$
- Initialize set of queried nodes $U := \emptyset$

while $|P| < k$ and $|U| < N_{tab}$

$P := P \cup$ Paths(s, r, RT_{co}, k)

if $|P| < k$

$c :=$ arg min$\rho(z, r); z \in Nodes(RT_{co}) \setminus U$

$RT_{co} := RT_{co} \cup c.RT$

$U := U \cup \{c\}$

return P

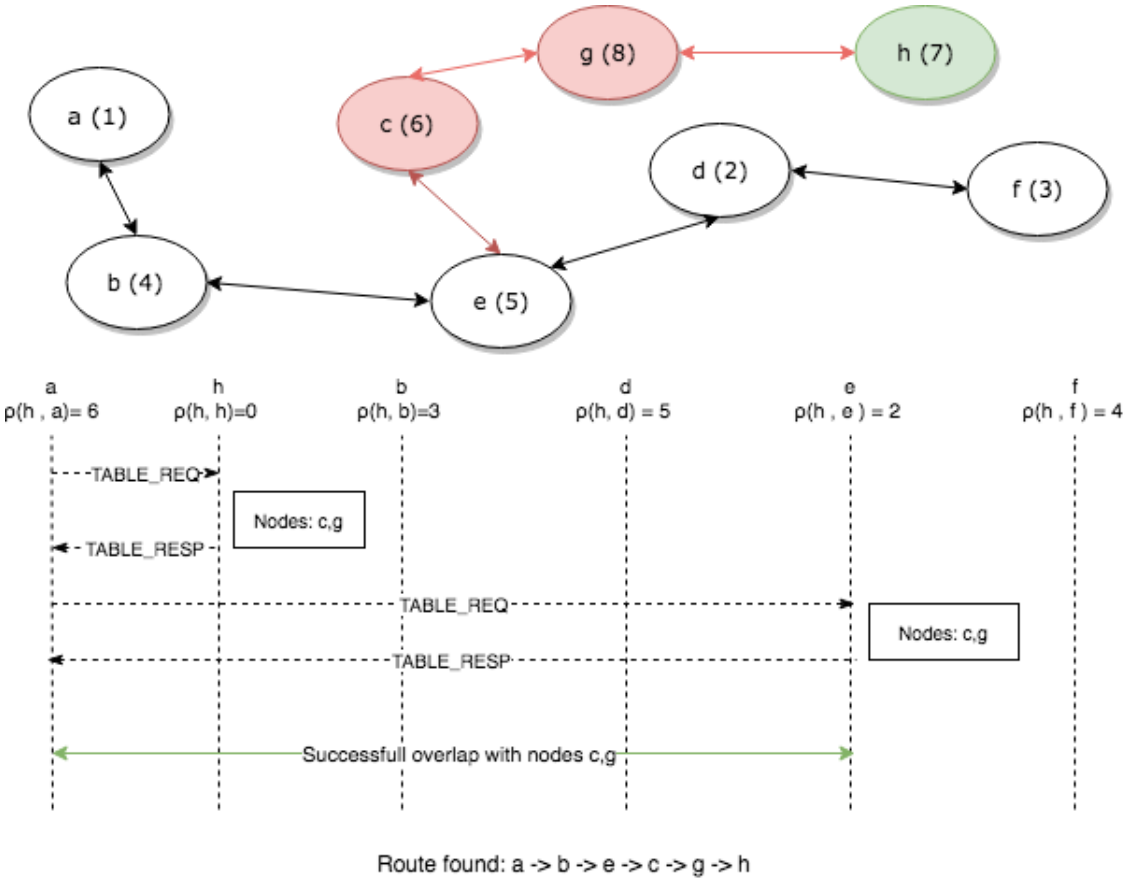

FIGURE 3.5: Route Selection

In the Figure 3.5 node **a** wants to route payment to node **h** but it does not know any path to node **h**. It queries for routing table from node **h** and gets the information that nodes **c** and **g** are in its routing table but node **a** does not know a path to them either. Node **a** then queries node **e** which is closest to node **h** in address space and finds out that nodes **c** and **g** are in routing table of node **e**. It is then successful in finding a path to node **h** which is **a** $\rightarrow$ **b** $\rightarrow$ **e** $\rightarrow$ **c** $\rightarrow$ **g** $\rightarrow$ **h**. In such manner route selection happens using the routing tables of nodes.

3.5 Route Ranking

All the routes that are obtained in Section 3.4 are ranked upon their dynamic characteristics. If a route fails some checks such as some channel on the route does not have enough capacity or the channel cannot be verified on blockchain or the polling message on the route times out then that route is discarded. Routes that pass the check are given a rank rr_p based on some characteristics such as cumulative fees required on the routes. The first route found that is above some predefined threshold rr_{good} is selected as final route. The algorithm is as follows:

Algorithm for route ranking

Input

- Value to transfer f
- Set of Routes P
- Cumulative fees on p route C_p
- Polling message timeout t_{poll}

Output

- Route ranking $rr(p) \in [-\infty, +\infty)$

Message

- `RANK_POLL` polling message to recepient

$\forall p \in P$

Verify all channels in route on blockchain

if any channel does not pass check

$rr(p) := -\infty$

Send polling message along the route p

else if message time outs to t_{poll}

$rr(p) := -\infty$

else if any channel cannot route payment

$rr(p) := -\infty$

else

Calculate route ranking $rr(p) := 1/C_p$

if $rr(p) \geq rr_{good}$

Select p as final route

return p

3.6 Simulation

Watts-Strogatz is a small-world connected graph that can be used to simulate the lightning network. Flare routing algorithm is implemented to examine reachability on a Watts-Strogratz graph of order = 2000 where each node is joined with its 4 nearest ring neighbors and has rewiring probability of 0.3. Additionally, maximum number of queried nodes N_{tab} is chosen as 10 and neighborhood scan radius r_{nb} as 2. Number of beacons N_{bc} is varied from 0 to 8.

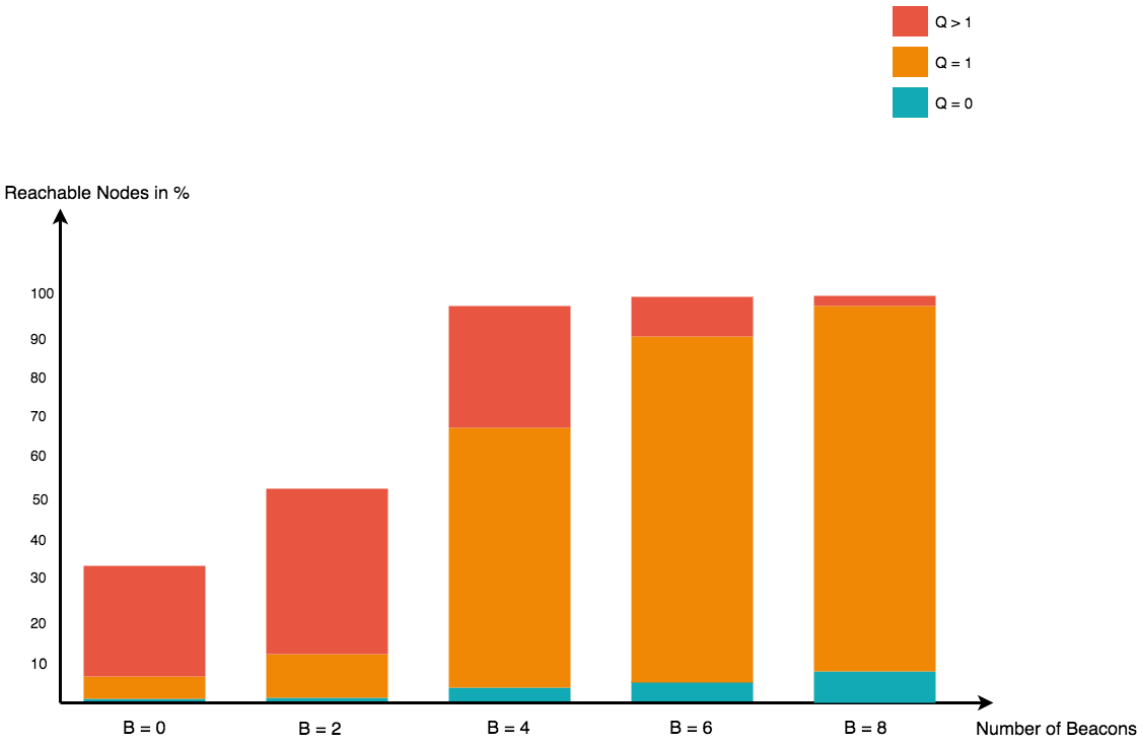

FIGURE 3.6: Reachability Experiment

In Figure 3.6, **B** is the number of beacons and **Q** is the number of queried nodes. As **B** and **Q** are varied, the reachability changes. We see that for B = 4 and Q = 1, reachability is 70%, that is just by querying routing table of destination node, payment can be routed in the network to upto 70% of nodes. By querying upto 10 nodes, the reachability increases to 99% which implies payment can be routed to any node with high probability. The reachability increases with more number of beacons. For B = 6 and Q = 1, 90% of destination nodes are reachable and for B = 8 this number increases to 97%.

There is a trade off between number of queried nodes and number of chosen beacons. With more number of beacons, less number of nodes need to be queried but that increases the size of the routing table. With less beacons, more nodes need to be queried which increases the time required to find a route.

Hence, a payment can be routed to any node with high probability by having minimum routing state which comprises of a few distant beacon nodes and local neighborhood channel

Chapter 4

Conclusion and Future Work

4.1 Conclusion

Flare routing protocol achieves reachability to any node in thelightning network with high probability by achieving minimum routing state. The minimum routing state is achieved by storing a few beacon nodes that helps expand the view of the network of the source node along with the local neighborhood channels in the routing table. The routing algorithm has a trade off between number of beacons and number of queried nodes and optimal number has to be chosen to achieve high reachability.

4.2 Future Work

Future work involves understanding Lightning network in more detail, analysis of Flare routing protocol, along with enhancing the capabilities of simulator for Lightning Network in Haskell.

Bibliography

[1] Saravanan Vijayakumaran. *An Introduction to Bitcoin*. October 2017. URL https://www.ee.iitb.ac.in/~sarva/bitcoin/bitcoin-notes-v0.1.pdf.

[2] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable off-chain instant payments. January 2016. URL https://lightning.network/lightning-network-paper.pdf.

[3] Rusty Russell. Reaching the Ground with Lightning. November 2015. URL https://github.com/ElementsProject/lightning/blob/master/doc/deployable-lightning.pdf.

[4] Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. Flare: An Approach to Routing in Lightning Network. July 2016. URL https://bitfury.com/content/downloads/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016.pdf.