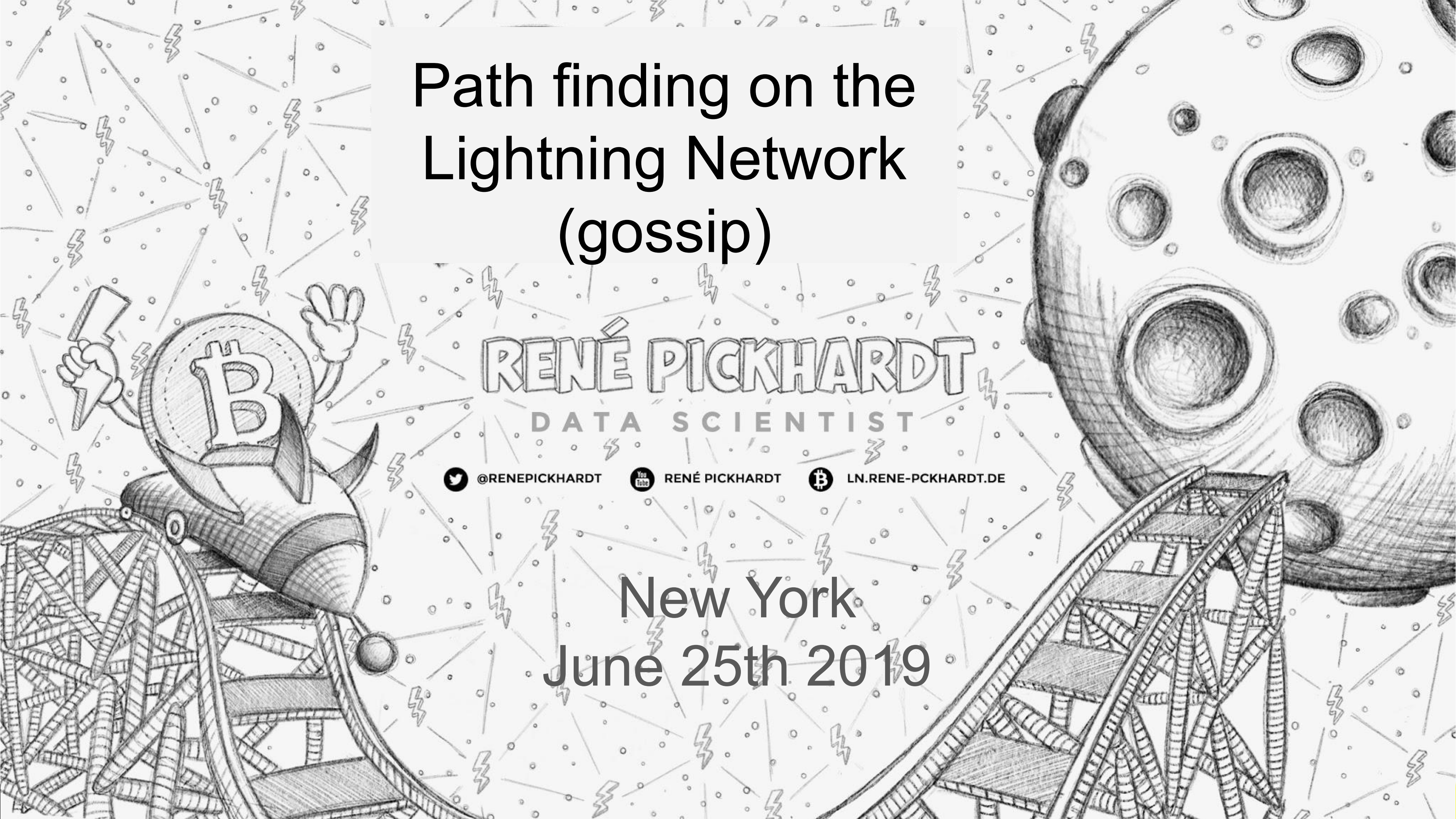
Path finding on the Lightning Network (gossip)



New York
June 25th 2019

Disclaimer

- Please note the copyright notice at the end of the slide deck
- The pdf version - which you probably read has some weird formatting problems.
    - https://docs.google.com/presentation/d/1-eyceLISmcLpbPJLzj6_CnVYQdo1AUP3y5XD716U-Lg has the source files without formatting problems and with animations
- This slidedeck is crowd funded (see last slide) if you wish to learn more or contribute
- I do not take any guarantee that the information in this slide deck are 100% correct.
    - Sometimes I made simplifications: which I point out
    - Also I could just have misunderstood something or just made a mistake

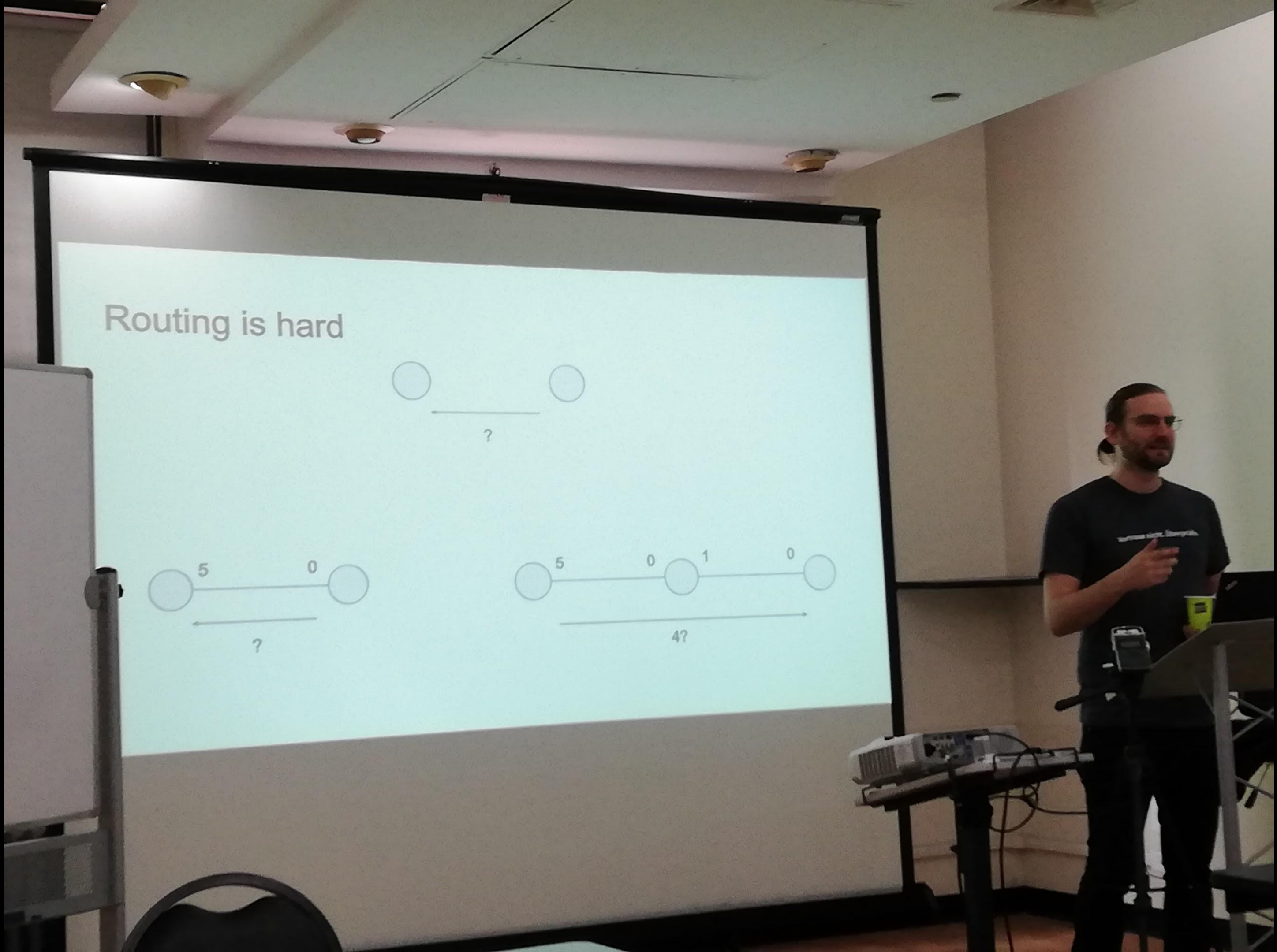
Outline

- A quick Recap of the 2018 Lightning Residency
- Comparing IP Routing with Routing on the Lightning Network
- Motivation for the difficulty of the pathfinding problem
- Actual Path finding strategies

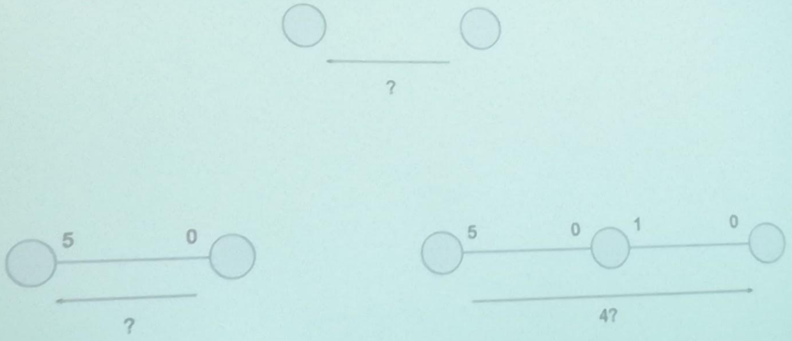
A quick recap of chaincode labs lightning residency

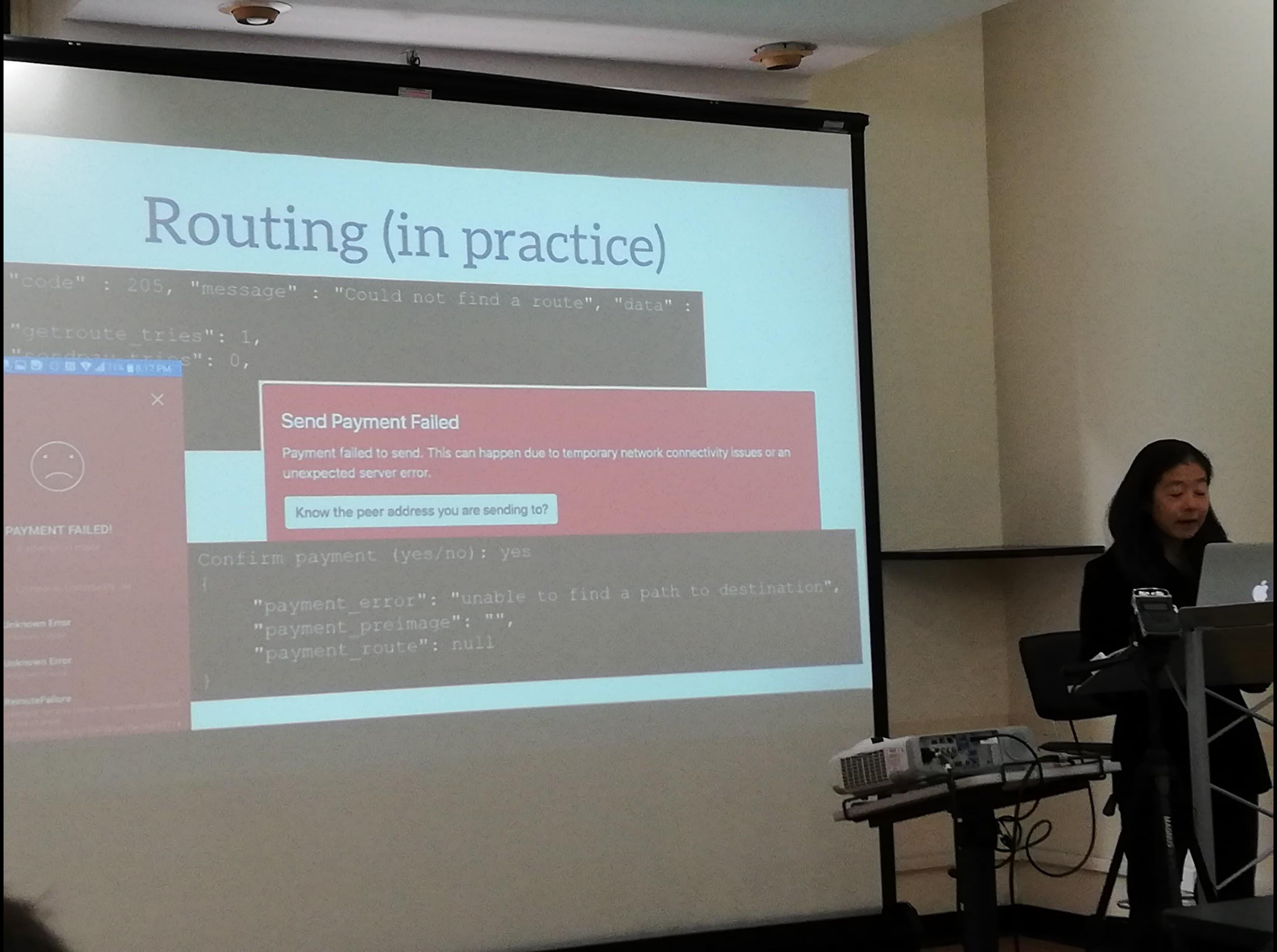
Chaincode Lightning Residency

October 22nd - October 26th 2018



Routing is hard





Routing (in practice)

```
"code" : 205, "message" : "Could not find a route", "data" :
"getroute_tries": 1,
"sendpay_tries": 0,
```

PAYMENT FAILED!

Unknown Error

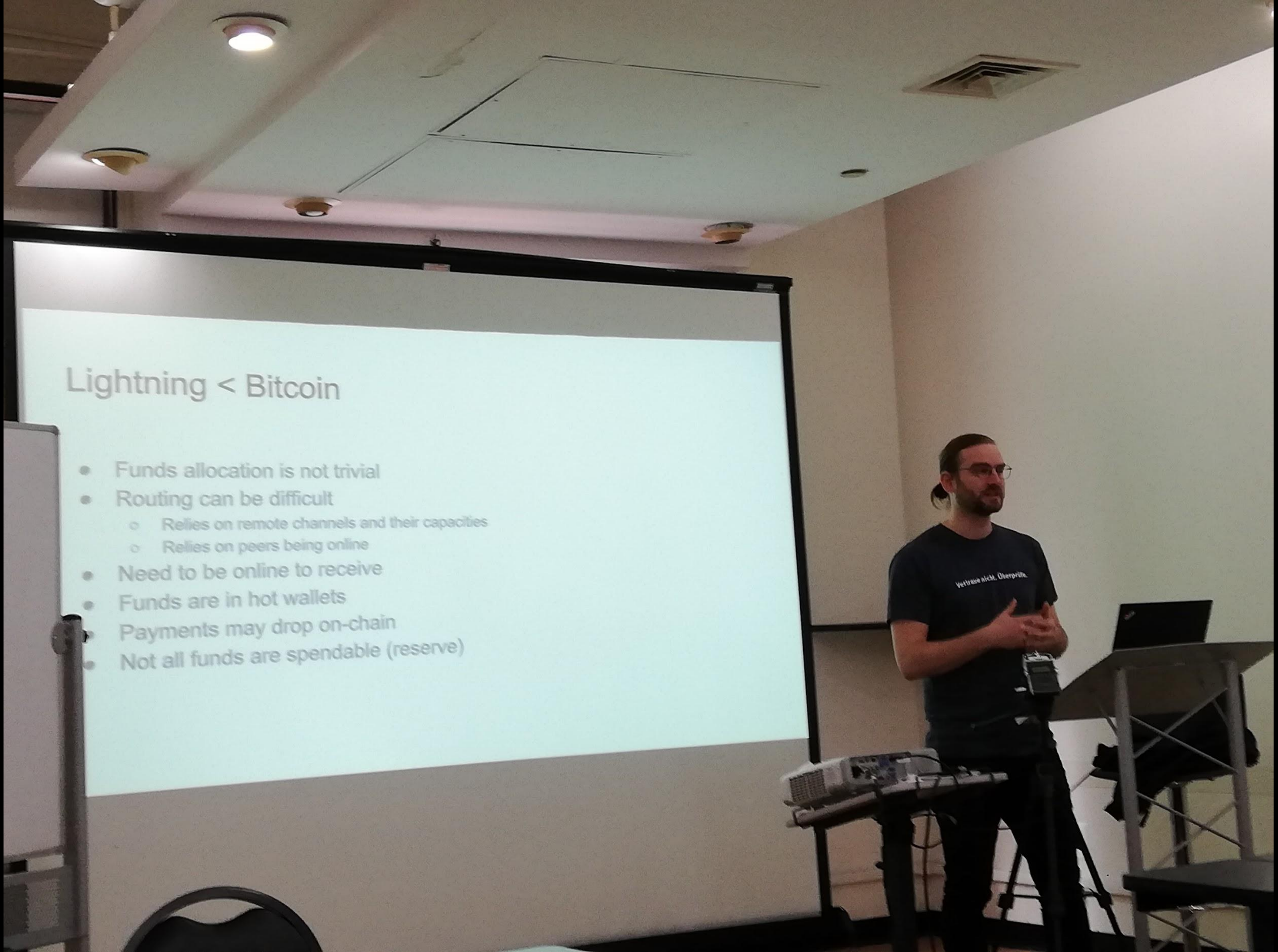
Unknown Error

RemoteFailure

Send Payment Failed

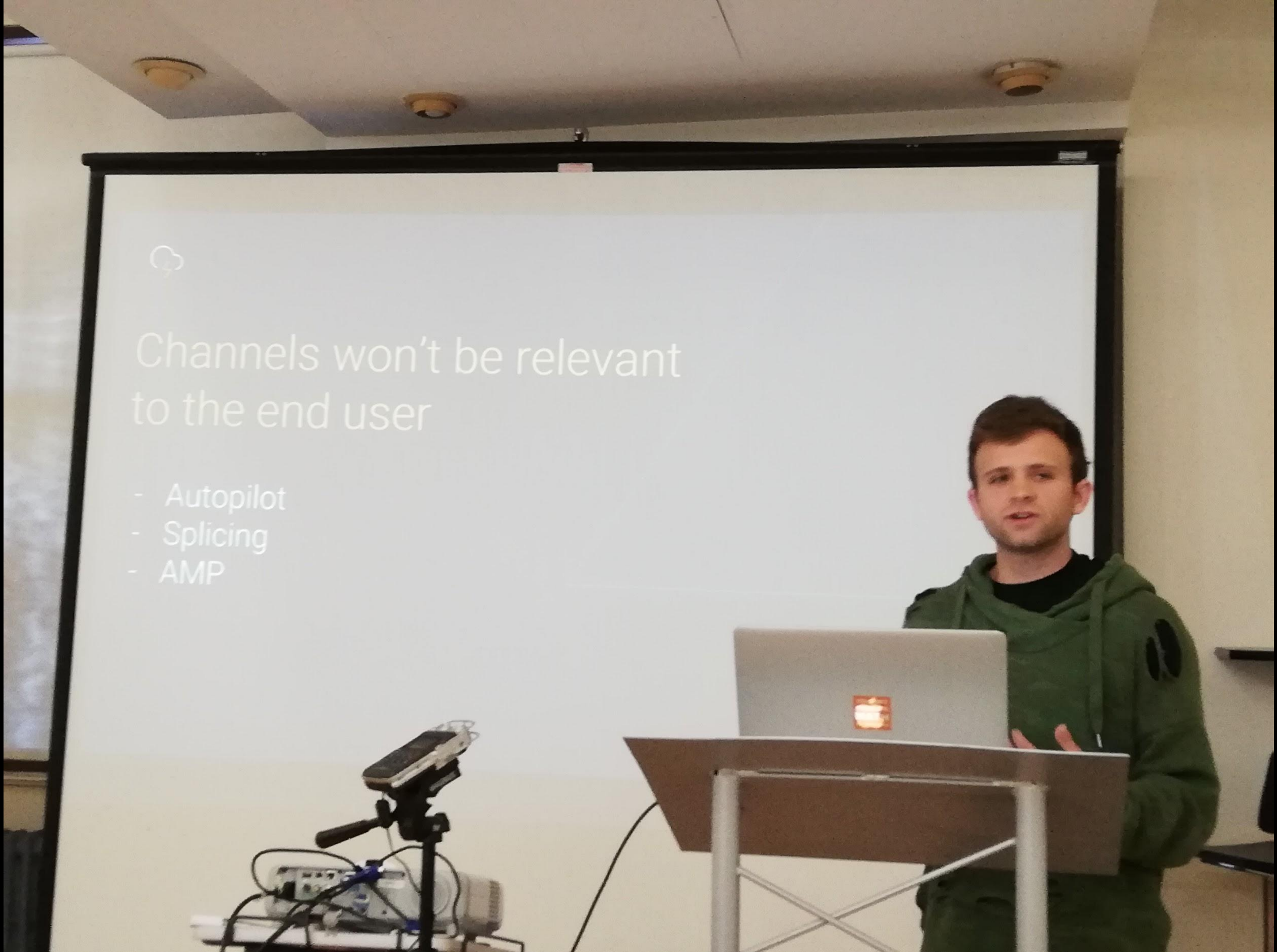
Payment failed to send. This can happen due to temporary network connectivity issues or an unexpected server error.

```
Confirm payment (yes/no): yes
{
    "payment_error": "unable to find a path to destination",
    "payment_preimage": "",
    "payment_route": null
}
```



Lightning < Bitcoin

- Funds allocation is not trivial
- Routing can be difficult
  - Relies on remote channels and their capacities
  - Relies on peers being online
- Need to be online to receive
- Funds are in hot wallets
- Payments may drop on-chain
- Not all funds are spendable (reserve)



Channels won't be relevant to the end user

- Autopilot
- Splicing
- AMP

Wrong!!!

Routing is actually solved.

Pathfinding is seems to be the hard part

I want you !

These are open problems and you can make a huge difference!

IP Routing vs Onion Routing

(no BOLT)

A quick recap of IP Routing

- A node processes the IP header of the IP package it receives
    - Extracts sender and recipient
    - Dissects the IP address of the recipient into
        - Network part (Hierarchical address scheme!)
        - Host part
- A router looks up the Network part in its routing table
    - If network is known send to the next link (via link layer protocol)
    - Otherwise send to the default route
- Routing tables are dynamically updated for example via Border Gateway protocol
- The concept is called best effort routing (and is amazingly strong)
    - Address of the IP package states "I want to go to Chaincodelabs"
    - Router: That is in New York
    - Router: I send you on this road as it gets you closer to New York. Ask again later!

Comparing IP Routing with Onion Routing

	IP - Routing	Onion Routing
Type		
Data format		
Sender / Recipient		
Address Space		
Edgeweights		
Topology / Information propagation		
DoS - Attacks		
Pathfinding		

Best effort has been proven to work for decades

	IP - Routing	Onion Routing
Type	Best effort	Source based
Data format		
Sender / Recipient		
Address Space		
Edgeweights		
Topology / Information propagation		
DoS - Attacks		
Pathfinding		

Privacy: A good thing for users but not for algorithms

	IP - Routing	**Onion Routing**
Type	Best effort	Source based
Data format	Open headers	Encrypted headers
Sender / Recipient		
Address Space		
Edgeweights		
Topology / Information propagation		
DoS - Attacks		
Pathfinding		

With IP basically all information for routing is public

	IP - Routing	Onion Routing
Type	Best effort	Source based
Data format	Open headers	Encrypted headers
Sender / Recipient	Known to routing nodes	Unknown to Routing nodes
Address Space		
Edgeweights		
Topology / Information propagation		
DoS - Attacks		
Pathfinding		

Address space can help with pathfinding in IP

	IP - Routing	Onion Routing
Type	Best effort	Source based
Data format	Open headers	Encrypted headers
Sender / Recipient	Known to routing nodes	Unknown to Routing nodes
Address Space	Logical - hierarchical overlay network (reflecting geographical properties)	Fully decentralized P2P network
Edgeweights		
Topology / Information propagation		
DoS - Attacks		
Pathfinding		

Lightning edges change with every payment!

	IP - Routing	Onion Routing
Type	Best effort	Source based
Data format	Open headers	Encrypted headers
Sender / Recipient	Known to routing nodes	Unknown to Routing nodes
Address Space	Logical - hierarchical overlay network (reflecting geographical properties)	Fully decentralized P2P network
Edgeweights	Mostly static (bandwidth of links)	Highly dynamic (fees, balance)
Topology / Information propagation		
DoS - Attacks		
Pathfinding		

Gossip protocols are an easy target for spam

	IP - Routing	**Onion Routing**
Type	Best effort	Source based
Data format	Open headers	Encrypted headers
Sender / Recipient	Known to routing nodes	Unknown to Routing nodes
Address Space	Logical - hierarchical overlay network (reflecting geographical properties)	Fully decentralized P2P network
Edgeweights	Mostly static (bandwidth of links)	Highly dynamic (fees, balance)
Topology / Information propagation	Network can react to change in topology (e.g. BGB)	Gossip is to slow / noisy to propagate all relevant information (also Privacy!)
DoS - Attacks		
Pathfinding		

Spamming HTLCs is almost impossible to detect

	IP - Routing	**Onion Routing**
Type	Best effort	Source based
Data format	Open headers	Encrypted headers
Sender / Recipient	Known to routing nodes	Unknown to Routing nodes
Address Space	Logical - hierarchical overlay network (reflecting geographical properties)	Fully decentralized P2P network
Edgeweights	Mostly static (bandwidth of links)	Highly dynamic (fees, balance)
Topology / Information propagation	Network can react to change in topology (e.g. BGB)	Gossip is to slow / noisy to propagate all relevant information (also Privacy!)
DoS - Attacks	Via spoofing can be mitigated by ISPs	Anyone can send / delay onions. Impossible to mitigate!
Pathfinding		

Very different model / approach for pathfinding

	IP - Routing	**Onion Routing**
Type	Best effort	Source based
Data format	Open headers	Encrypted headers
Sender / Recipient	Known to routing nodes	Unknown to Routing nodes
Address Space	Logical - hierarchical overlay network (reflecting geographical properties)	Fully decentralized P2P network
Edgeweights	Mostly static (bandwidth of links)	Highly dynamic (fees, balance)
Topology / Information propagation	Network can react to change in topology (e.g. BGB)	Gossip is to slow / noisy to propagate all relevant information (also Privacy!)
DoS - Attacks	Via spoofing can be mitigated by ISPs	Anyone can send / delay onions Impossible to mitigate!
Pathfinding	Collaboratively by the Network	Achieved by sender or 3rd party service

Why is pathfinding hard on the Lightning Network?

(no BOLT)

Key ingredient for routing is to know a path

- In an ideal world pathfinding is easy
  - We create a graph
    - Nodes are lightning network nodes
    - Edges are payment channels
    - Weights are the routing fees
    - There is a label for edges called the balance
  - Finding a path is just as simple as calculating Dijkstra Algorithm on the network
    - The balance labels should be respected (constrained Dijkstra)
- After computing the constrained Dijkstra Algorithm
  - Either a path is found
  - Or we know that no path with enough balance exists

Reality Check! Weights are dynamic

- In an ideal world pathfinding is easy
  - We create a graph
    - Nodes are lightning network nodes
    - Edges are payment channels
    - Weights ~~are the routing fees~~ change with payment amount and over time
    - There is a label for edges called the balance
  - Finding a path is just as simple as calculating Dijkstra Algorithm on the network
    - The balance labels should be respected (constrained Dijkstra)
- After computing the constrained Dijkstra Algorithm
  - Either a path is found
  - Or we know that no path with enough balance exists

Reality Check! Balances are unknown...

- In an ideal world pathfinding is easy
  - We create a graph
    - Nodes are lightning network nodes
    - Edges are payment channels
    - Weights ~~are the routing fees~~ change with payment amount and over time
    - There is ~~a~~ no label for edges called the balance, a private and highly dynamic property
  - Finding a path is just as simple as calculating Dijkstra Algorithm on the network
    - The balance labels should be respected (constrained Dijkstra)
- After computing the constrained Dijkstra Algorithm
  - Either a path is found
  - Or we know that no path with enough balance exists

Reality Check! Shortest paths might just fail

- In an ideal world pathfinding is easy
  - We create a graph
    - Nodes are lightning network nodes
    - Edges are payment channels
    - Weights ~~are the routing fees~~ change with payment amount and over time
    - There is ~~a~~ no label for edges called the balance, a private and highly dynamic property
  - Finding a path is just as simple as calculating Dijkstra Algorithm on the network
    - The balance labels ~~should~~ can't be respected (constrained Dijkstra)
- After computing the constrained Dijkstra Algorithm
  - Either a path is found
  - Or we know that no path with enough balance exists

Reality Check! Topology changes during pathfinding

- In an ideal world pathfinding is easy
  - We create a graph
    - Nodes are lightning network nodes
    - Edges are payment channels
    - Weights ~~are the routing fees~~ change with payment amount and over time
    - There is ~~a~~ no label for edges called the balance, a private and highly dynamic property
  - Finding a path is just as simple as calculating Dijkstra Algorithm on the network
    - The balance labels ~~should~~ can't be respected (constrained Dijkstra)
- After computing the ~~constrained~~ Dijkstra Algorithm
  - Either a path is found
  - Or we ~~don't~~ know that no path with enough balance exists

Reality Check! We just make educated(?) guesses

- In an ideal world pathfinding is easy
  - We create a graph
    - Nodes are lightning network nodes
    - Edges are payment channels
    - Weights ~~are the routing fees~~ change with payment amount and over time
    - There is ~~a~~ no label for edges called the balance, a private and highly dynamic property
  - Finding a path is just as simple as calculating Dijkstra Algorithm on the network
    - The balance labels ~~should~~ can't be respected (constrained Dijkstra)
- After computing the ~~constrained~~ Dijkstra Algorithm
  - Either a path is found
  - Or we ~~don't~~ know that no path with enough balance exists
- Probe for paths until we find one!

Reality Check! Topology changes are unknown

- In an ideal world pathfinding is easy
  - We create a graph
    - Nodes are lightning network nodes
    - Edges are payment channels
    - Weights ~~are the routing fees~~ change with payment amount and over time
    - There is ~~a~~ no label for edges called the balance, a private and highly dynamic property
  - Finding a path is just as simple as calculating Dijkstra Algorithm on the network
    - The balance labels ~~should~~ can't be respected (constrained Dijkstra)
- After computing the ~~constrained~~ Dijkstra Algorithm
  - Either a path is found
  - Or we ~~don't~~ know that no path with enough balance exists
- Probe for paths until we find one!
  - Even when not finding one we don't know that no path existed.
    - The network might have changed while probing and a tested one could now work

Reality Check! Dynamic problem with uncertainty

- In an ideal world pathfinding is easy
  - We create a graph
    - Nodes are lightning network nodes
    - Edges are payment channels
    - Weights ~~are the routing fees~~ change with payment amount and over time
    - There is ~~a~~ no label for edges called the balance, a private and highly dynamic property
  - Finding a path is just as simple as calculating Dijkstra Algorithm on the network
    - The balance labels ~~should~~ can't be respected (constrained Dijkstra)
- After computing the ~~constrained~~ Dijkstra Algorithm
  - Either a path is found
  - Or we ~~don't~~ know that no path with enough balance exists
- Probe for paths until we find one!
  - Even when not finding one we don't know that no path existed.
    - The network might have changed while probing and a tested one could now work
- Highly dynamic problem with a lot of uncertainty

The Gossip Protocol (What do we know about the network to find paths?)

(BOLT 07)

Purpose of the Gossip Protocol

- Routing is source based
- Nodes need to be aware of the topology of the lightning network
    - Which nodes exist?
        - How to connect to them?
        - How to open channels with them?
        - What are channel policies?
    - Which channels exist?
        - What are the policies of the channels for routing?
            - Fees
            - Htlc_minimum_msat
            - Cltv_expiry_delta
            - Channel flags
        - Capacity
- Share information between peers

4 announcement messages

- Announcment_signatures
  - Needed to negotiate if the channel shall become public
- Channel_announcement
  - Makes a channel publicly known to the network
  - Proofs ownership of the channel
- Node_announcement
  - Shares meta data of a node (like its IP address)
  - Only possible if at least one channel was announced
- Channel_update
  - Updates channel policies
  - Not yet relevant to the capacity or owners

The announcement_signatures message

- Needed to negotiate if the channel shall become public
- Type: 259
- Data:
    - 32: channel_id
    - 8: short_channel_id
    - 64: node_signature
    - 64: bitcoin_signature
- The signatures are used / exchanged to construct a channel_announcement message
    - This ensures that a channel is only announced if both parties agree
    - If shared they link (!!!) the onchain funds and addresses to the node_id

The channel_announcement message

- Makes a channel publically known to the network
- Proves ownership of the channel
    - Proving funding tx pays to bitcoin_key_1 and bitcoin_key_2
        - Verify that P2WSH funding tx output is of this form
            - 2 <pubkey1><pubkey2> 2 OP_CHECKMULTISIG
        - pubkey1 is nummerically lesser of the DER-encoded funing_pubkeys
    - Proving that node1 owns bitcoin_key_1
    - Proving that node2 owns bitcoin_key_2
- Bitcoin_key ownership is explicitly done with signatures
- Node_signatures verify that the nodes agree on the proofs and message

The channel_announcment message format

- Type: 256
- Data
  - 64: node_sig_1 (commiting to SHA-256(SHA-256(message[256:])))
  - 64: node_sig_2 (commiting to SHA-256(SHA-256(message[256:])))
  - 64: bitcoin_sig_1
  - 64: bitcoin_sig_2
  - 2: len
  - len: features
    - According to BOLT 09 a mean for backwards compatibility
  - 32: chain_hash
    - 6fe28c0ab6f1b372c1a6a246ae63f74f931e8365e15a089c68d6190000000000 (BTC)
  - 8: short_channel_id
    - Must confirm to the funding tx
  - 33: node_id_1
  - 33: node_id_2
  - 33: bitcoin_key_1
  - 33: bitcoin_key_2

The node_announcement message

- Shares meta data of a node (like its IP address)
  - Can be spoofed
  - Ipv4 & IPv6 are both supported
  - Can be a tor onion
- Only possible if at least one channel was announced
- Type: 257
- Data
  - 64: signature
  - 2: flen
  - flen: features
  - 4: node_id
  - 33: node_id
  - 3: rgb_color
  - 32: alias
  - 2: addrlen
  - addrlen: addresses

The channel_update message

- Defines channel policies
  - Mainly for routing (meaning forwarding of htlcs)
- Type: 258
- Data:
  - 64: signature
  - 32: chain_hash
  - 8: short_channel_id
  - 4: timestamp
    - Used for others to decide if the channel can be updated
  - 1: message_flags (option_channel_htlc_max)
  - 1: channel_flags (includes direction, availability of the channel)
  - 2: cltv_expiry_delta
  - 8: htcl_minimum_msat
  - 4: fee_base_msat
  - 4: fee_proportional_millionths
  - 8: htlc_maximum_msat (static option not designed to leak balance)

Deleting channels from the network view

- Attempts to close a channel are not broadcasted via gossip
- Also successful closing is not broadcasted
    - Can be seen on the bitcoin network by seeing a spending of the funding tx
- nodes might still try to route htlcs if the mutual shutdown was initiated
- Channel lifetime can be seen on the blockchain

Querying the gossip protocol for information

- Nodes can retrieve old gossip messages from peers
- Query_schort_channel_ids / reply_short_channel_ids_end
  - Ask for channel_announcment and channel_update messages for a specific channel
  - Usually because seen channel_update before a channel_announcement
  - Or because unknow short_channel_ids from reply_channel_range
- Query_channel_range / reply_channel_range
  - Asks for short channel ids from a starting block up to a certain hight
- Gossip_timestamp_filter
  - Only channel_updates have a timestamp
  - Sends channel_announcements that correspond to channel_updates
- Initial Sync
  - Requested in the init message via feature flag
  - Gossip_queries was not part of the initial protocol so initial_routing_sync must be supported
- Rebroadcasting happens on a basis of newer timestamps
- Pruning the network by monitoring spends of the funding transactions

Summary of topology information for pathfinding

- Channel
  - 1: message_flags (option_channel_htlc_max)
  - **1: channel_flags (includes direction, availability of the channel)**
  - **2: cltv_expiry_delta**
  - **8: htcl_minimum_msat**
  - **8: htlc_maximum_msat**
  - Short_channel_id
  - **Fees**
    - **Feerate**
    - **Base fee**
- Node
  - Features
  - Node_id
  - Internet address

Path finding on the Lightning Network

(no BOLT)

Strategies for path finding

1. Probing cheapest paths
2. AMP (Atomic multipath) Routing - split payments to smaller pieces
3. JIT (Just in time) Routing - bring the best effort concept to lightning
4. Trampoline payments (sorry don't know hope to learn here)
5. Use previous knowledge
    a. For example reliability of nodes on previous routing attampts

Considerations when probing for paths

- One has to compute shortest paths from the destination to the sender
- Reason
  - Fee_per_hop = amt_to_forward * feerate + base_fee
  - Fees of later hops have to be respected in the fee rate
- Needs an adapted dynamic version of Dijkstra
  - Dynamic as actual weights change with every hop as the the amount in the onion changed
  - Adapted since we need to probe k paths in case the first k-1 paths fail
    - https://en.wikipedia.org/wiki/K_shortest_path_routing
    - https://en.wikipedia.org/wiki/Yen%27s_algorithm
- Runtime
  - O(kN(M+N log N))
  - Pretty inefficient or hard to compute on a large scale lightning network on mobile devices
- If all nodes behave well probing a single path should be quick (~ 1 second) independently of network size
- If nodes miss behave probing can take up to cltv_expiry / cltv_delta (~days)

Considerations with AMP

- Idea:
  - Known fact that larger payments have higher likelihood to fail
  - Split one large payment to several smaller payments
- Disadvantages
  - Routing time increases
    - Time = max{time for each path}
  - Probing still is necessary for every path
  - Base fee is paid several times
    - Side note: Do a lunch discussion about the fee model in lightning
  - Not even clear if likelihood for successful payment is higher (details on next slide)
  - More HTLCs in flight
    - Statistically more space wasted on base layer if channels break
    - Channels might become overloaded (remember routing time increases too!)
  - HTLCs are smaller (maybe even below dust?)
- Improvements
  - I refer to the talk by Alex (:

Arbitrary chosen probabilities of AMP success.

- No cherry picking involved
- Let P be the probability of success for large payment
  - Usually P is low (e.g.: 20%)
- Q = probability of success for small payment
  - Usually Q is high (e.g.: 80%)
- AMP success probability ~ $Q^{\#paths}$
- Is $Q^{\#paths} > P$?

Better simulation with real data might help.

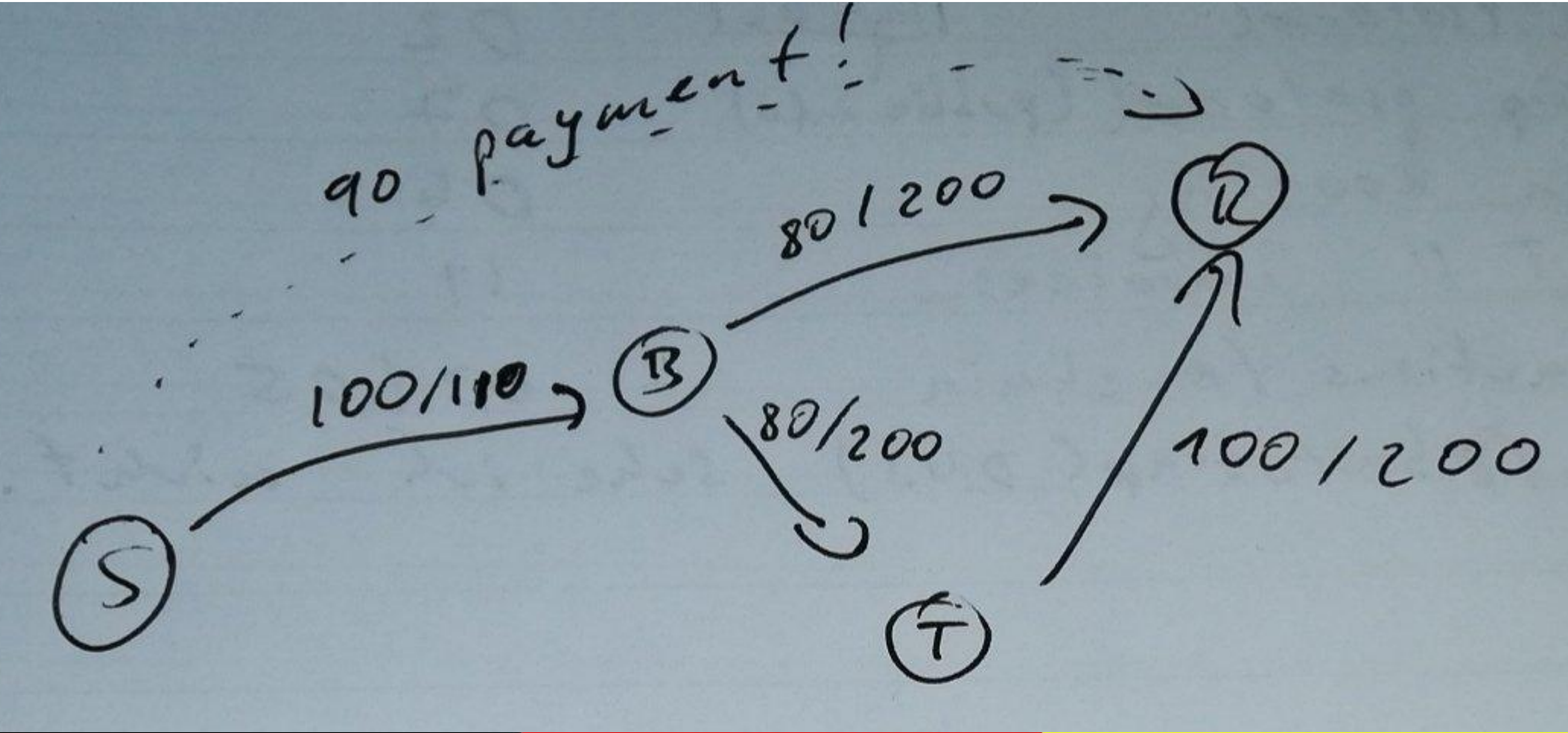
Arbitrary chosen probabilities of AMP success.

- No cherry picking involved
- Let P be the probability of success for large payment
  - Usually P is low (e.g.: 20%)
- Q = probability of success for small payment
  - Usually Q is high (e.g.: 80%)
- AMP success probability ~ $Q^{\#paths}$
- Is $Q^{\#paths} > P$?

Better simulation with real data might help.

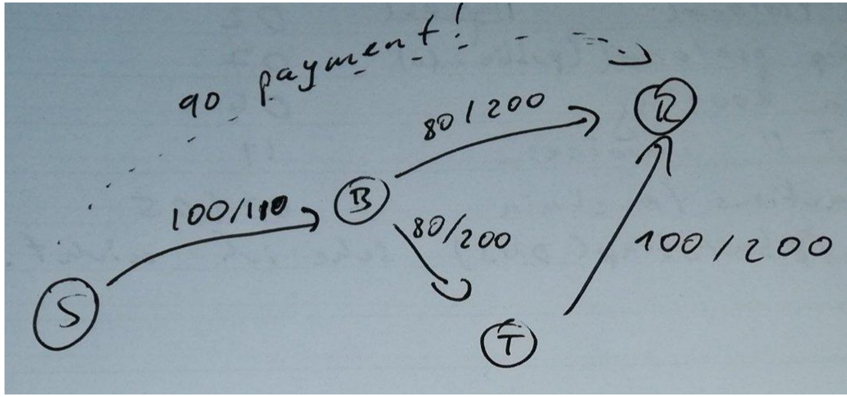
number of paths	P [success large]	Q [success small]	AMP [success]
n	20	80	
1	0.2	0.2	0.20
2	0.2	0.4	🟥 0.16
3	0.2	0.6	🟩 0.22
4	0.2	0.8	🟩 0.41
5	0.2	0.8	🟩 0.33
6	0.2	0.8	🟩 0.26
7	0.2	0.8	🟩 0.21
8	0.2	0.8	🟥 0.17
9	0.2	0.8	🟥 0.13
10	0.2	0.8	🟥 0.11

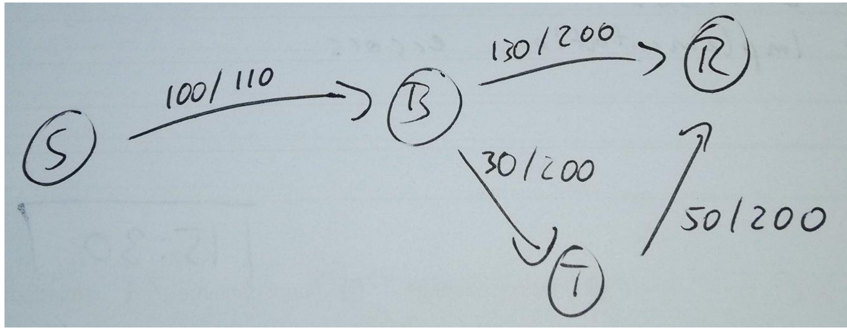
Short Intro to Just in Time Routing



Short Intro to Just in Time Routing

- B decides to rebalance 50 sat via
  - B → T → R → B
- Now B forwards the payment
- It simulates the AMP behaviour
  - 40 sat via S → B → R
  - 50 sat via S → B → T → R
- S doesn't need to make the decision
- Peers might
  - Share information (increase rebalance success)
  - Omit fees on the rebalancing route
- Rebalance in the local FOAF network!
  - All cycles of length 5 are contained there
- Background:
  - Emerged: https://twitter.com/renepickhardt/status/1102481472643129344
  - Details: https://lists.linuxfoundation.org/pipermail/lightning-dev/2019-March/001891.html





Considerations with JIT

- Idea:
  - Source based paths might fail due to insufficient channel balance
  - Interrupt routing (Defer to future - a strong concept!) and rebalance the channel as needed
    - Potentially with the same payment hash (only rebalance if routing is successful!)
  - No direct protocol change needed!
- Disadvantages
  - Needs additional probing for rebalancing operation
  - Maybe costly for routing nodes
    - Cost of rebalancing could be higher than the routing fee for the payment
  - CLTV deta of channel might not be sufficient to do a rebalancing operation in case of timeout
  - Longer routing time due to rebalancing
- Improvements
  - FOAF overlay / rebalancing network (extend gossip?)
    - Similar to BGB nodes could share some (aggregated?) information about their channels
  - Cost free circular onions
    - Could even be without onions as a local circle might have common interest to rebalance

(Multihop) Trampoline payments

- High level Idea:
  - Get the best effort component in
  - Define a rough route (multitrampoline)
  - Let the nodes on the rough node figure out how to do actual paths between them
- Disadvantages
  - Needs Protocol changes
    - The onion format needs to be adapted
  - A clique of Trampoline routing nodes might emerge and build a kartell
    - Centralization or hub / spoke topology might emerge
  - Privacy model decreased
  - Don't work for some of the spontaneos payment suggestions
- Improvements
  - I refer to the talk by Christian (:

General thoughts on pathfinding on Lightning

- We should acknowledge that it is an open / difficult problem
- Various proposals / ideas exist
  - Some require protocol changes so we might have to be quick to get them in
- Different strategies could be combined
- There seems to be a tradeoff between
  - Privacy of individual nodes
  - The ability of the network to solve the path finding problem
- Lightning might want to copy some ideas from Autonomous systems
  - In particular to have some form of hierarchical address space
- Personal opinion:
  - Path finding services will emerge as a business model / opportunity
  - Once you offer that service you easily might have more information than other node
  - You can offer better paths
  - Similar to google who does not only have a web crawl but also the search query log files

Path finding has many open challenges!

(Join us and help to tinker about them!)

References and helpful links

- https://github.com/lightningnetwork/lightning-rfc
- https://www.youtube.com/user/RenePickhardt
- https://bitcoin.stackexchange.com/questions/tagged/lightning-network
- https://lightning.network/lightning-network-paper.pdf
- JIT https://lists.linuxfoundation.org/pipermail/lightning-dev/2019-March/001891.html
- AMP https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html
- Base AMP https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-November/001581.html
- Trampoline: https://lists.linuxfoundation.org/pipermail/lightning-dev/2019-April/001956.html

About this slide deck

The purpose is to help spreading education about the Lightning Network Protocol so that the technology will be adopted more quickly by more people. This shall be my contribution to the Bitcoin / Lightning Network Community.

This slide deck was presented during Chaincodelabs Lightning Residency program in June 2019. It is part of https://commons.wikimedia.org/wiki/File:Introduction_to_the_Lightning_Network_Protocol_and_the_Basics_of_Lightning_Technology_(BOLT_aka_Lightning-rfc).pdf. To the best of my knowledge the original file is the most comprehensive work making an introduction to the BOLT standard.

The slides are part of my effort to create a book about the lightning network. You can follow that effort at: https://github.com/renepickhardt/The-Lightning-Network-Book or you can support the effort at my fundrasing pages at: https://tallyco.in/s/lnbook or at: https://www.patreon.com/renepickhardt or at 1GZx8tWgDd21Rd8b1QdMrzdZGHgyfVkzaD part of this effort also consists of creating video tutorials and teaching materials on my Youtube Channel over at: https://www.youtube.com/user/RenePickhardt

This work was funded (sorted by amount of contribution from top to bottom) by: Me personally, fulmo.org, everyone who contributed to the above mentioned fundraiser and George Danzer.

Thank you to the lightning developers and people in various telegram groups for helpful discussions