Utreexo and Lightning

Agenda

- Why Utreexo is useful for Lightning
- A 10,000-feet view of Utreexo
- How could we integrate Utreexo in Lightning
  - The correct way
  - The permissionless way

Who am I

- BOSS @ Vinteum
- Maintainer @ Floresta and rustreexo
- PGP: **8996 4EC3 AB22 B2E3**
- GitHub: Davidson-Souza

Why would we integrate it with Lightning?

- Lightning nodes are required to fetch UTXOs to confirm that channel outpoints actually exists
- But not every Lightning node has access to a full node, so it delegates this query to a centralized entity, such as an Electrum Server
    - Thus reducing their privacy and security – The third party service could lie about those UTXOs' existence
- Running a node with a full UTXO set isn't always viable
- But Utreexo clients, are

Why would we integrate it with Lightning?

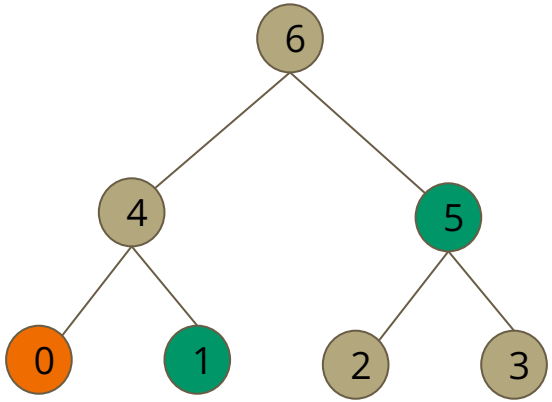
- Floresta is a client that uses ~300 MB of disk and ~200MB of RAM
- We’ve successfully ran it on an old Android phone
- Performs all operations locally, better privacy
- Better security model, even on phones
- Depends on CBF to provide Lightning informations

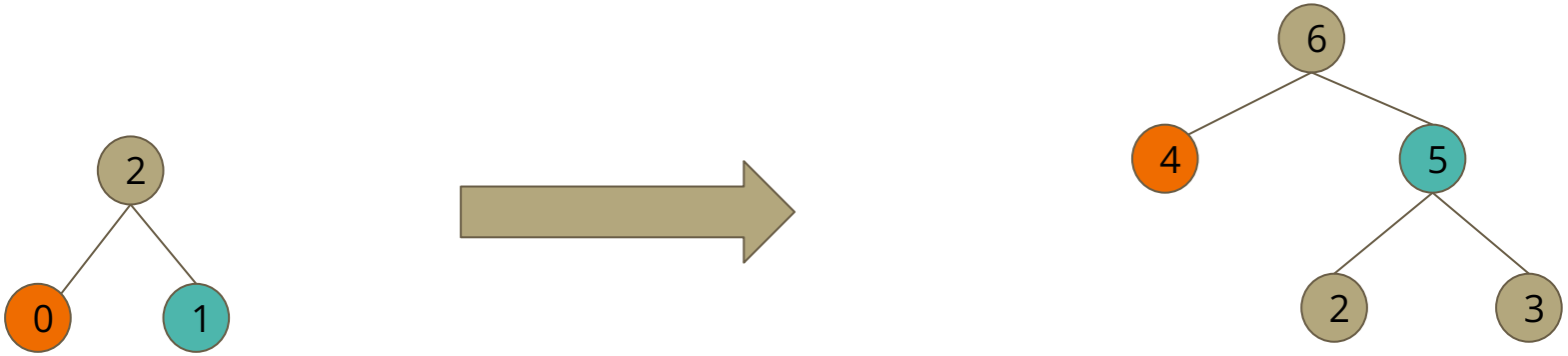
Utreexo

- Utreexo is a dynamic accumulator for the UTXO set
- It allows for super small, fully validating clients
- The whole UTXO set is compressed to less than one kilobyte
- You then have to download proofs for each input in a block
- Trading off some Bandwidth and CPU for storage and IO

A primer on Utreexo

- Utreexo uses a forest of Merkle Trees
- To prove something, you just reveal the hashes needed to recompute the root of one tree
- It was optimized to allow efficient keeping and updating of multiple proofs
- Updating the accumulator also updates all proofs.


6
4
5
0
1
2
3


2
0
1
6
4
5
2
3

A primer on Utreexo

- In Utreexo, you may keep: all of the leaves, some of the leaves or no leaves at all
- If you chose to keep all leaves, that will require more disk, but allows you to prove every single UTXO in existence
- On the other hand, having no leaves won't let you prove anything but you can still validate
- As a middle ground, you can hold only leaves that you are interested in, like your own UTXOs.

A primer on Utreexo

- This is possible because the addition and deletion algorithms in Utreexo only have these inputs:
  - How many leaves there are
  - The current roots of the accumulator
  - A proof for what's being deleted
  - A list of new things being added

Why not Compact Block Filters?

- Compact Block Filters are great for wallet sync, but has some downsides when it comes to Lightning
  - Constant rescans may be hard on the battery and bandwidth
  - Slow and high-latency connections may hurt UX
  - More data to store

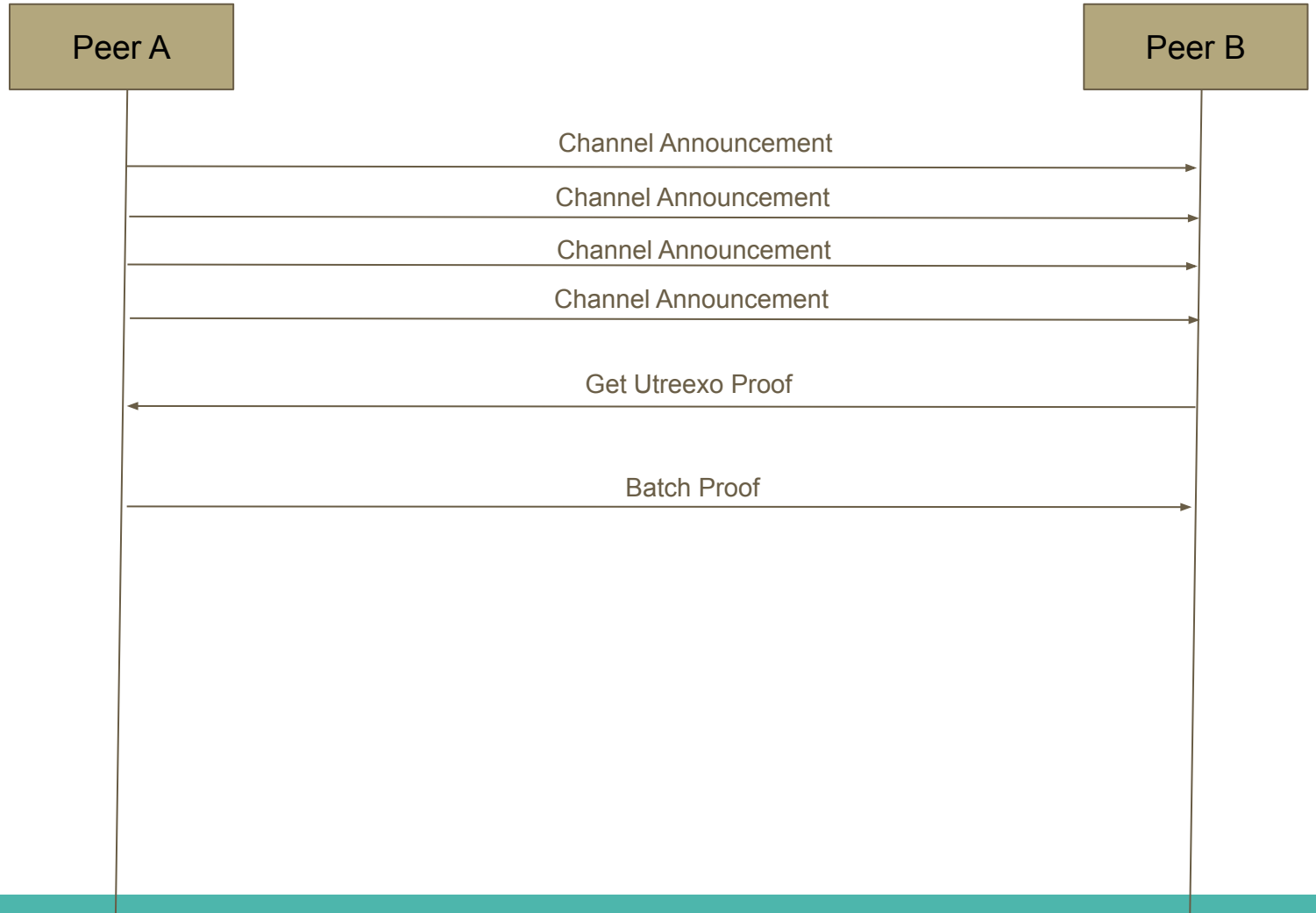
How could we accomplish that?

- The Correct Way
  - Requires changes to the P2P protocol, as well as all clients
  - Trustless

- The Permissionless Way
  - Don’t require any change to the lightning P2P protocol
  - Semi-trusted
  - Don’t need support from nodes that won’t talk utreexo

The Correct Way

- For each channel, nodes should keep a utreexo proof for each channel outpoint
  - They will download this proof once, probably the first time they see this channel, then hold on to it until the channel is deemed as closed or inoperative.
- Upon learning about a new channel, nodes SHALL request a proof for this channel
  - Nodes MAY request the proof to the same peer that sent that channel to them
  - Proofs MAY be batched, to reduce bandwidth usage
  - Upon receiving a get proof request, nodes SHOULD use their own accumulator to prove those UTXOs
  - If the proof is invalid:
    - nodes SHOULD NOT mark the channel as invalid – proofs are third-party malleable, and may be intentionally made invalid.
    - The receiving node SHOULD ban this peer and drop the channel announcement for now


Peer A
Peer B
Channel Announcement
Channel Announcement
Channel Announcement
Channel Announcement
Get Utreexo Proof
Batch Proof

Compatibility with zk provers

- There are some experiments that allow verifying Utreexo proofs in a zk prover
  - though the proving times would still be an inpeditive in most case
- Interoperability with aut-ct is possible as well

The Permissionless Way

- Could be deployed today, without modifying other clients
- Uses some kind of express sync to get old channels and their proofs
- Keep a cache of the latest UTXOs that may be a Lightning Channel (p2tr or p2wsh outputs)
- When receiving a new channel announcement, just check the local cache for that UTXO
- This fixes one of the trusting sides of express sync, all UTXOs must exist onchain

The Permissionless Way

- A client implementing this wouldn't need support from their peers, would be completely local
- Your node will keep only the required UTXOs on cache, and you can check whether they still exist at any time
- The downside, obviously, it's centralized and requires **some** trust in the express sync server

Conclusions

- Utreexo could vastly improve the UX for self-custodial mobile Lightning wallets
- The changes to the protocol aren't that intrusive
- Most of the "Utreexo stuff" would live inside the Bitcoin node, rather than the Lighting one
- We already have libraries in Golang and Rust. Other libraries could either be carved-out as binding to *rustreexo*, or implemented from scratch. The required functionalities are simple to implement

Thank You

References

https://eprint.iacr.org/2019/611.pdf

https://github.com/vinteumorg/Floresta

https://github.com/halseth/output-zero

https://github.com/AdamISZ/aut-ct