What's up with Package Relay and V3 Commitments Anyway?

- Iterative package relay in bitcoind 28
    - Send the child
    - Node realizes doesn't have parent, then fetches using the existing p2p messages
    - 1 child - 1 parent
        - Try with CPFP, then keep RBF'ing that from there
    - No change in format, no extra messages or anything
    - Nodes will now try optimistically
    - Wallet level integration
        - Submitpackage (RPC)
            - 1p1c
            - Don't try anything batch related
        - 1 sat/vb floor for the parent
            - Commitment txns can be 1 sat/vb
        - Mobile wallets kk
            - Give child
            - getdata for the parent
            - Then need to respond to that
        - Electrum needs updates here as well
            - Short term would have some fork w/ the new APIs
            - Thinking also about a more LN friendly way to sync to the chain
                - Current versions don't handle re-rgs well
            - None of them (esplora, etc) don't support package relay yet
    - **Wallet Integration Guide:**
        - https://docs.google.com/document/d/1Topu844KUUnrBED4VaJE0lVnk9_mV6UZSz456slMO8k/edit
    - Package RBF
        - If package of size 2, will attempt to do package RBF against conflicting transaction sets
        - Ability to use 1p1c to update the fee rate of conflicting packages
            - My commitment txn not in mempool
            - Remote one is more widely propagated
            - I can send the CPFP and have my commitment txn accepted with the CPFP bump
- TRUC
    - Opt in topology restriction
        - Parent max size: 10k vb
        - Child max size: 1k vb
        - New max HTLCs: 120 something
    - Splicing edge case

        - Splice out
        - Paying someone on chain
        - Don't have another output
        - You unilaterally close
        - Spend the spice, commitment txn, and anchor output spend
            - Depth of 3
            - Not covered/allowed rn
        - Stop gap
            - Anchor output with the splice out
            - Then can do both, use the one with a higher enough fee
        - Or
            - Just spend from the OG output, ignore the splice
    - Sibling eviction
        - Can take the old format of commitment txns (everything unlocked)
        - Then have your anchor, any child txn can evict another child txn
        - If limits exceeded, then can try to evict another child
            - Basically can ensure that some child can't bog everything down
            - Pinning protection across outputs
    - Cluster mempool
        - ??
    - Can remove CSV 1 from all the scripts
    - Alternate anchor output handling
        - Still have two, split the dust amt evenly between them?
- Pay to anchor (P2A)
    - New standard pk script
        - New segwit version
        - In 28.0
        - segwit output v1 with special push
            - 2 byte data push
        - Stands out, but force closing anyway
    - No signature data
        - Segiwt input, can't put sig script data
        - No witness data is allowed
    - Can allow 3rd party to replacement cycle you if it's keyless
        - Maybe for n=2 don't want
        - But if it's a large multi-party channel, then you don't care about keys
    - If it's keyed
        - What do we put in the ephemeral anchor output?
        - Need to put the funds somewhere
        - T-bast's version today
            - Put the amt in the ephemeral anchor
            - So total dust HTLC amt in the anchor
        - Should also have a max dust exposure

                - Otherwise can send out dust HTLCs and then try to sweep them w/ a force close
                - But it's also a race condition
- Replacement cycling protection
    - Anti-cycle repo: https://github.com/instagibbs/anticycle
        - Client side protection against pinning stuff
        - Uses zmq etc to detect spurious state transitions
- Cluster mempool
    - 29??
    - Would be everything
    - May end up also overhauling fee estimation along the way
    - Generalized package relay after
        - https://github.com/bitcoin/bips/blob/master/bip-0331.mediawiki
    - Will there be policy changes
        - Inclusion
        - relay
        - Eviction
        - Mining
    - Will the definition of v3 change?
        - There'll be a v4
        - Or it won't be super fundamentally different
- Ephemeral dust
    - Allowed to have up to 1 dust output, zero fee
    - If spend from a txn that has dust, the you must spend that output
        - P2A - 239 sats or less
- Zero fee commitment experiments
    - Main goal is to get rid of update_fee
    - Still care about fees a bit, in terms of figuring out if HTLCs are dust
    - Design questions
        - Do we go keyed or unkeyed?
        - Do all the HTLC dust amts go into the new anchor output?
        - New commitment type and new channel type?
            - Only current commitment or also without taproot channels?
            - Can use dynamic commitments to upgrade both
            - Long term, we push towards 2, but then acknowledge 3 in the short term
    - Current version
        - Zero fee
        - Removed update fee
        - Ephemeral anchors
        - HLTC dust to anchor outputs
    - Wallet interaction
        - What do we do for wallets?
        - Want them to have some funds left over

            - Some ppl already having reserved funds for fee bumping
        - Block working on some research, focused on the LSP
            - How many UTXOs do you need, what size of the UTXOs, etc
            - Can also do a splice in where you're just adding fees
- Interaction with v3
    - If using v3 for everything, then maybe some services get mad
        - As they rely on spending multiple levels of unconfirmed change
    - Is it safe to just use v3 everywhere?
        - Some other interactions, forcing ppl to use v3 when spending outputs
    - Splicing
        - If sending on chain, then receiver needs to use v3

PTLCs & Simplified Commitments

- P2p messaging changes
    - https://gist.github.com/instagibbs/1d02d0251640c250ceea1c66665ec163
- Single sig adaptors
    - Musig2 adapters
- Question of if single sig or musig2 adapter
    - Would need nusig2 to make sure the adapter is secure?
- nonces preshared with simplified commitment
    - Each time you send the 'yield' message, then you send the extra set of nonces again
    - Either way sharing the PTLC (public point) along with each of the messages
- Simplified commitment motivation
    - https://github.com/lightning/bolts/pull/867/files
    - Can't predict the other commitment, also need the extra set of nonces for each commitment update
    - Also able to NACK an HTLC, if you send a nonce while you ACK it
- stfu ends up being gained implicitly
    - One side going at a time, after a yield, you have the floor
- Simplex
    - Two implementations
        - Can send a yield_please message, then send
            - Potentially higher latency
        - Can send a yield please, and start adding stuff, then you ignore the stuff once
    - May also simplify reconnection somewhat
    - RTS (request to send)
- Merging PTLCs
    - Can get 2 PTLCs in, send 1 out

        - Able to upgrade the PTLCs
- Implementation exists, but no BIP
- Ordering
    - Lower pubkey starts
- Optimistic version
    - You can send updates if its not your turn, they then decide ti send `yield`
- Simpler version
    - You can't send updates if it isn't your turn
    - RTS + CTS
- More links
    - Old links I should have included in the writeup: https://github.com/BlockstreamResearch/scriptless-scripts/pull/30 <---single sig adaptor for locks
    - musig2 variant: https://github.com/BlockstreamResearch/scriptless-scripts/blob/master/md/multi-hop-locks.md
    -
    - Arik writeup from long ago: https://github.com/arik-so/bolts/blob/ptlc-bolt-draft/xx-point-timelock-contracts-DRAFT.md#offered-ptlcs
    -
    - Message reordering to save RTT from tbast: https://github.com/t-bast/lightning-docs/blob/master/taproot-updates.md#point-time-locked-contracts

Getting Fancy Off-Chain: Super Scalar, Channel Factories & Friends

- Diff ways to scale UTXOs
    - Channel factories
    - Timeout trees
    - Ark
    - Clark
        - Assumes out of round payments
- Super scalar 1j
    - Components
        - Decker wattenhofer
            - Supports N Parties
            - Has a set amount of state changes, based on decrementing lock times
            - Closing

        - Multiple transactions on chain
        - Variable waiting times, not just a single update, the timeout varies depending on how much states you've actually used
    - Updates
        - Decrement the nSequence when you want to add a new state
        - Ensure that only the latest state is published
        - Eventually the counter goes down to zero
        - If you need another state change, you go one level up, and hten decrement that instead
- Timeout trees
    - At each node, you have a CLTV branch that allows the LSPs to grab teh funds and run
    - The timeout is the interval that everyone needs to act, or exit somehow
    - Either need to exit, but sending the funds to a diff channel or a diff timeout tree, or swap to on chain, etc
    - Each node: CTV && CLV
        - If not CTV, then you use pre-signed transactions instead
- Laddering
    - If you have a contract that locks your funds, to get more funds later on (you never touch them there)
    - Instead of putting all your money into a single contract, you ladder them, s.t they have the same total period and start with a different time (laddering)
    - Laddering gives you more flexibility in how you change your liquidity
    - Example
        - Have many 5 year GLCs
            - Then next year start another noe
            - Then 5 years from now, you can put back into another cert, or add more funds to it, etc
    - Timeout trees mean the LSPs lock some of their funds into the tree
        - To get more flexibility, the LSp has a chance to decide how to allocate the funds
        - You ladder multiple timeout trees, gives them a chance to re=allocate the set of funds
    - Reallocation
        - If you have a threshold, then can sign
        - Otherwise, wait for the timeout
    - UTXO wise
        - Basically independent instances of each of the trees
        - Kind of like Ark rounds in the future

- Combining it all
    - Combine DW and timeout trees
    - You run multiple instance of them
    - Tree structure
        - You have less number each child node
        - At the leaf you have two clients
        - If you want to change the state of the leaf, just LSP and two clients
            - A & B & C
            - Only two of them needs to be online
        - You can reallocate some of the funds in a leaf
            - Can allocate it to A or B
            - Can do so w/o all the clients online, only A or B needs to be online
        - Don't need everyone to come online and change the state or re-allocate between them
    - If you want a channel between two clients, they need to be one the same
    - What should the value of L be?
        - Exercise for the reader
    - Can go back up and wake up more clients to be able to further re-allocate funds
        - Less likely to succeed as you move further up in the tree
        - After so many layers, we just do plain timeout trees instead
        - As the prob of all being online starts to drop as you get higher
- routing/network awareness
    - Advertise as just one massive node
    - Each client has some random scid or just use blinded paths to route into it
- Exit
    - Just as bad as a timeout tree
    - Active period
        - Period where they should get out of the construction co-op close somehow
    - Dying period
        - Come online and get their funds out or move to a different tree
        - Basically a safety period, a sort of buffer, only after this do they stop allowing people to send funds into the tree, but only support outgoing payments
- Extensions
    - Can use PTLCs to swap out by revealing a secret to allow a party to take over a key in the tree
- Assuming OP_CAT
    - If some peers don't ever come online, then craft it s.t it can only sign once
    - Allows the LSP to spend the L output
        - But in doing so it binds itself to only one version of it
        - Uses a single signing script

        - Then requires a tree of bonds to surface funds to allow the binding
        - The bond can be a leaf itself, so allocate per user pair
    - Two shot adapter3 signatures
        - Sign twice, then you leak your key
        - https://eprint.iacr.org/2024/025

Lightning Talks

- Ability to recover from force closes
    - Give a key to the counterparty to allow them to spend theirs
    - Fast exit from a commitment transaction
    - Can give them your revocation base point secret
        - Allows them to exit early
    - A friendly thing to do, helps out the other party
    - Could be in the chan reest message
        - Usually ignore it, but can keep it around, and respond with the information

Fundamental Limits on payment delivery

Presented and discussed ongoing research about payment channel networks' ability to successfully deliver payments.

- A successfully conducted payment between two users affects the ability of (potentially) all other users to make payments.
- Integer linear programming can be used to decide if a (wealth) distribution of coins among the users of the Lightning Network is feasible given the current ChannelGraph topology
    - https://github.com/renepickhardt/Lightning-Network-Limitations/pull/2
- Deciding if a payment is feasible can be done without wondering about actual liquidity states in channels by studying the change in wealth distributions.
    - https://delvingbitcoin.org/t/estimating-likelihood-for-lightning-payments-to-be-in-feasible/973
    - Useful for simulations and estimating expected rate of infeasible payments via random walks
- Infeasible payments need at least 1 on chain transaction to be delivered
    - Thus the expected bandwidth of payments per second that are supported on the lightning network can be estimated by dividing bandwidth of on chain transactions per second by the expected rate of infeasible payments on the payment channel network

        - There must be no more than 0.0149% of all requested payments infeasible for the network to support 47 thousand payments per second like visa
- It seems that the rate of infeasible payments declines exponentially with more users joining the network
    - This is mitigated a bit by the coins that a new user brings to the network
    - But as a rule of thumb every user puts a burden to the network's ability to facilitate payments
- Introducing credit would mitigate these problems
    - Possible with taproot assets
- Channel depletion is expected to happen because of concentration of mass.
    - It is questionable how much of it can be mitigated through economic user incentives.
- Multiparty channels seem to be the best mitigation to the limits that two party channels put on payments feasibility
    - Simple argument: k*C/n is the average access to liquidity a user can have in a k-party channel network of n users with C coins.
- Concerns with the results:
    - Impact of the assumptions about distribution of wealth unclear
    - Random walk should be weighted as payments are not uniform across the participants
        - Weighting not clear / known though

Make Gossip Suck Less

Gossip strategies

- CLN
    - Sync from 5 diff peers
    - No rate limiting
    - Keeps a UTXO set cache of potential channel points
- Lnd
    - 3 sync pers
    - Rotate over time
    - Historical spot checking
- LDK
    - Don’t watch on chain closes
    - Grab everything all the time
- Eclair
    -

Adding SPV proofs:

- Should be added to SPV proofs
    - Related to a direction for utreexo proofs

Additions in gossip 2

- Adding fields for capacity and max HTLC
- Adding channel gone as well
    - Basically away to advertise disable on that setting

Minisektch limitations:

- Can only have a certain amt of set diff
    - Can tune the size of the sketch itself
    - Can maintain a big one then send a small one
    - Over time you can bisect things smaller and smaller
        - Ata certain point then you can bail out
- Can't represent the node ID, as it's larger
- 64-bit keys
    - Can pack in the scid itself
    - Can also add the timestamp as well
    - Few other bits you can
- Going for a global set per peer
    - Maybe need 3
        - Chan ann
        - Chan upd
        - Node ID
    - If keeping closed channels for 12 blocks
        - Then can mess up the filter set
        - As they'd never converge in the first place
- Impl plan
    - $?\geq_{mnbv}$Global
- Able to VCI add an exclusion list, things that you didn't have in the sketch

<u>Laolu spilled coffee on his laptop at this point</u>

Flexible encrypted comms

- Have comms key
    - Allows proxied connections
    - Otherwise, need to do mapping by ports
- Node ID is pretty overloaded at this point
    - We should start to add additional keys to the node ann
- Other variants
    - Possible noise extension that can allow a proxied version?
        - Able to send node want to connect to, after the actual handshake
        - Potentially can just be a **composite** handshake protocol?

Last Mile Mobile On Boarding

- Onboarding issues for self custodial users
    - Receivng small amts on boarding for the first time
    - Opening channels can be expensive for small paymet amts, 21 sats or w/e
    - Can also assume that first time bitcoin users can also be high turnover
    - If someone needs to open liquidity to all the users, then assuming high churn, you have a lot of locked up liquidity for all the users
        - Some mobile wallets introducing a fee credit system
        - Others tried e-cash, so would have ecash to start w/ then ideally upgraded later
            - Certain threshold amt then they would open a channel
            - Feedback from users was that they got confused, so they split up the balances instead, more confusing UX wise
            - Didn't fully play out though
            - Ecash mints in general have an uncertain future, will there be issues of rug pulls, etc in the future
    - Other difficulties with high fee spikes, can compromise UX, annoy users
        - If it's $20-30 to splice or open a channel, becomes economically feasible for balances of a few dollars, etc, etc
    - What's the realistic amt for a wallet user today? Maybe on the order of $300-500 users, factoring in all the potential closing costs
    - Many LN users today have existing Bitcion, so not as big of a deal to lose some or have some be locked up for a period of time, but for newer users can become a bigger issue
    - Channel reserve also can make UX gotachas
        - Many impls now support having this be ero
- Pending HTLCs and reserve
    - Unclear to users why they can't spend or recv a certain amt of money
        - General set up for payment channels, but with HTLCs can add additional edge cases
    - Fee spikes can render larger amts of the channel unusable
    - Zero fee commitments help, but then you may need additional UTXOs
- Fee rebates
    - Basically an upfront fee payment for the service
    - Not an actual balance, can only be used towards paying for opening a channel
    - UI will show what the current threshold is to open a new channel
        - dynamic, based on the current on chain fee levels
    - Current super set form of liquidity ads has basically everything needed to implement this, alongside JIT channels, etc, etc

        - Has provisions of other ways to pay the fee, can be done via HTLCs, or via the fee rebates, etc
    - Can also be used to pay on chain fees for splices, etc
- Stackernews
    - Slick UX, but mainly because it's a custodial wallet
- Off chain channel on boarding
    - Dynamic membership is difficult
        - Convince thatm that they have a stake in the platform
        - Is there a way of adding someone to an existing contract w/ interacting with the block chain
        - Basically k-of-n vs n-of-n
    - Can you adjust the amt of outputs via script?
        - If not, then we can't have a unilateral
- O(1) exit schemes
    - Assuming a set of covenants (CSFS, CAT, EC ADD+MUL), then can send a single message to a contract to exit with a set of funds
        - With the above, can reconstruct the taproot output public key given a set of leaf scripts
- Chain fees
    - Ultimately, you may not be able to pay for chain fees, non custodial, but you need a min amt of funds to be able to exit from the state
        - Chain fees
        - Drjya bath tub basically
    - Incentivize wise, edge cases re large amt of users, small balances, large amt of users
        - 1 sat each, 100 million users, can't get it all on chain
        - So then either *SP can steal it, or users stit there and get nothing
        - So then you want an ability for users to just be able to say: "fuck it", and burn all the funds
            - Then able to keep the *SP in check, as they can't gain anything
        - Burning requires some proof of the *SP trying to cheat in some regard
            - So then users can hold them accountable, ensure they don't try renegade on chain
        - Users always have the option to push the big red button, and blow the entire thing up
            - MAD
    - Fundamentally can't make chain fees affordable for all the parties
- Is there some impossibility result associated with the ability to the join off-chain?

BOLT 12: What's Next

- Other stuff

    - Invoice replacement
        - Start paying something, gets stuck, but want to fetch some other invoice instead
        - Receiver then says they'd never try to redeem the older invoice
        - If they do pay for w/e reason, then can show that they tried to do so?
        - Also have the ability to do cancellable HTLCs/PTLCs, so is this actually needed?
    - "Refund for"
        - Just a payment label, can say what the payment was a refund for?
    - Recurrence
        - Seconds, days, months or years
        - Also has a payment window, limit, and start period
        - Does this support streaming sats?
            - Payment period of 1 second make sense for some cases
            - Can continue to fetch invoices each time
            - Can add
        - Uses another layer of a hash chain to allows the receiver to not need to remember all the payment hashes
        - Rusty wanting to start to work on this reas next
            - Reformat the spec, rebase stuff, push it out there to see what's next
            - Have some ppl that want to implement on the wallet side
    - Issuer + DNS proofs
        - The reverse version of BIP 353 (user @ domain) instead that paying "blockstream.com"
        - Basically the ability to bind a node's pubkey to a domain name
    - Contact information
        - BOLT 12 contact stuff
            - Payer key required
    - Payment proofs
        - Encapsulates the invoices, include the preimage
        - Signed with the payer key
- Rate limiting and back pressure stuff
    - Levels of rate limiting
        - Free budget for those that don't actually have a channel
        - Initial global rate limiting based on some messages/s kb/s limit
        - Back pressure rate limiting
            - Discussed in the past
            - Some simulations shown that works ok
            - Do nodes need to know what the current limit is?
            - How far will the back pressure messages go?
            - Similar to TCP AIMD
                - Start to increase slowly

            - If they tell you you're sending too much, then divide w/e you have by two
        - Payment slots
            - Certain point things start to shut down, want to be able to send packets through still
    - Ideally rate limiting is somewhat proportional to the size of the channel one is sending over

Covenants!!!11

- Btcplusplus in May @ Austin
    - Talked a lot about script stuff
    - Something that was talked was adding covenants and other scripting items
    - In the past LN had talked about moving to eltoo + LN symmetry
        - Initial ideas required sighash_noinput, now anyprevout
        - Ended up evolving from there, had other proposals to give you similar functionality in Bitcoin
- Zooming out:
    - What are the options that exist that give you LN symmetry?
        - Nice cuz: minimizes risk of force closes, no penalties, easier back ups, multi-party channels, etc
    - Options
        - Anyprevout -> sig hash
            - Also gives the ability
            - Actually could've been possible with taproot, but the challenge hash included the pubkey, so pubkey recovery wasn't possible