NUCLEUS: Capital-efficient multipeer Lightning payment channels

Working draft, version 0.1.0 of 21 Aug 2023 (Helium)

GKawjD-cefXUrjP-Lr6h8oPt-wPj9vX44-SDhdnJFv-DHfPbx[1]

[1]atomic-mr-nuclear@onionmail.org

***Abstract*—Lightning payment channels are the only solution for blockchain scalability problem available for existing UTXO-based P2P networks which doesn't require consensus-breaking changes. Still, existing forms of payment channels are capital-inefficient and put high availability requirements on the participants, which reduces system utility and adoption. The current work proposes a new form of Lightning channels (named "Nucleus"), which can be created and run by multiple participants making their liquidity fully available inside a multipeer channel. This significantly increases capital efficiency, improves liveness and doesn't require the creation or participation in an atomic swap routing network (Lightning network). The solution can operate on top of any UTXO-based blockchain equipped with N-of-M threshold signature scheme(s) and timelocks, without introducing new consensus-level requirements of script operational codes.**

I. INTRODUCTION

Blockchains, which are distributed event log databases ("distributed ledgers"), do not scale well and are poor on privacy. While the latter problem is potentially addressable with zero-knowledge proof systems (like in Mimblewimble [7]), it appears that the former problem can't be addressed at the blockchain (layer 1) level without increasing trust assumptions, thus requiring a dedicated abstraction layer above.

One of the early and still promising directions for layer 2 constructions are *Lightning channels* providing the ability to trustlessly maintain state by a number of participants on top of UTXO-based blockchain(s). Multiple Lightning channel variants do exist today (Poon-Dryja [8], Decker-Wattenhofer [3], Decker-Russell-Osuntokun [2], Burchert-Decker-Wattenhofer [1]), however, all of them require participants to be permanently online and to provide excessive capital for getting inbound liquidity, which significantly reduces capital efficiency. If both conditions are not met the utility of Lightning channels as an offchain settlement layer significantly degrades; which leads to the fact that the only existing liquidity network made with lightning channels (*Lightning network*) falls more than an order of magnitude behind both onchain payments (by transactional volume) and the total capital locked for other types of layer 2 solutions (for instance, rollups has more than $2b of total capital locked with some leading protocols providing >$1b [4], comparing to $160m of capital locked in Lightning network [5], which got to production years before).

II. DESIGN

A. Basic construction

A Nucleus channel can be created by any number of *peers* (two and more), $\mathbf{P} := \{A, B, C, ...X\}, N := |\mathbf{P}|, N \geq 2$, and has a structure shown on the Figure 1.

The channel is opened by a *funding transaction*, containing single output (*funding output*) locking funds to an N-of-N multisig (such multisigs hereinafter we will denote as $\sum_{i=0}^{N} p_i$ with $p_i \in \mathbf{P}$). The funding transaction should be mined onchain as a prerequisite for channel security.

A state of a Nucleus channel includes outputs of the *allocation transaction*, always directly spending the funding output of the funding transaction, distributing it among the channel peers, and from zero to any number of *operation transactions*, spending allocation transaction outputs and, optionally outputs of other operation transactions.

Outputs of both allocation transaction and operation transactions are constructed in the same manner: they can be spent by either the output owner (one of the channel peers), but after a certain timelock tl, – or by the same peer plus any other half of the channel peers, i.e. by cooperative signatures from $\mathbf{Q} \subset \mathbf{P}$, where $|\mathbf{Q}| \geq \lfloor |\mathbf{P}|/2 \rfloor + 1$.

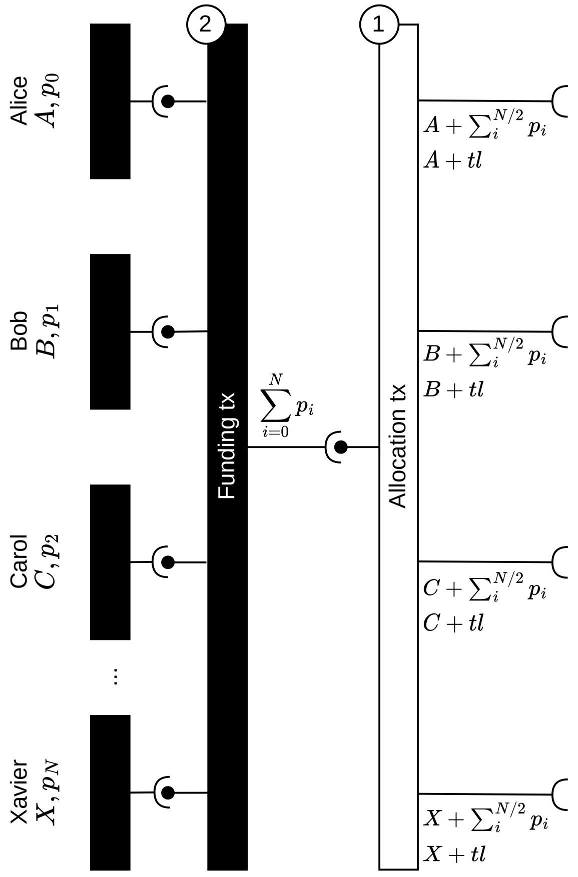
As will be explained later, the unspent outputs of the allocation transaction represent *secured state*, which can be redeemed by the peers onchain not relying on any assumptions; the unspent outputs of the operation transaction DAG represent a *partial state update*, which security relies on a *honest majority assumption* (i.e. that $\lfloor N/2 \rfloor + 1$ of the peers are honest) during the time until the channel is not closed or a new allocation transaction is signed.

B. Channel operations

If all peers are online and cooperative, a channel state is updated by simply signing a new version of the allocation transaction, which requires signatures from all channel peers. In case not all channel peers are online or ready to sign a state update, a new operation transaction is created, which spends inputs only of those peers who like to update their state (for instance, make a payment to each other).

Once all the peers are online, they should cooperate and sign a new version of the allocation transaction, which must

Figure 1: Initial set of transactions at the moment of Nucleus channel establishment



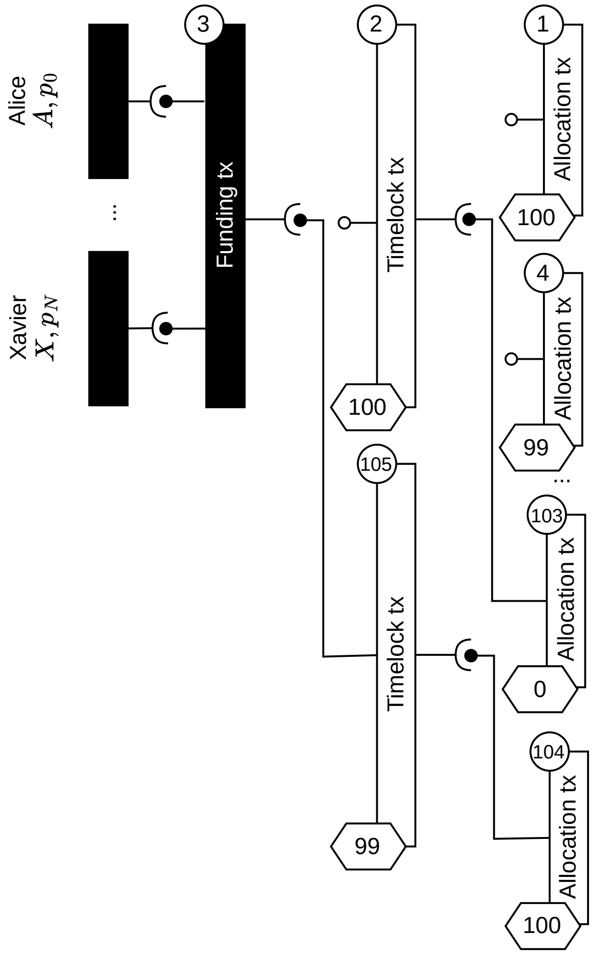
Each row on the diagram represents a peer. Square boxes are transactions. Transaction outputs are shown as sockets with the spending conditions listed as a text note. Each row of the note is an alternative condition (i.e. represents different taproot script path spendings), where big Latin letters represent the signature requirement from a peer and tl stays for "timelock". For spent transaction outputs spending conditions which were satisfied by a witness of the spending transaction are shown in bold and above the output socket.

Black boxes are mined transactions and white - transactions are kept as a part of a state channel. Numbers in circles are the order by which the peers must sign their inputs of the corresponding transactions.

correspond to the previous allocation transaction with all operation transaction graphs applied on top. The peers are incentivised on doing that since it reduces the cost for each of them to close the channel in a unilateral way (the smaller DAG size means lower fees). This also resets the security of the channel state back to the trustless level.

Each update of the allocation transaction must contain a smaller timelock value than the previous transaction – using the same design as was originally proposed in the seminal work on channel factories [1], including an ability to extend the size of the history with additional transaction level above the allocation transaction, creating timelock tree (Figure 2).

Figure 2: Timelock tree of channel transactions



Hexagonal labels at the bottom of the transaction box provide the value for transaction timelock; empty circular inputs represent inputs which prevent transactions from being mined due to timelock conditions (when a transaction with a larger timelock exists). For the rest of the legend refer to Figure 1.

C. Exclusion of peers

A set of peers $\mathbf{E} \subset \mathbf{P}, M := |\mathbf{E}|, M < N$ can be *voluntary* or *involuntary* excluded from the Nucleus channel by the remaining subset of peers $\mathbf{S} := \mathbf{P} \backslash \mathbf{E}$, which cooperatively have to construct *exclusion transactions*.

1) Voluntary exclusion: A *voluntary exclusion transaction* replaces the current allocation transaction and doesn't contain any timelock, able to be mined immediately, in front of any previous allocation transaction. Its preparation requires all $\mathbf{P}$ peers to be online and willing to support the claim of $\mathbf{E}$ peers to be excluded from the channel. Because of the cooperative nature of this workflow, the number of excluded peers can be up to the number of the channel peers minus two ($M \leq N - 2$) – otherwise the procedure becomes equivalent to the cooperative channel closing described below.

The exclusion transaction directly spends the only output of the funding transaction and contains $M + 1$ outputs, where M outputs send funds to the E peers with no encumbrances, and one of the outputs contains the remaining funds organized as $(N - M)$-of-$(N - M)$ multisig. Once signed, the transaction gets published to the network and, once mined, replaces the current funding transaction, providing single funding output to the new channel of $\mathbf{S}$ peers, leaving $\mathbf{E}$ with their funds onchain, outside of the channel.

2) Forced exclusion: In case of forced (or involuntary) exclusion, the peer subset $\mathbf{E}$ may be offline or not willing to cooperate, thus the exclusion transaction can't spend the funding output and a new funding transaction is created on top of the currently-valid allocation transaction. It spends all outputs of the allocation transaction for the $\mathbf{S}$ peers by using cooperative signatures – and has a single output, spendable by M-of-M multisig. Additionally to this, $\mathbf{S}$ peers create and sign a new allocation transaction (NB: the new allocation transaction must be signed before the new funding transaction), which has the same structure as the previous allocation transaction – but with the reduced numbers of peers.

Now, the peers can publish both previous allocations and new funding transactions. Non-zero timelock of previous allocation transaction prevents it from being instantly mined and provides an opportunity for the set-to-be-excluded peers to cooperate and either become responsive by signing a new allocation transaction with a lower timestamp (invalidating published allocation transaction) – or to sign a voluntary exclusion transaction.

Until the published exclusion transactions can be mined or invalidated by a new allocation or voluntary exclusion transaction (if excluded peers become cooperative), the channel continues to operate by using subchannel funding from the new funding transaction and using a new allocation transaction for the subchannel $\mathbf{S}$ underneath it.

The details of the transaction graph for the channel going through the forced exclusion procedure are graphically illustrated in Figure 3.

D. Inclusion of new peers

New peers – or additional funding – can be added to the channel by replacing the current funding transaction with a new transaction, spending the current funding output (signed by all current channel peers $\mathbf{P}$) and adding new peers $\mathbf{X}$ or funds for the existing peers as additional inputs to it. The funding transaction output defines a new channel, $\mathbf{P} + \mathbf{X}$ and requires signatures from all previous and new peers.

E. Cooperative closing

Cooperative closing can happen when all channel peers are online and willing to close the channel. A special *closing transaction*, spending funding transaction output is created and its input is signed by all peers, fulfilling funding transaction spending conditions (N-of-N multisig). The transaction should not contain any timelock and thus can be mined instantly as published - ahead of other allocation transactions.

F. Uncooperative closing

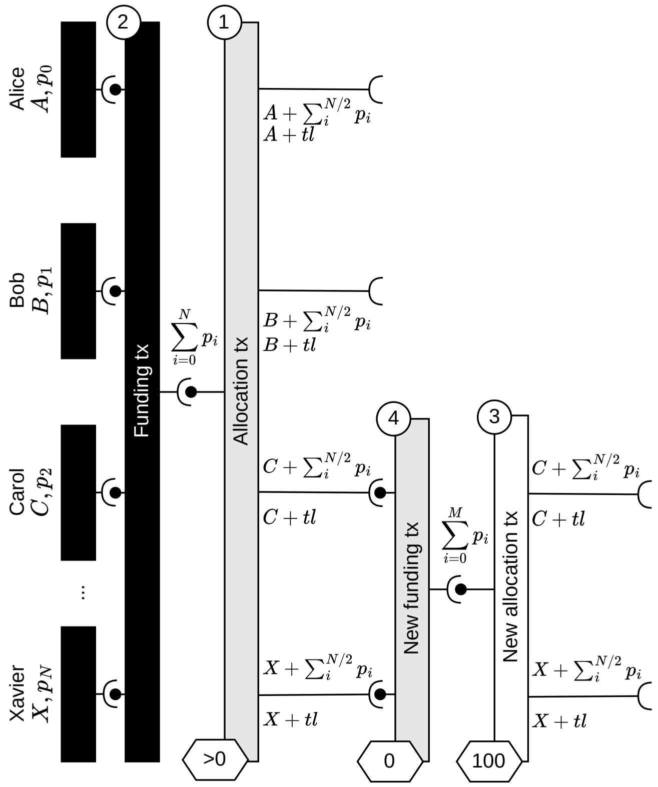
Any party may publish all or some of the channel transactions, including timelock transactions (if the channel operates with a timelock tree as shown in Figure 2), allocation transactions and operation transactions; however, because of the timelock mechanism, this will not lead to the channel closing, since both allocation and timelock transaction contains a non-zero timelock.

If the published transactions do not represent the latest state of the channel, any other peer may publish the latest version, replacing invalid transactions with a valid ones having a lower timelock. Then, the channel peers can either wait for the timelock to run out, which will leave them with their funds available onchain – or save on fees and sign a new funding transaction, which will be double-spending the funding transaction output with zero timelock and use it to fund a new channel. The new channel may include all or some of the current channel peers, thus, this scenario can be seen as a way of doing a channel split or peer exclusion.

III. Channel example

To simplify the explanation, we provide an example of a Nucleus channel with four peers, Alice, Bob, Carol and Dave, continuing to operate when Dave gets offline or uncooperative

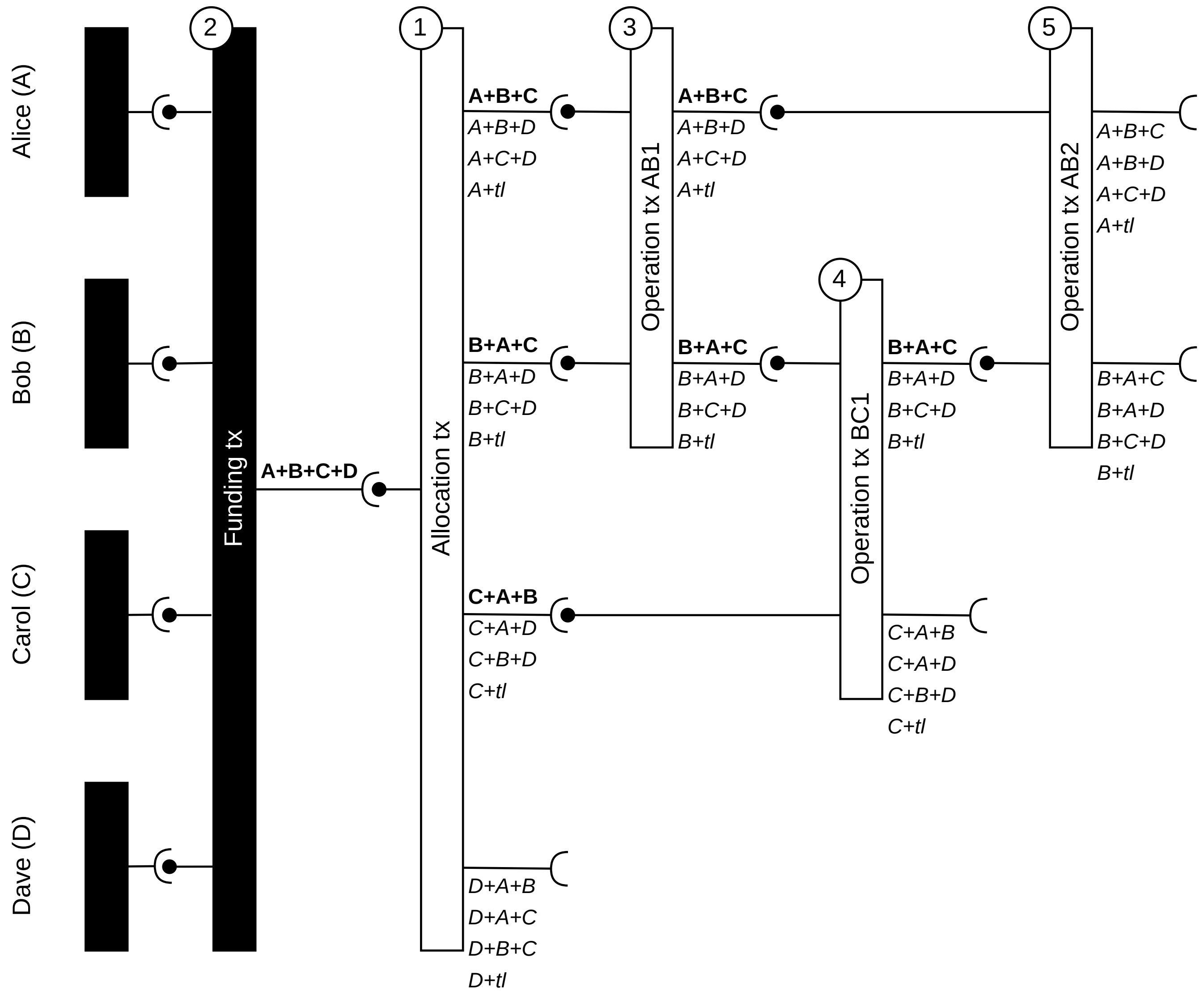
Figure 3: Forced exclusion transaction graph



The graph illustrates the non-voluntary (forced) exclusion of two peers, Alice and Bob, by the remaining set of $N-2$ peers creating a new subchannel. Transactions which are published to the network, but not mined until their timelock will expire, and thus can be replaced if the excluded peers become cooperative, are shown with grey colour.

For the full legend see Figure 2.

Figure 4: Example 4-peer Nucleus channel with one unresponsive peer



For the legend refer to the Figure 1.

(Figure 4). The three remaining peers, instead of signing the updated versions of the allocation transaction with shorter timelocks switch into producing separate operation transactions for each move of the funds between them. The operation transactions have zero timelock and may spend inputs of each other, using the signature from two (or more) transacting peers, plus signatures of other peers such that a signature threshold condition of $\lfloor N/2 \rfloor + 1$ signatures are met. Since the channel has 4 peers, each input of each operation transaction has to be signed by all three remaining peers, which prevents Alice and Bob from cheating and double-spending the output of their operation transaction AB1 when transferring funds with Carol.

If Dave stays offline or uncooperative for a long period of time, the three remaining peers can proceed with the forced exclusion procedure and re-establish the channel without Dave. This will help them to reset the size of the channel operation graph down to a single new allocation transaction.

IV. Analysis

A. Liveness

The protocol requires at least $\lfloor N/2 \rfloor + 1$ peers to be online and cooperative for the channel to be able to continue operations, where all peers remaining online can transact with other peers inside the channel. This is a significant improvement over existing payment channel design [8], [6], [2], where the unresponsiveness of even one peer stops or creates significant obstacles for the channel operations.

B. Capital efficiency

All of the liquidity owned by each of the peers is available for arbitrary transactions with other channel peers, removing the requirement of inbound liquidity present in the modern-day Lightning network [8], [6]. This is a signification improvement for capital efficiency which may boost layer 2 capital supply and adoption.

C. Scalability

1) Transaction speed: Each transaction between channel peers requires either $\lfloor N/2 \rfloor + 1$ or N signatures. The original channel factories proposal [1] requires just two signatures for peer-to-peer transactions, however, N^2 transactions, establishing subchannels between all of the peers, has to be pre-signed, and the liquidity can't be shared between those channels.

2) Storage: Since the channel doesn't use a penalty mechanism and relies on timelocks, it is not required to store signatures for the past transaction, which removes the storage growth problem present in the Lightning network as defined by the BOLT specs [8], [6]. If a node loses part of all of the transaction graph, it doesn't lose access to the funds (see Security subsection below) and the importance of state backups is also reduced compared to the existing approach taken by the BOLT specs [6].

3) Number of peers: The channel computational and storage requirements scale linearly with the growth of the number of peers. This is an improvement compared to the channel factories, where it takes $O(N^2)$ operations to set up the channel factory [1].

Per-peer liquidity accessibility doesn't get reduced with the increase in the number of peers, and the total liquidity usable by a channel grows linearly.

4) Liquidity: Liquidity can be removed and added to channels with channel split-in and split-out operations, where a funding transaction is replaced with a new one. This requires a single onchain operation and is similar to the splicing proposals for the modern Lightning network [6].

5) Inter-channel scalability: Liquidity is freely exchangeable between peers participating in the same channel. Performing transactions outside of the channel will require a form of cross-channel atomic swaps – with mechanics similar to ones used by the BOLT Lightning network for all "routed" transactions [6]. This may include hash-time-locked contracts (HTLC), elliptic curve point-time-locked contractions (PTLC) or other future atomic operation schemes.

D. Security

Liquidity brought to the channel by the peers under no conditions can move without their signature, providing security guarantees against thieves. Nevertheless, in case some of the peers become unresponsive, the channel starts operating with partial state updates (using the operation transaction graph on top of the allocation transaction), which decreases the security of the funds transferred in such operations to the honest majority assumption. This happens due to the fact that outputs in the graph can be double-spend by their owners with cooperation from the majority, and using such outputs as inputs in other operational transactions relies on the honesty of the channel peer majority preventing such double-spends. However, this state is only temporary, since the channel security can be reset to the fully trustless level by excluding uncooperative peers from the channel.

E. Interoperability

Nucleon channels may interoperate with existing Lightning network [6] by meeting two conditions: (1) supporting HTLC-based atomic swaps (or other future atomic swap protocols) and routed payment workflow and (2) having peers running software supporting BOLT P2P and gossip network protocols.

The exact mechanism of atomic swap integration into the Nucleon channels is the subject of a separate research and design effort.

F. Comparative analysis

1) BOLT Lightning network: Nucleus channels remove the requirement for inbound liquidity and allow non-routed multi-peer offchain payments. Any number of peers may participate and use all of their liquidity to interact with any other peer without constraints. However, the case of two peers doesn't provide any advantages compared to the Poon-Dryja channels [8] used in the Lightning network today. Thus, this case can be used only as an intermediary step before adding more peers.

2) Channel factories: An operational transaction may be seen as a form of a "subchannel", created by an allocation transaction acting as a "channel factory" designed after [1], however, there are several important differences. First, parties are able to create new operation transaction even when up to half of peers is offline, while either all possible subchannels has to be pre-signed – or all peers of a channel factory have to be online to sign subchannel commitment transaction. Second, unlike subchannels, which can't share liquidity, operational transactions can spend outputs from other operational transactions. This provides a great increase in flexibility (different peers can go offline at different moments of time - and the remaining peers can still continue to transact) and higher capital efficiency (no capital is kept inside separate subchannels) – at the cost of reduced security: if the majority of the peers are cheating, they can attack other peers with double-spend attacks and non-cooperatively close the channel into that state, stealing some of the funds coming from the compromised operation transactions (see section Honest majority assumption for further analysis of this specific attack vector).

V. Trivia

The protocol name *Nucleus* is inspired by atomic and quantum physics analogy, where baryons participating in the atomic nucleus (nucleons) are bound together by strong forces – however they may still leave the nucleus via nuclear fission. Nucleons are constantly exchanging gluons, shifting from neutrons to protons and back all the time. This is similar to the set of peers bound into the same multi-peer channel, interacting and exchanging funds, and also being able to leave the channel via voluntary or involuntary exclusion procedures.

References

[1] Conrad Burchert, Christian Decker, and Roger Wattenhofer. *Scalable funding of Bitcoin micropayment channel networks*. 2018. URL: https://royalsocietypublishing.org/doi/pdf/10.1098/rsos.180089.

[2] Christian Decker, Rusty Russel, and Olaoluwa Osuntokun. *eltoo: A Simple Layer2 Protocol for Bitcoin*. URL: https://blockstream.com/eltoo.pdf. Retrieved 2018-05-02.

[3] Christian Decker and Roger Wattenhofer. *A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels*. URL: https://tik-old.ee.ethz.ch/file/716b955c130e6c703fac336ea17b1670/duplex-micropayment-channels.pdf.

[4] *DefiLlama Arbitrum TLV*. URL: https://defillama.com/chain/Arbitrum.

[5] *DefiLlama Lightning TLV*. URL: https://defillama.com/protocol/lightning-network.

[6] *Lightning Network In-Progress Specifications*. URL: https://github.com/lightning/bolts.

[7] Andrew Poelstra. *Mimblewimble*. URL: https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf. 2016-10-06 (commit e9f45ec).

[8] Joseph Poon and Thaddeus Dryja. *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. URL: https://lightning.network/lightning-network-paper.pdf. Version 0.5.9.2.