Elmo: Recursive Virtual Payment Channels for Bitcoin

Anonymised Submission

***Abstract*—A dominant approach towards the solution of the scalability problem in blockchain systems has been the development of layer 2 protocols and specifically payment channel networks (PCNs) such as the Lightning Network (LN) over Bitcoin. Routing payments over LN requires the coordination of all path intermediaries in a multi-hop round trip that encumbers the layer 2 solution both in terms of responsiveness as well as privacy. The issue is resolved by *virtual channel* protocols that, capitalizing on a suitable off-chain setup operation, enable the two endpoints to engage as if they had a direct payment channel between them. Once the channel is unneeded, it can be optimistically closed in an off-chain fashion.**

Apart from communication efficiency, virtual channel constructions have three natural desiderata. A virtual channel constructor is *recursive* if it can also be applied on pre-existing virtual channels, *variadic* if it can be applied on any number of pre-existing channels and *symmetric* if it encumbers in an egalitarian fashion all channel participants both in optimistic and pessimistic execution paths. We put forth the first Bitcoin-suitable recursive variadic virtual channel construction. Furthermore our virtual channel constructor is symmetric and offers optimal round complexity for payments, optimistic closing and unilateral closing. We express and prove the security of our construction in the universal composition setting, using a novel induction-based proof technique of independent interest. As an additional contribution, we implement a flexible simulation framework for on- and off-chain payments and compare the efficiency of Elmo with previous virtual channel constructors.

1. Introduction

The popularity of blockchain protocols in recent years has stretched their performance exposing a number of scalability considerations. In particular, Bitcoin and related blockchain protocols exhibit very high latency (e.g., Bitcoin has a latency of 1h [47]) and a very low throughput (e.g., Bitcoin can handle at most 7 transactions per second [18]), both significant shortcomings that jeopardize wider use and adoption and are to a certain extent inherent [18]. To address these considerations a prominent approach is to optimistically handle payments via a *Payment Channel Network* (PCN) (see, e.g., [31] for a survey). Payments over a PCN happen *off-chain*, i.e., without adding any transactions to the underlying blockchain. They only use the blockchain as an arbiter in case of dispute.

The key primitive of PCN protocols is a payment channel. Two parties initiate the channel by locking some funds on-chain and subsequently exchange direct messages to update the state of the channel. The key feature is that state updates are not posted on-chain and hence they remain unencumbered by the performance limitations of the underlying blockchain protocol, making them a natural choice for parties that interact often. Multiple overlapping payment channels can be combined and form a PCN.

Closing a channel is an operation that involves posting the state of the channel on-chain. Closing should be efficient, i.e., needing $O(1)$ on-chain transactions, independent of the number of payments that have occured off-chain. It is also essential that any party can unilaterally close a channel as otherwise a malicious counterparty (i.e., the other channel participant) could prevent an honest party from accessing their funds. This functionality however raises an important design consideration: how to prevent malicious parties from posting old states of the channel. Addressing this issue can be done with some suitable use of transaction *timelocks*, a feature that prevents a transaction or a specific script from being processed on-chain prior to a specific time (measured in block height). For instance, diminishing transaction timelocks facilitated the Duplex Micropayment Channels (DMC) [21] at the expense of bounding the overall lifetime of a channel. Using script timelocks, the Lightning Network (LN) [51] provided a better solution that enabled channels staying open for an arbitrary duration: the key idea was to duplicate the state of the channel between the two counterparties, say Alice and Bob, and facilitate a punishment mechanism that can be triggered by Bob whenever Alice posts an old state update and vice-versa. The script timelocking is essential to allow an honest counterparty some time to act.

Interconnecting channels in LN enables any two parties to transmit funds to each other as long as there is a route of payment channels that connects them. The downside of this mechanism is that it requires the active involvement of all parties along the path for each payment. Instead, *virtual payment channels* suggest the more attractive approach of an one-time off-chain initialization step to set up a virtual payment channel over the preexisting channels, which subsequently can be used for direct payments with complexity —in the optimistic case— independent of the path length. When the virtual channel has exhausted

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.23xxx
www.ndss-symposium.org

its usefulness, it can be closed off-chain if the involved parties cooperate. Initial constructions for virtual channels capitalized on the extended functionality of Ethereum, e.g., Perun [25] and GSCN [27], while more recent work [2] brought them closer to Bitcoin-compatibility (by leveraging adaptor signatures [1]).

We call the two parties that share the channel *endpoints* and the parties of any of the underlying channels *intermediaries*. A virtual channel constructor can be thought of as an *operator* over the underlying payment channel primitive. We identify three natural desiderata for it.

- Recursive. A recursive virtual channel constructor can operate over channels that themselves could be the results of previous applications of the operator. This is important in the context of PCNs since it allows building virtual channels on top of pre-existing virtual channels, allowing the channel structure to evolve dynamically.
- Variadic. A variadic virtual channel constructor can virtualize any number of input payment channels directly, i.e., without leveraging recursion, contrary to a *binary* constructor. This is important in the context of PCNs since it enables applying the operator to build virtual channels of arbitrary length, without the undue overhead of opening, managing and closing multiple virtual channels only to use the one at the "top" of the recursion.
- Symmetric. A symmetric virtual channel constructor offers setup and closing operations that are symmetric in terms of computation, network and storage cost between the two endpoints or the intermediaries (but not necessarily a mix of both) for the optimistic and pessimistic execution paths. Importantly, this ensures that no party is worse-off or better-off after an application of the operator in terms of accessing the basic channel functionality.

We note that recursiveness, while identified already as an important design property [27], has not been achieved for Bitcoin-compatible channels (it was achieved only for DMC-like fixed lifetime channels in [33] and left as an open question for LN-type channels in [2]). This is because of the severe limitations imposed by the scripting language of Bitcoin-compatible systems. With respect to the other two properties, observe that successive applications of a recursive binary virtual channel operator to connect distant endpoints will break symmetry (since the sequence of operator applications will impact the participants' functions with respect to the resulting channel). This is of particular concern since most previous virtual channel constructors proposed are binary [27], [2], [33].

The primary motivation for recursive channels is offering more flexibility in moving off-chain coins quickly, with minimal interaction and at a low cost, even under consistently congested ledger conditions. Without recursiveness and in the face of unresponsive channel parties, one would have to first close its virtual channel on-chain in order to then use some of its coins with another party, which is as slow as any on-chain transaction and in case of high congestion prohibitively expensive. Even if parties collaboratively close the original channel off-chain, the entire channel closes even if only some of its coins are needed. On the other hand, a recursive virtual channel permits using some of its coins with other parties by opening off-chain a new virtual channel on top without involvement of the base parties of the original channel, and even keeping the remaining coins in the latter. Importantly, users can decide to open a recursive virtual channel long after having established their underlying one. This flexibility can inspire confidence in virtual channels, prompting users to transfer more coins off-chain and reduce on-chain congestion.

A scenario only possible with both the recursive and the variadic properties is as follows (Fig. 1): Initially Alice has a channel with Bob, Bob one with Charlie and Charlie one with Dave. Alice opens a virtual channel with Dave over the 3 channels – this needs the variadic property. After a while she realises she has to pay Eve a few times, who happens to have a channel with Dave. Alice interacts just with Dave and Eve to move half of her coins from her virtual channel with Dave to a new one with Eve – this needs the recursive property.

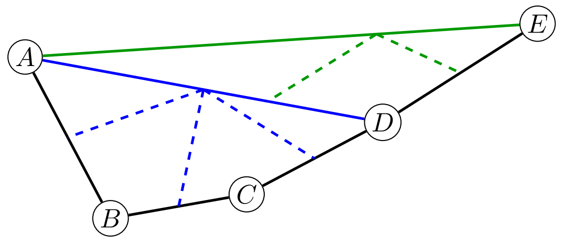

Figure 1: An example Elmo network with 5 nodes, 4 black *simple* (i.e., on-chain) channels, 1 blue virtual channel built only on simple ones, and 1 green virtual channel built on the blue virtual and a simple one. Each virtual channel is connected with its *base* channels with dashed lines of the same colour. The variadic and recursive properties of Elmo are showcased.

Our Contributions. Elmo (named after St. Elmo's fire) is the first Bitcoin-suitable recursive virtual channel constructor that supports channels of indefinite lifetime. In addition, our constructor is variadic and symmetric. Both optimistic and pessimistic execution paths are optimal in terms of round complexity: issuing payments between two endpoints requires just three messages of size independent of the channel length, closing a channel cooperatively needs at most three messages from each party while closing a channel unilaterally demands up to two on-chain transactions for any involved party (endpoint or intermediary) that can be submitted simultaneously, also independent of the channel length. We build Elmo on top of Bitcoin, as this means it can be adapted for any blockchain that supports Turing-complete smart contracts such as Ethereum [56]. The latter provides additional tools to increase Elmo efficiency. Furthermore, Elmo can inspire future blockchain designs that maintain minimal scripting capabilities while providing robust off-chain functionality.

We achieve the above by leveraging a sophisticated virtual channel setup protocol which, on the one hand, enables endpoints to use an interface that is invariant between on-chain and off-chain (i.e., virtual) channels, while on the other, parties can securely close the channel cooperatively off-chain, or instead close unilaterally on-chain, following an arbitrary activation sequence. The latter is achieved by enabling anyone to start closing the channel, while subsequent respondents, following the activation sequence, can choose the right action to complete the closure process by posting a single transaction each.

We formally prove the security of our protocol in the Universal Composition (UC) [15] setting; our ideal functionality $\mathcal{G}_{\mathrm{Chan}}$ represents a single channel and is thus a global functionality, as defined in [7] (cf. Sec. 5). Elmo requires the ANYPREVOUT signature type (slated for inclusion in the next Bitcoin update[1]), which does not sign the hash of the transaction it spends, thus enabling a single pre-signed transaction to spend any output with a suitable script. We leverage ANYPREVOUT to avoid exponential storage and ensure off-chain payments of base channels are practical. We further conjecture that without ANYPREVOUT no efficient off-chain virtual channel constructor over Bitcoin can be built. In particular, if any such protocol (i) offers an efficient closing operation (i.e., with $O(1)$ on-chain transactions), (ii) has parties store the channel state as transactions and signatures in their local storage and (iii) does not require locking on-chain coins (unlike [3]), then each party will need exponentially large space in the number of intermediaries. Note that the second protocol requirement is natural, since, to our knowledge, all trustless layer 2 protocols over Bitcoin require all implicated protocol parties to actively sign off every state transition and locally store the relevant transactions and signatures of their counterparties, thus ensuring their ability to unilaterally exit later.

Related work. The first proposal for PCNs [55] only enabled unidirectional payment channels. As mentioned previously, DMCs [21] with their decrementing timelocks have the shortcoming of limited channel lifetime. This was ameliorated by LN [51] which has become the dominant paradigm for designing Bitcoin-compatible PCNs. LN is currently implemented and operational for Bitcoin. It has also been adapted for Ethereum, named Raiden Network. Compared to Elmo, LN is more lightweight in terms of storage and communication when setting up, but suffers from increased latency and communication for payments, as intermediaries have to actively participate in multi-hop payments. Its privacy also suffers, as intermediaries learn the exact time and value of each payment.

An alternantive payment channel system for Bitcoin that aspires to succeed LN is eltoo [20]. It is conceptually simpler, has smaller on-chain footprint and a more forgiving attitude towards submitting an old channel state than LN (the old state is superseded without punishment), but it needs ANYPREVOUT. Because eltoo and LN function similarly, the previous comparison of Elmo with LN applies to eltoo as well. On a related note, the payment logic of Elmo could also be designed based on the eltoo mechanism instead of the currently used LN.

Perun [25] and GSCN [27] exploit the Turing-complete scripting language of Ethereum to provide virtual state channels. GSCN also uses a per-channel functionality and a similar recursive argument as we use in our UC-security analysis. Their security argument is however flawed, as they incorrectly argue that every level is subroutine respecting with respect to the same environment and subroutines. We believe that, given the versatile scripting of Ethereum, GSCN could be straightforwardly extended to support variadic channels. Similar features are provided by Celer [22]. Hydra [16] provides state channels for Cardano [17].

Solutions alternative to PCNs include sidechains (e.g., [6], [29], [37]), commit-chains (e.g., [50]) and non-custodial chains (e.g., [50], [38], [26]). These approaches offer more efficient payment methods, at the cost of requiring a distinguished mediator, additional tust, or on-chain checkpointing. Furthermore, they do not enable payments between different instances of the same protocol. Due to their conceptual and interface differences as well as differing levels of software maturity, dedicated user studies need to be carried out in order to compare the usability and overall costs of each approach under various usage patterns. Rollups [13], [49] are incompatible with Bitcoin, as they only optimise computation, not storage, whereas Bitcoin has by design minimal computation needs.

Last but not least, a number of works propose virtual channel constructions for Bitcoin. LVPC [33] enables a virtual channel to be opened on top of two preexisting channels and uses a technique similar to DMC, unfortunately inheriting the fixed lifetime limitation. Let *simple channels* be those built directly on-chain, i.e., channels that are not virtual. Bitcoin-Compatible Virtual Channels [2] enables virtual channels on top of two preexisting simple channels and offers two protocols, the first of which guarantees that the channel will stay off-chain for an agreed period, while the second allows the single intermediary to turn the virtual into a simple channel. This strategy has the shortcoming that even if it is made recursive (a direction left open in [2]) after k applications of the constructor the virtual channel participant will have to publish on-chain k transactions in order to close the channel if all intermediaries actively monitor the blockchain.

Donner [3] (released originally concurrently with the first technical report of Elmo) also achieves variadic virtual channels but without recursive ones. This is done by having the funder lock as collateral twice the amount of the desired channel funds: once on-chain with funds that are external to the *base channels* (i.e., the channels that the virtual channel is based on) and once off-chain within its base channel. Thus the required collateral for the funder is double that of other protocols, including LVPC and Elmo. The collateral for all other parties is the same across LVPC, Donner, and

1. https://anyprevout.xyz/

Table 1: Features & requirements comparison of virtual channel protocols

	Unlimited lifetime	Recursive	Variadic	Symmetric	Script requirements
LVPC [33]	✗	◐[a]	✗	✓	Bitcoin
BCVC [2]	✓	✗	✗	✓	Bitcoin
Perun [25]	✓	✗	✗	✓	Ethereum
GSCN [27]	✓	✓	✗	✓	Ethereum
Donner [3]	✗	✗	✓	✗	Bitcoin
this work	✓	✓	✓	✓	Bitcoin + `ANYPREVOUT`

a. lacks security analysis

Elmo. Additionally, a Donner channel needs active periodic collaboration of the endpoints and all base channel parties to refresh its lifetime, therefore a Donner channel does not have a truly unlimited lifetime. We conjecture that using external coins precludes variadic virtual channels with unlimited lifetime. This design choice further means that Donner is not symmetric. Donner also uses placeholder outputs which, due to the minimum coins they need to exceed Bitcoin's *dust limit*, may skew the incentives of rational players and adds to the channel opportunity cost. The aforementioned incentives together with its lack of recursiveness mean that if a party with coins in a Donner channel decides to use them with another party, it first has to close its channel either off-chain, which needs cooperation of all base parties, or else on-chain, with all the delays and fees this entails. Further, its design complicates future iterations that lift its current restriction that only one of the two channel parties can fund the virtual channel. On the positive side, Donner is more efficient than Elmo in terms of storage, computation and communication complexity, and boasts a simpler design. Their work also introduces the *Domino attack*, which we address in Section 7.

We refer the reader to Appx. A for further related work, including a technical issue in Donner and its resolution. Table 1 contains a comparison of the features and limitations of virtual channel protocols, including the current work.

2. Protocol Description

Conceptually, Elmo is split into four actions: channel opening, payments, cooperative closing and unilateral closing. Parties P_1 and P_n may open a channel (P_1, P_n) between them directly on-chain, in which case they follow an opening procedure like that of LN; such a channel is called *simple* and is explaned in more detail below. Otherwise they can open it on top of a path of preexisting *base* channels (P_1, P_2), (P_2, P_3), …, (P_{n-1}, P_n), in which case (P_1, P_n) is a *virtual* channel, also explained later. A channel is either simple or virtual, not both. Since Elmo is recursive, each base channel may itself be simple or virtual. To open a virtual channel, all parties on the path set aside funds in their channels as collateral; they do this by creating so-called *virtual* transactions (txs) that essentially tie the spending of two adjacent base channels into a single atomic action. Once intermediaries are done, they have created a special *funding* output off-chain with the sum of P_1 and P_n's channel balance. P_1 and P_n finally create the channel, applying the logic of simple channels on top of the funding output: their channel is now open. LN demands that the funding output is on-chain, but we lift this requirement. We instead guarantee that either endpoint can put the funding output on-chain unilaterally. A payment over an established channel (described later for simple channels) follows a procedure inspired by LN.

Parties $P_1, \ldots, P_n$ can close optimistically a virtual channel completely off-chain. At a high level, the parties that control the base channels *revoke* their *virtual* txs and the related *commitment* txs. They cannot use their revoked txs anymore. This effectively "peels" one *virtualisation layer*. In the process, they redistribute coins so that intermediaries "break even", while P_1 and P_n get their rightful coins (as reflected in the last virtual channel state) in their base channels ((P_1, P_2) and (P_{n-1}, P_n) respectively).

Finally, the unilateral closing procedure of a virtual channel (P_1, P_n) consists of either P_1 or P_n signing and publishing a number of txs on-chain. In the simplest case, one of the two endpoints, say P_1, publishes her virtual tx. This prompts P_2 to publish her virtual tx as well and so on up to P_{n-1}, at which point the funding output of (P_1, P_n) is automatically on-chain and closing can proceed as for simple channels. If instead any intermediary stays inactive, then a timelock expires and causes a suitable output to become the funding output for (P_1, P_n), at the expense of the inactive party. As discussed later, the funding output employs `ANYPREVOUT` to ensure that the channel needs only a single commitment tx per endpoint, avoiding an exponential state blowup in the recursion depth and making off-chain payments efficient.

In more detail, during the channel opening procedure (in the full version) the two counterparties (i) create new keypairs and exchange the resulting public keys, then (ii) if the channel is virtual, prepare the underlying base channels, next (iii) they exchange signatures for their initial commitment txs and lastly, (iv) if the channel is simple, the *funder* (who is the only party that provides coins for a new channel, like in LN) signs and publishes the *funding* tx to the ledger.

In order to build intuition, let us present examples of the lifecycles of a simple and a virtual channel. Consider 5 parties, $A, B, \ldots, E$ and 4 channels, (A, B), …, (D, E), that will act as the base of the virtual channel (A, E). We first follow the operations of the simple channel (A, B) and then those of (A, E). We simplify some parts of the protocol to aid comprehension — see the full version for more details.

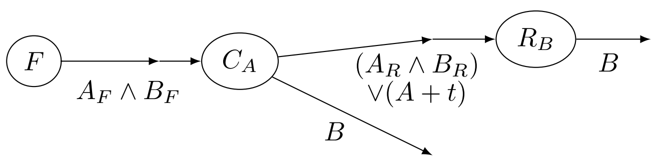

Figure 2: Left to right: funding (F), A's commitment (C_A) and B's revocation (R_B) txs. The symmetric commitment and revocation txs of B and A respectively are not shown. A_F and A_R are A's funding and revocation keys respectively. $A + t$ needs a signature by A after relative timelock of t. The first C_A output is spendable either by both A_R and B_R, or by $A + t$. Inputs and outputs are represented by separate arrows: The input of C_A spending the output of F is shown as two connected arrows.

Simple channel. First A and B generate funding and revocation keypairs and exchange public keys (A_F, B_F, A_R, B_R in Fig. 2 – 2 messages). Each then locally generates the *funding* and the two *commitment* txs (F, C_A in Fig. 2). They sign C_A and exchange the signatures (2 messages). A then publishes the F on-chain. Once it is finalised, the channel is open.

The funding tx F moves A's initial coins to a 2-of-2 multisig, i.e., an output that needs signatures from both A_F and B_F to be spent. There is one commitment tx per party, stored locally off-chain. The one held by A (C_A in Fig. 2) spends the funding tx and has one output for A (initially with all coins) and one for B (initially with 0 coins). A's output can be spent by either a multisig, or by A after a *relative timelock* of t (relative means that the countdown starts at the moment of publication). This is, as we will promptly see, so that B has time to *punish* A if she cheats. B's commitment tx C_B is symmetric.

When A pays c coins to B in the channel, the parties create two new commitment txs. They have the same outputs and scripts as their previous ones, save for the coins: A's outputs have c coins less, B's outputs have c coins more. They sign them and exchange the signatures. In order to ensure only one set of commitment txs is valid at a time, they then revoke their previous ones. They do this by generating and signing the revocation txs of the previous commitment txs. 3 messages are needed for a payment. B's revocation tx (R_B in Fig. 2) gives B the coins that belonged to A in the previous commitment tx and vice versa. This way both parties are disincentivised from publishing an old commitment tx under the threat of losing all their channel coins.

A or B can now unilaterally close (A, B) by simply publishing the latest commitment tx on-chain and waiting for the timelock to expire. Since the last commitment tx is not revoked, punishment is impossible. The mechanics of simple channels are essentially a simplification of LN.

Virtual channel. Assume now that channels (A, B), $\dots, (D, E)$, are open and the "left" party of each owns at least c coins in it. These channels can be either simple or virtual – in the latter case, the virtual channel (A, E) will leverage the recursive property – thanks to the similarity of all layers, the description below is identical in both cases. In order for (A, E) to open using (A, B), $\dots, (D, E)$, as base channels, initially with A having c coins, the 5 parties act as follows. First, they generate and exchange a number of new keys (i.e., all keys that appear in the outputs of Figs. 3–6). Then each base pair removes c coins from the "left" party in their base channel. The updated commitment txs use some of the new keys for the multisig in their input, since, as we will see, so-called *virtual* txs will stand between the funding and the commitment txs from now on. These virtual txs will form the *virtual layer* (Fig. 7).

Next, parties generate and sign the virtual txs and send the signatures among them. These txs sit at the core of Elmo. Intuitively, they force each intermediary to interact with both of its base channels instead of one at a time, ensuring that information relevant to the virtual channel "flows" along the base channels. There are 3 virtual tx types – looking ahead, we now explain them via example of their use during channel closing. The *initiator* virtual tx spends the two funding outputs of an intermediary (say D) and thus needs to be signed by C_F, D_F, and E_F. It produces 4 outputs: one new funding output for each of (C, D) and (D, E) (Fig. 3, top & bottom outputs), one output that refunds the collateral to D (2nd from top) and, crucially, a so-called *virtual* output (3rd from top). The latter can be spent by C with an *extend-interval* tx, the second type of virtual tx (Fig. 4), which needs signatures from all 5 parties (top input). This tx also spends the other, as-of-yet unspent, C's funding output, namely that of (B, C) (bottom input). It has 3 outputs: one refunding the collateral to C (top), another virtual output (middle), and a funding output that replaces the one just spent (bottom). In our example, A also uses its initiator tx, which is different since A is an endpoint (Fig. 5). It spends only the funding output of (A, B) and produces 2 outputs: a new funding output for (A, B) (top) and a virtual output (bottom) – this is the only virtual tx A needs. B can in turn spend A's and C's virtual outputs with a *merge-intervals* tx (Fig. 6), the last virtual tx type, which also needs signatures from the *virtual keys* $A_V \dots E_V$ of all 5 parties. It has 2 outputs: one refunds B and the other produces a new virtual output. Now all base funding outputs are spent, all intermediaries are refunded and the virtual layer has "collapsed". Additionally, the virtual output of B's tx plays the role of the funding output of the virtual channel (A, E).

The virtual txs are designed around two axes: First, each intermediary can publish only one virtual tx, which refunds its collateral exactly once. We will see how this is enforced below. Second, if the chain of virtual txs is at any point broken by, e.g., an inactive intermediary that does not publish its virtual tx, the virtual channel will still be funded correctly and the inactive party will lose its collateral. This is guaranteed because the unclaimed virtual output automatically turns into the funding output of the virtual channel after a timelock. See, e.g., 2nd spending method of the 3rd output of Fig. 3. Keys $A_b \dots E_b$ can be

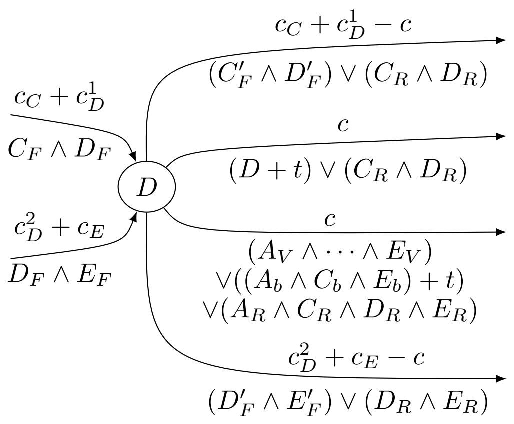
$c_C + c_D^1$
$C_F \wedge D_F$
$c_D^2 + c_E$
$D_F \wedge E_F$
D
$c_C + c_D^1 - c$
$(C'_F \wedge D'_F) \vee (C_R \wedge D_R)$
c
$(D + t) \vee (C_R \wedge D_R)$
c
$(A_V \wedge \cdots \wedge E_V)$
$\vee((A_b \wedge C_b \wedge E_b) + t)$
$\vee(A_R \wedge C_R \wedge D_R \wedge E_R)$
$c_D^2 + c_E - c$
$(D'_F \wedge E'_F) \vee (D_R \wedge E_R)$

Figure 3: A–E virtual channel: D's initiator transaction. Spends the funding outputs of the C–D and D–E channels. D can use it if neither C nor E have published a virtual transaction yet. A_V: A's "virtual" key. A_b: A's "bridge" key.

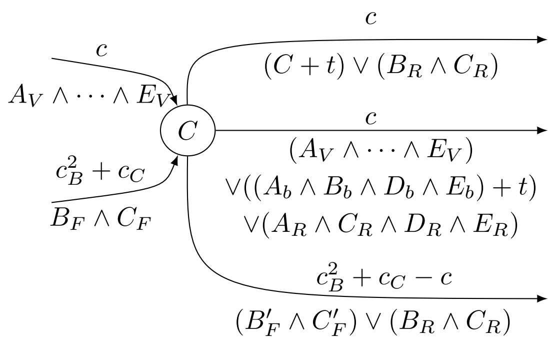
c
$A_V \wedge \cdots \wedge E_V$
$c_B^2 + c_C$
$B_F \wedge C_F$
C
c
$(C + t) \vee (B_R \wedge C_R)$
c
$(A_V \wedge \cdots \wedge E_V)$
$\vee((A_b \wedge B_b \wedge D_b \wedge E_b) + t)$
$\vee(A_R \wedge C_R \wedge D_R \wedge E_R)$
$c_B^2 + c_C - c$
$(B'_F \wedge C'_F) \vee (B_R \wedge C_R)$

Figure 4: A–E virtual channel: One of C's extend-interval transactions. Spends the virtual output of D's initiator transaction and the funding output of the B–C channel. C can use it if D has already published its initiator transaction and B has not published a virtual transaction yet.

spent by *bridge* txs, the output of which funds the (A, E) channel. Bridge txs are discussed in the full version.

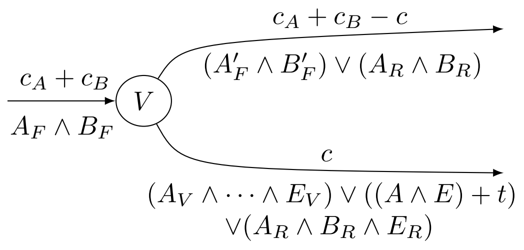
$c_A + c_B$
$A_F \wedge B_F$
V
$c_A + c_B - c$
$(A'_F \wedge B'_F) \vee (A_R \wedge B_R)$
c
$(A_V \wedge \cdots \wedge E_V) \vee ((A \wedge E) + t)$
$\vee(A_R \wedge B_R \wedge E_R)$

Figure 5: A–E virtual channel: A's initiator transaction. Spends the funding output of the A–B channel. A can use it if B has not published a virtual transaction.

Some considerations remain before we can ascertain the security of the scheme. Firstly, we must ensure that no intermediary can publish more than one virtual tx to protect the endpoints from an unbounded sequence of virtual txs preventing them from accessing their funding output indefinitely – note that malicious parties can fabricate arbitrarily many virtual outputs using their own, external to the protocol, coins, therefore if all virtual outputs were identical, an adversary could publish a perpetual stream of merge-intervals txs, spending one valid and one fabricated virtual output. This is safeguarded by specifying on each virtual output the exact sequence of parties that have already published a virtual tx and only allowing the parties at the two edges of the sequence to extend it with their virtual tx. If all intermediaries publish a virtual tx, then the last virtual output that was published is not spendable by another virtual transaction. This ensures that the endpoints will eventually obtain a funding output. Preventing this attack means that intermediaries need to store $O(n^3)$ virtual transactions for a virtual channel over n parties. Secondly, we have to carefully select the exact values of timelocks to ensure that each party has enough time to act. The timelocks increase linearly with the depth of the recursion. The exact values are shown in Sec. 4.

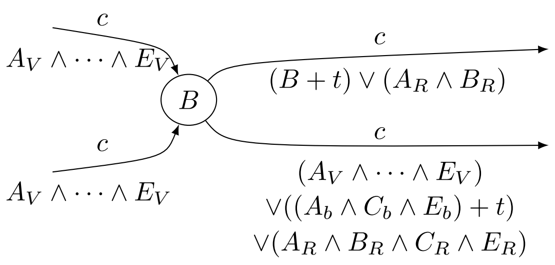
c
$A_V \wedge \cdots \wedge E_V$
c
$A_V \wedge \cdots \wedge E_V$
B
c
$(B + t) \vee (A_R \wedge B_R)$
c
$(A_V \wedge \cdots \wedge E_V)$
$\vee((A_b \wedge C_b \wedge E_b) + t)$
$\vee(A_R \wedge B_R \wedge C_R \wedge E_R)$

Figure 6: A–E virtual channel: One of B's merge intervals transactions. Spends the virtual outputs of A's and C's virtual transactions. B can use it if both A and C have already published their initiator or extend-interval transactions.

We now return to the opening procedure. After the 5 parties set up the virtual txs, they revoke their previous commitment txs. They do this by signing the relevant revocation txs, just like for a simple channel. It takes $12 \cdot (n - 1)$ messages, i.e., 6 messages per endpoint and 12 messages per intermediary, to set up the virtual layer.

At last, the 5 parties have set up virtual layer: Both A and E can unilaterally force the funding output of their virtual channel on-chain, irrespective of the actions of the rest of the parties. Likewise, honest intermediaries can unilaterally retrieve their funds.

A and E finally exchange commitment transactions for their new channel, thus concluding its opening. A and E can pay each other over their virtual channel exactly like they would over a simple channel; we refer the reader to

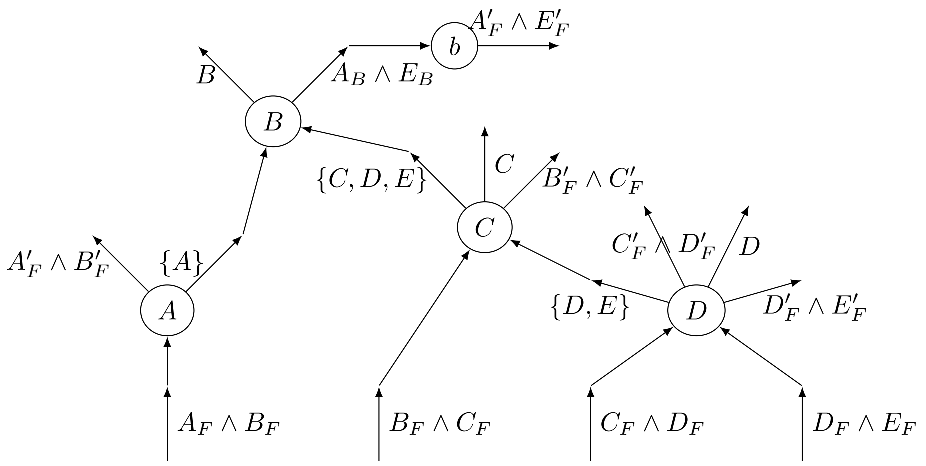

Figure 7: 4 simple channels supporting a virtual one. A and D start closing by publishing their initiator virtual txs, then C publishes its suitable extend-interval virtual tx, after which B publishes its suitable merge-intervals virtual tx. No party stays inactive. The virtual transactions A–D form the virtual layer. Virtual outputs are marked with the set (interval) of parties that have already published a tx. *Bridge* txs like b are used by A and E to convert the various virtual outputs into the same funding output, as ANYPREVOUT only works across identical outputs.

the relevant description above.

Note that funding outputs use the ANYPREVOUT flag, thus ensuring that a single pair of commitment txs can spend any of the funding outputs. If ANYPREVOUT were not used, each virtual layer would need a copy of the entire set of discussed txs for each possible funding output of its base layer, resulting in exponential storage requirements. To make matters worse, a payment over (A, E) would need A and E to renegotiate exponentially many commitment txs, as well as recalculate all their downstream txs, which would in turn need interaction with intermediaries of all virtual channels built over (A, E), completely defeating the essence of payment channels.

Cooperative closing. To enhance usability, our protocol allows the virtual channel to be closed off-chain, given that all parties cooperate. To do this, the endpoints first let the intermediaries know their final virtual channel balance. Then the parties of each base channel create new commitment txs for their channels, moving the collateral back into the channel: the "left" party gets A's coins and the "right" one gets E's. Thus all intermediaries "break even" across their two channels. Once this is done, the parties revoke all virtual txs, using a logic similar to the revocation procedure of simple channels but scaled up to all parties. This is why all virtual tx outputs (Figs. 3–6) have a spending method with $A_R \dots E_R$ keys.

What if one party does not cooperate? Then one or more of the other parties must close unilaterally on-chain. Fig. 7 shows how this would play out if A and D initiated this procedure.

Our protocol is recursive because both simple and virtual channels are ultimately represented by a funding output that either already is or can be put on-chain, therefore new virtual channels can be built on either. Both simple and virtual channels avoid key reuse on-chain, thus ensuring party privacy from on-chain observers.

3. Model

3.1. $\mathcal{G}_{\text{Ledger}}$ Functionality

In this work we embrace the Universal Composition (UC) framework [15] together with its global subroutines extension, UCGS [7], to model parties, network interactions, adversarial influence and corruptions, as well as formalise and prove security. We model the Bitcoin ledger with the $\mathcal{G}_{\text{Ledger}}$ functionality as defined in [10], [8]. $\mathcal{G}_{\text{Ledger}}$ formalizes an ideal data structure that is distributed and append-only, akin to a blockchain. Participants can read from $\mathcal{G}_{\text{Ledger}}$, which returns an ordered list of transactions. Additionally a party can submit a new transaction which, if valid, will eventually be added to the ledger when the adversary decides, but necessarily within a predefined time window. This property is named liveness. Once a transaction becomes part of the ledger, it then becomes visible to all parties at the discretion of the adversary, but necessarily within another predefined time window, and it cannot be reordered or removed. This is named persistence.

Moreover, $\mathcal{G}_{\text{Ledger}}$ needs the $\mathcal{G}_{\text{CLOCK}}$ functionality [34], which models the notion of time. Any $\mathcal{G}_{\text{CLOCK}}$ participant can request to read the current time and inform $\mathcal{G}_{\text{CLOCK}}$ that her round is over. $\mathcal{G}_{\text{CLOCK}}$ increments the time by one once all parties have declared the end of their round. Both $\mathcal{G}_{\text{Ledger}}$ and $\mathcal{G}_{\text{CLOCK}}$ are global functionalities [7] and therefore can be accessed directly by the environment. The definitions of $\mathcal{G}_{\text{Ledger}}$ and $\mathcal{G}_{\text{CLOCK}}$ can be found in the full version.

3.2. Modelling time

The protocol (Π_{Chan}) and functionality ($\mathcal{G}_{\text{Chan}}$) defined in this work do not use $\mathcal{G}_{\text{CLOCK}}$ directly. The only notion of time is provided by the blockchain height, as reported by $\mathcal{G}_{\text{Ledger}}$. We thus omit it in our lemmas and theorems statements to simplify notation; it should normally appear as a hybrid together with $\mathcal{G}_{\text{Ledger}}$.

Our protocol is fully asynchronous, i.e., the adversary can delay any network message arbitrarily long. The protocol is robust against such delays, as an honest party can unilaterally prevent loss of funds even if some of its messages are dropped by $\mathcal{A}$, given that the party can communicate with $\mathcal{G}_{\text{Ledger}}$. In other words, no extra synchrony assumptions to those required by $\mathcal{G}_{\text{Ledger}}$ are needed. We also note that, following the conventions of single-threaded UC execution model, the duration of local computation is not taken into account (as long as it does not exceed its polynomial bound).

4. Protocol Pseudocode

We here present a simplified version of the Π_{Chan} protocol. We omit complications imposed by UC.

```
Process $\Pi_{\text{Chan}}$– self is $P$

• At the beginning of each activation:
    if we have not been activated for more than $p$ blocks then
        We are negligent // no balance security guarantees
• Open channel with counterparty $P'$: // $A$, $B$ of Fig. 2 are $P$, $P'$ resp.
    Generate funding ($P_F$) and revocation ($P_R$) keypairs.
    Exchange funding, revocation & own ($P$) public keys with $P'$.
    if opening virtual (off-chain) channel then
        Run next bullet "Host a virtual channel" as endpoint.
    Exchange & verify signatures by $P_F$ and $P'_F$ on $C_P$ and $C_{P'}$.
    if opening simple (on-chain) channel then
        Prepare and submit funding tx ($F$) to ledger and wait for its inclusion. // only one party funds the channel, so the funding tx needs only the funder's signature
        $t_P \leftarrow s + p$ // simple channel timelock
        // $s$: max blocks before submitted tx enters ledger
• Host virtual channel of $c$ coins (endpoint/intermediary): // Fig. 3–7
    Ensure we have at least $c$ coins.
    Generate one new funding keypair ($P'_F$), $O(n^2)$ virtual keypairs ($P_V$) (1 per hop and party, to control which virtual txs can spend which) and one virtual revocation keypair ($P_R$).
    // Revocation keys in virtual and commitment txs are distinct, but we reuse notation in Fig. 2 and Figs. 3–7 for simplicity.
    Exchange these public keys with all base channel parties.
    Generate and sign new commitment txs ($C_P$, $C_{P'}$) with our base channel counterparty/ies (1 if endpoint, 2 if intermediary), using the new funding and latest revocation keys and reducing by $c$ the balance of the party "closer" to the funder.
    Exchange signatures with counterparty/ies and verify them.
    Generate and sign all $O(n^3)$ virtual and bridge txs ($b$ of Fig. 7) with the virtual ($P_V$) and bridge ($P_B$) keys.
    Exchange signatures among all base channel parties and verify that all our virtual txs have signed virtual inputs.
    Exchange with counterparty/ies and verify signatures for the funding inputs of our initiator and extend-interval txs.
    Exchange with counterparty/ies and verify signatures for the revocation txs of the previous channel state.
    if we are intermediary then
        $t_P \leftarrow \max\{t \text{ of left channel}, t \text{ of right channel}\}$
    else // we are endpoint
        $t_P \leftarrow p + \sum_{j=2}^{n-1}(s - 1 + t_j)$ // max delay is $O($sum of intermediaries' delays). Occurs when we use initiator tx and each intermediary uses extend-interval tx sequentially.
• React if counterparty publishes virtual tx:
    Publish our only valid virtual tx. // if both counterparties have published, this is a merge-intervals tx, otherwise it is an extend-interval tx.
• Pay $x$ coins to $P'$ over our (simple or virtual) channel:
    Ensure we have at least $x$ coins.
    if we host another virtual channel then
        Ensure new balance prevents griefing.
    Generate and sign new commitment txs ($C_P$, $C_{P'}$), with $x$ coins less for the payer and $x$ coins more for the payee.
    Exchange and verify signatures by funding keys ($P_F$, $P'_F$).
    Sign revocation txs ($R_P$, $R_{P'}$) corresponding to old commitment txs with revocation keys ($P_R$, $P'_R$).
    Generate next revocation keypairs.
    Exchange and verify revocation signatures and public keys.
• Close virtual channel unilaterally:
    Publish initiator & bridge tx. // Funding output is on-chain
    Publish our latest commitment tx on-chain.
• Close virtual channel cooperatively: // Only if not hosting
    Endpoints send their balance $(c_1, c_2)$ to all parties.
    Parties ensure endpoints agree and $c_1 + c_2 = c$.
    All parties generate and sign new commitment txs with:
      – the funding keys used before opening virtual channel,
      – the new revocation keys, and
      – $c_1$ more coins to party closer to funder, $c_2$ to the other.
    All parties generate new revocation keypairs.
    All pairs exchange & verify sigs & new revocation public keys.
    All parties generate and sign revocation txs for the old virtual, bridge and commitment txs with their virtual revocation keys.
```

```
        All pairs exchange and verify these signatures.
  • Punish malicious counterparties: // Run every $p$ blocks
        if an old commitment tx is on-chain then
            Sign w/ revocation key & publish corresp. revocation tx.
        if the ledger contains an old virtual or bridge tx then
            Sign w/ revocation key & publish corresp. revocation tx(s).
```

Figure 8: High level pseudocode of the Elmo protocol

5. Security

Before providing the UC-based security guarantees, it is useful to obtain concrete properties directly from our protocol. We first delineate the security guarantees Elmo provides by proving Lemma 1 which discusses the conservation of funds. The formal statements along with all proofs are found in the full version. Informally, it establishes that if an honest, non-negligent party (Fig. 4, 1st bullet) was implicated in a channel that has now been unilaterally closed, then the party will have at least the expected funds on-chain.

Lemma 1 (Real world balance security (informal)). Consider a real world execution with $P \in \{Alice, Bob\}$ an honest, non-negligent party. Assume that all of the following are true:

- P opened the channel, with initial balance c,
- P is the host of n channels, each funded with f_i coins,
- P has cooperatively closed k channels, where the i-th channel transferred r_i coins from the hosted virtual channel to P,
- P has sent m payments, each involving d_i coins,
- P has received l payments, each involving e_i coins.

If P closes unilaterally, eventually there will be h outputs on-chain spendable only by P, each of value c_i, such that

$$\sum_{i=1}^{h} c_i \geq c - \sum_{i=1}^{n} f_i - \sum_{i=1}^{m} d_i + \sum_{i=1}^{l} e_i + \sum_{i=1}^{k} r_i \ . \quad (1)$$

The expected funds are [initial balance - funds for hosted virtuals + funds returned from hosted virtuals - outbound payments + inbound payments]. Note that the funds for hosted virtuals only refer to those funds used by the funder of the virtual channel, not the rest of the base parties.

Proof Sketch [Lemma 1]: All execution paths are followed, keeping track of the resulting balance in each case and concluding that balance is secure in all cases, except if signatures are forged. □

It is important to note that in fact Π_{Chan} provides a stronger guarantee: a party can always unilaterally close its channel and obtain the expected funds on-chain within a known number of blocks. This stronger guarantee is sufficient to make Elmo reliable enough for real-world applications. However an ideal world functionality with such guarantees would have to be aware of specific txs and signatures, making it as complicated as the protocol, thus violating the spirit of the simulation-based security paradigm.

$\mathcal{G}_{\text{Chan}}$ (Appx. B) halts on security breaches (e.g., lower than expected balance). In the full version we prove the "no-halt" Lemma, which informally states that if an ideal party is honest, $\mathcal{G}_{\text{Chan}}$ does not halt with overwhelming probability.

Since $\mathcal{G}_{\text{Chan}}$ corresponds to a single channel, which in turn can form the base of multiple independent virtual channels and thus needs to be accessible by all of them, $\mathcal{G}_{\text{Chan}}$ is a global functionality, i.e., it can communicate with entities outside the (single-channel) protocol. The alternative of modelling all channels within a single protocol [36] leads to a monolithic, hard-to-reuse ideal functionality.

As per Def. 19 of [14], a *subroutine respecting* protocol must not pass input to a party of a different session. In order to open a virtual channel however, Π_{Chan} passes inputs to a Π_{Chan} instance of another session, thus Π_{Chan} is not subroutine respecting. To address this, we first add a superscript to Π_{Chan}, i.e., Π^n_{Chan}. Π^1_{Chan} is always a simple channel. This is done by ignoring instructions to OPEN on top of other channels. As for higher superscripts, $\forall n \in \mathbb{N}^*, \Pi^{n+1}_{\text{Chan}}$ is the same as Π_{Chan} but with base channels of a maximum superscript n. It then holds that $\forall n \in \mathbb{N}^*, \Pi^n_{\text{Chan}}$ is $(\mathcal{G}_{\text{Ledger}}, \Pi^1_{\text{Chan}}, \ldots, \Pi^{n-1}_{\text{Chan}})$*-subroutine respecting* [7]. The same superscript trick is done to $\mathcal{G}_{\text{Chan}}$, thus the composition theorem of [7] is applicable (Appx. C). To the best of the authors' knowledge, this recursion-based proof technique for UC security is novel. It is of independent interest and can be reused to prove UC security in protocols that may use copies of themselves as subroutines. Our UC-security theorems (found in the full version) state that $\forall n \geq 1, \Pi^n_{\text{Chan}}$ UC-realises $\mathcal{G}^n_{\text{Chan}}$. Furthermore, all ideal global subroutines can be replaced with their real counterparts (Lemma 3 and Theorem 4).

6. Efficiency Evaluation & Simulations

We offer here a cost and efficiency comparison of this work with LVPC [33] and Donner [3]. We focus on these due to their exclusive support of virtual channels over any number of base channels. We remind that LVPC achieves this via recursion, while Donner because it is variadic (cf. Table 1).

We count the communication, storage and on-chain cost of a virtual channel in each protocol. We also simulate the execution of a large number of payments among many parties and derive payment latency and fees. We thus obtain an end-to-end understanding of both the requirements and benefits of each protocol.

Cost calculation. Consider the setting of 1 funder (P_1), 1 fundee (P_n) and $n-2$ intermediaries ($P_2, \ldots, P_{n-1}$) where P_i has a base channel with each of P_{i-1}, P_{i+1}. We compare the costs of off-chain opening (Table 2) and on-chain closing unilaterally (Table 3).

Regarding opening, in Table 2 we measure for each of the 3 protocols the number of communication rounds required, the total size of outgoing messages as well as the amount of space for storing channel data. We measure funder, fundee and intermediary requirements, along with the aggregate for all parties. The communication rounds for a party are calculated as its [#incoming messages + #outgoing messages]/2. The size of outgoing messages and the stored data are measured in raw bytes. The data is counted as the sum of the relevant channel identifiers (8 bytes each, as defined by the Lightning Network specification[2]), transaction output identifiers (36 bytes), secret keys (32 bytes each), public keys (32 bytes each, compressed form – these double as party identifiers), Schnorr signatures (64 bytes each), coins (8 bytes each), times and timelocks (both 4 bytes each). UC-specific data is ignored.

For LVPC, multiple different topologies can support a virtual channel between P_1 and P_n (all of which need $n-1$ base channels). We here consider the case in which the funder P_1 first opens one virtual channel with P_3 on top of channels (P_1, P_2) and (P_2, P_3), then another virtual channel with P_4 over (P_1, P_3) and (P_3, P_4) and so on up to the (P_1, P_n) channel, opened over (P_1, P_{n-1}) and (P_{n-1}, P_n). We choose this topology as P_1 cannot assume that there exist any virtual channels between other parties (which could be used as shortcuts).

A subtle byproduct of the above topology is that during the opening phase of LVPC every intermediary P_i acts both as a fundee in its virtual channel with the funder P_1 and as an intermediary in the virtual channel of P_1 with the next party P_{i+1}. The above does not apply to the first intermediary P_2, since it already has a channel with P_1 before the protocol starts. Table 2 shows the total cost of intermediaries $P_3, \ldots, P_{n-1}$. The first intermediary P_2 incurs instead [intermediary's costs - fundee's costs] for all three measured quantities.

For Elmo, the data are derived assuming a virtual channel opens directly on top of $n-1$ base channels. In other words the channel considered is opened without the help of recursion and only leverages the variadic property of Elmo. In Table 2 the resources calculated for Elmo are exact for $n \geq 4$ parties, whereas for $n = 3$ they slightly overestimate.

For the closing comparison, we measure on-chain transactions' size in vbytes[3], which map directly to on-chain fees and thus are preferable to raw bytes. Using vbytes also ensures our comparison remains up-to-date irrespective of the network congestion and bitcoin-to-fiat currency exchange rate at the time of reading. We use a suitable tool[4] to aid size calculation. For the case of intermediaries, in order to only show the costs incurred due to supporting a virtual channel, we subtract the cost the intermediary would pay to close its channel if it was not supporting any virtual channel.

The on-chain number of transactions to close a virtual channel in the case of LVPC is calculated as follows: One "split" transaction is needed for each base channel ($n-1$ in total), plus one "merge" transaction per virtual channel ($n-2$ in total), plus a single "refund" transaction for the virtual channel, for a total of $2n-2$ transactions.

In Table 3 we measure for each of the three protocols the worst-case on-chain cost for a party in order to unilaterally close its channel. The cost is measured both in the number of transactions and in their total size.

For the two endpoints (funder and fundee), we show the cost of unilaterally closing the virtual channel. On the other hand, for each intermediary we report the cost of closing a base channel. We also present the worst-case total on-chain cost, aggregated over all parties. Note that the latter cost is not simply the sum of the worst-case costs of all parties, as one party's worst case is not necessarily the worst case of another. This cost rather represents the maximum possible load an instantiation of each protocol could add to the blockchain when closing.

We note that Elmo exploits MuSig2 [44], [48] to reduce both its on-chain and storage footprint: the n signatures that are needed to spend each virtual and bridge output can be securely reduced to a single aggregate signature. The same cannot be said for Donner, since this technique cannot optimise away the n outputs of the funder's transaction $\mathtt{tx}^{\mathtt{vc}}$. Likewise LVPC cannot gain a linear improvement with this optimisation, since each of its relevant transactions ("split", "merge" and "refund") needs constant signatures.

We furthermore note that, since human connections form a small world [45], we expect that in practice the need for virtual channels with a large number of intermediaries will be exceedingly rare. This is corroborated by the fact that LN only supports payments of up to 20 hops without impact to its usefulness. Therefore, the asymptotic network and storage complexity are not as relevant as the concrete costs for specific, low values of n. Under this light, the overhead of Elmo is tolerable. For example, the network requirements for a funder when opening an Elmo channel of length 6 are less than 3 times those of Donner and slightly cheaper than LVPC (Table 2).

Payment simulations. We implemented a simulation framework[5] in which a list of randomly generated payments are carried out. A single simulation is parametrised by a list of payments (sender, receiver, value triples), the protocol (Elmo, Donner, LVPC, LN or on-chain only), which future payments each payer knows and the utility function it maximises. The knowledge function defines which future payments inform each decision. Several knowledge functions are provided, such as full knowledge of all future payments and knowledge of the payer's next m payments.

The utility of a payment is high when its latency and fees are low, it increases the payer's network centrality, and reduces distance from other parties. We weigh low latency and fees most, then small distance and high centrality last. Latency here is the time that passes until a payment is finalized. This depends on whether the party decides to do an on-chain transaction, open a new channel, or do an off-chain transaction if possible. Note that the first two are bound by

2. https://github.com/lightning/bolts/blob/master/07-routing-gossip.md#definition-of-short_channel_id

3. https://en.bitcoin.it/wiki/Weight_units

4. https://jlopp.github.io/bitcoin-transaction-size-calculator/

5. gitlab.com/anonymised-submission-8778e084/virtual-channels-simulation

Open												
	Funder			Fundee			Intermediary			Total	
	party rounds	size		party rounds	size		party rounds	size		size	
		sent	stored		sent	stored		sent	stored	sent	stored
LVPC	$8(n-2)$	$1381(n-2)$	$3005(n-2)$	7	1254	2936	16	2989	6385	$4370n-8740$	$9390n-18780$
Donner	2	$184n+829$	$1332.5k+ 43n+125.5$	1	$43n+192.5$	$1332.5k+ 43n+125.5$	1	547	$1332.5k+ 43n+125.5$	$774n-71$	$1332.5kn+ 43n^2+125.5n$
Elmo	6	$32n^3-128n^2 +544n-276$	$\frac{128}{3}n^3-128n^2 +\frac{1276}{3}n+220$	6	$32n^3-128n^2 +544n-340$	$\frac{128}{3}n^3-128n^2 +\frac{1276}{3}n+220$	12	$96n^3-256n^2 +404n-40$	$96n^3-256n^2 +468n+88$	$96n^4-384n^3+ 724n^2+240n-792$	$96n^4-\frac{1088}{3}n^3+ 660n^2+\frac{8}{3}n+520$

Table 2: Open efficiency comparison of virtual channel protocols with n parties and k payments

Unilateral Close								
	Intermediary		Funder		Fundee		Total	
	#txs	size	#txs	size	#txs	size	#txs	size
LVPC	3	627	2	383	2	359	$2n-2$	$435n-510.5$
Donner	1	204.5	4	$704+43n$	1	136.5	$2n$	$458n-26$
Elmo	1	297.5	3	376	3	376	$n+1$	$254.5n-133$

Table 3: On-chain worst-case closing efficiency comparison of virtual channel protocols with n parties

the ~$10'$ latency of Bitcoin blocks. We measure latency in seconds. Recognising the arbitrary nature of the concrete weights, we chose them before running our simulations in order to minimise bias. Each payment is carried out by dry-running all known future payments with the three possible payment kinds (simply on-chain, opening a new channel, using existing channels), comparing their utility and executing the best one.

Our simulation framework is of independent interest, as it is flexible and reusable for a variety of payment network protocol evaluations. We here show the performance of the 3 protocols with respect to the metrics payment channels aim to improve, namely payment latency (Fig. 9) and fees (Fig. 10).

Due to the privacy guarantees of LN, we are unable to obtain real-world off-chain payment data. We therefore generate payments randomly. More specifically, we provide three different payment workloads to mimic different usage schemes: For the first, each party has a *preferred receiver*, chosen uniformly at the beginning, which it pays half the time, the other half choosing the payee uniformly at random. Each payment value is chosen uniformly at random from the $[0, \max]$ range, for $\max = \frac{\text{(initial coins)}\cdot\text{\#players}}{\text{\#payments}}$. We employ 1000 parties, with a knowledge function disclosing to each party its next $m = 100$ payments, as it appeared this is a realistic knowledge function for this case. This scenario occurs when new users are onboarded with the intent to primarily pay a single counterparty, but sporadically pay others as well. For the second, in an attempt to emulate real-world payment distributions, the value and number of incoming payments of each player are drawn from the zipf [52] distibution with parameter 2, which corresponds to real-world *power-law* distributions with a heavy tail [11]. Each payment value is chosen according to the zipf(2.16) distribution which corresponds to the $80/20$ rule [23], moved to have a mean equal to $\frac{\max}{2}$. We consider 500 parties, and a knowledge function with $m = 10$, as this is more aligned with real-world scenarios. For the third, all choices are made *uniformly at random*, with each payment chosen uniformly from $[0, \max]$, employing a total of 3000 parties, again with each knowing its next $m = 10$ payments. For all scenarios the payer of each payment is chosen uniformly at random, no channels exist initially, and all parties initially own the same amount of coins on-chain. A payer funds a new channel with the minimum of all the on-chain funds of the payer and the sum of the known future payments to the same payee plus 10 times the current payment value. The coins that fund new virtual channels are essentially decided in the same way as funding coins for simple channels. The only difference is that the availability of funding from the underlying channel is artificially limited so that it does not deplete too fast. This ensures that the underlying channel can be used as a base for several virtual channels, as well as for more transactions. The authors would use this heuristic to allocate their own funds in real-life use cases. The fees for new virtual channels are decided as follows. There is a fixed base fee that has to be payed for each intermediary. Furthermore, each intermediary gets a small fixed proportion of the number of coins that will be put on the virtual channel. The number of parties is chosen to ensure the simulation completes within a reasonable length of time.

In order to avoid bias, we simulate each protocol with the same payments. We simulate each scenario with 20 distinct sets of payments and keep the average. In Figs. 9 and 10, scale does not begin at zero for better visibility. Payment delays are calculated based on which protocol is used and how the payment is performed. Average latency is high as it describes the whole run, including slow on-chain payments and channel openings. Total fees are calculated by summing the fee of each "basic" event (e.g., paying an intermediary for its service). None of the 3 protocols provide fee recommendations, so we use the same baseline fees for the same events in all 3 to avoid bias. These fees are not systematically chosen, therefore Fig. 10 provides relative, not absolute, fees.

As Fig. 9 shows, delays are primarily influenced by the payment distribution and only secondarily by the protocol: The preferred receiver is the fastest and the uniform is the slowest. This is reasonable: In the preferred receiver scenario at least half of each party's payments can be performed over a single channel, thus on-chain actions are reduced. On the other hand, in the uniform scenario payments are spread over all parties evenly, so channels are not as well utilised.

As evidenced, Elmo is the best or on par with the best protocol in every case. We attribute this to the flexibility of Elmo, as it is both variadic and recursive, thus able to exploit the cheapest payment method in all scenarios. In particular, Donner is consistently the most fee-heavy protocol and LVPC the slowest. Elmo experiences similar delays to Donner and slightly higher fees than LVPC.

7. Discussion and Future work

Domino attack. In [3] the Domino attack is presented. Briefly, it claims that a malicious virtual channel member can force other channels to close. To illustrate this, suppose A and E of Fig. 7 open channels (A, B), (D, E) and (A, E) with the sole intent of forcing channels (B, C) and (C, D) to close. We observe that, contrary to the attack goal, honest base parties are only forced to publish a single virtual transaction each, which places their funding outputs on-chain but does not cause the base channels to close. There is still a small downside: the channel capacity is reduced by the collateral, which is paid directly to one of the two base channel parties. Since no coins are stolen, the only cost to B, C and D is the on-chain fees of one transaction. This is an inherent but small risk of recursive channels, which must be taken into account when making one's channel the base of another. This risk can be eliminated by making an attacker pay the fees for the others' on-chain transactions. This fee need not apply in case of cooperative closing nor during normal operation and can be reduced for reputable parties. Suitable reputation systems, as well as mechanisms for assigning inactivity blame (i.e., proving which parties tried collaborative closing before closing unilaterally), which are needed to determine who must pay the fees, can be designed but are beyond the scope of this work.

Furthermore, the following modification to Elmo eliminates the channel capacity reduction under a Domino attack, while also reducing the on-chain footprint of unilateral closing: from each virtual tx, we eliminate the output that directly pays a party (e.g., 1st output of Fig. 4) and move its coins into the funding output of this transaction (e.g., 3rd output of Fig. 4). We further ensure at the protocol level that the base party that owns these coins never allows its channel balance to fall below the collateral, until the supported virtual channel closes. This change ensures that the collateral automatically becomes available to use in the base channel after the virtual one closes, keeping more funds off-chain after a Domino attack. This approach however has the drawback that, depending on which of the two parties first spends the funding output (with a virtual tx), the funds allocation in the channel differs. Base parties thus would have to maintain two sets of commitment and revocation txs, one for each case. Since this overhead encumbers the optimistic, cooperative case and only confers advantages in the pessimistic case, we choose not to adopt this approach into our design.

Future work. A number of features can be added to our protocol for additional efficiency, usability and flexibility. First of all, in our current construction, each time a particular channel C acts as a base channel for a new virtual channel, one more "virtualisation layer" is added. When one of its owners wants to close C, it has to put on-chain as many transactions as there are virtualisation layers. Also the timelocks associated with closing a virtual channel increase with the number of virtualisation layers of its base channels. Both these issues can be alleviated by extending the opening and cooperative closing subprotocol with the ability to cooperatively open and close multiple virtual channels in the same layer, either simultaneously or by amending an existing virtualisation layer.

In this work we only allow a channel to be funded by one of the two endpoints. This limitation simplifies the execution model and analysis, but can be lifted at the cost of additional protocol complexity.

Furthermore, as it currently stands, the timelocks calculated for the virtual channels are based on p and s, which are global constants that are immutable and common to all parties. The parameter s stems from the liveness guarantees of Bitcoin, as discussed in Proposition 2 (Appx. C) and therefore cannot be tweaked. However, p represents the maximum time (in blocks) between two activations of a non-negligent party, so in principle it is possible for the parties to explicitly negotiate this value when opening a new channel and even renegotiate it after the channel has been opened if the counterparties agree. We leave this usability-augmenting protocol feature as future work.

Our protocol is not designed to "gracefully" recover from a situation in which halfway through a subprotocol, one of the counterparties starts misbehaving. Currently the only solution is to unilaterally close the channel. This however means that DoS attacks (that still do not lead to channel fund losses) are possible. A practical implementation of our protocol would need to expand the available actions and states to be able to transparently and gracefully recover from such problems, avoiding closing the channel where possible, especially when the problem stems from network issues and not from malicious behaviour.

Additionally, our protocol does not feature one-off multi-hop payments like those possible in Lightning. This however is a useful feature in case two parties know that they will only transact once, as opening a virtual channel needs substantially more network communication than performing an one-off multi-hop payment. It would be therefore fruitful to also enable the multi-hop payment technique and allow human users to choose which method to use in each case. Likewise, optimistic cooperative on-chain closing of simple channels could be done just like in Lightning, obviating the need to wait for the revocation timelock to expire and reducing on-chain costs if the counterparty is cooperative.

What is more, any deployment of the protocol has to explicitly handle the issue of tx fees. These include miner fees for on-chain txs and intermediary fees for the parties that own base channels and facilitate opening virtual channels. These fees should take into account the fact that each intermediary has quadratic storage requirements, whereas endpoints only need constant storage, creating an oppor-

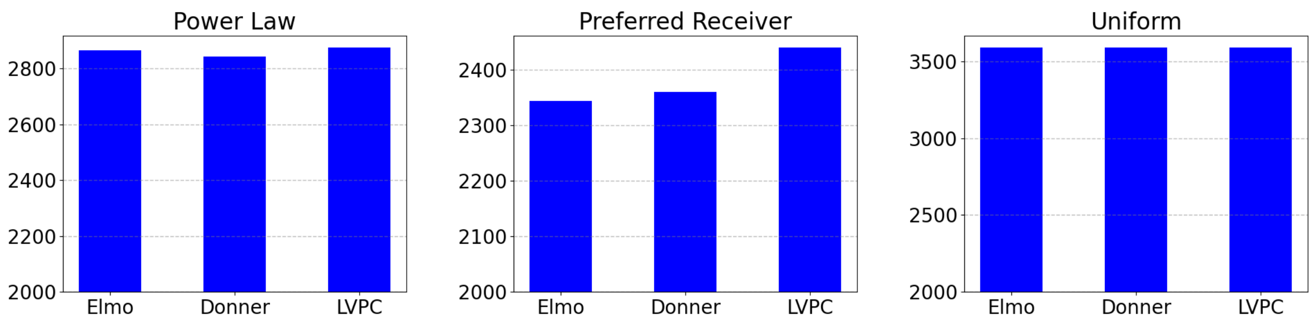

Figure 9: Average per-payment delay (both on- and off-chain) in sec. Less is better.

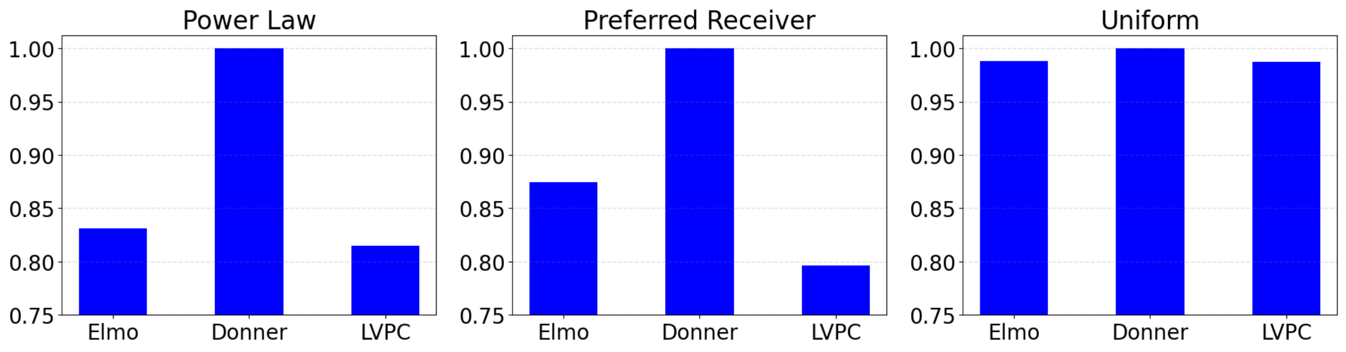

Figure 10: Average per-payment relative fee. Less is better.

tunity for amplification attacks. Additionally, a fee structure that takes into account the opportunity cost of base parties locking collateral for a potentially long time is needed. A straightforward mechanism is for parties to agree on a time-based fee schedule and periodically update their base channels to reflect contingent payments by the endpoints. We leave the relevant incentive analysis as future work.

In order to increase readability and to keep focus on the salient points of the construction, our protocol does not exploit various possible optimisations. These include allowing parties to stay offline for longer [4], and some techniques employed in Lightning that drastically reduce storage requirements, such as storage of per-update secrets in $O(\log n)$ space[6], and other improvements to our novel virtual subprotocol.

As mentioned before, we conjecture that a variadic virtual channel protocol with unlimited lifetime needs each party to store an exponential number of signatures if `ANYPREVOUT` is not available. We leave proof of this as future work. Furthermore, the formal verification of the UC security proof is deferred to such a time when a practical framework for mechanised UC proofs becomes available.

Last but not least, the current analysis gives no privacy guarantees for the protocol, as it does not employ onion packets [19] like Lightning. Furthermore, $\mathcal{G}_{\text{Chan}}$ leaks all messages to the ideal adversary therefore theoretically no privacy is offered at all. Nevertheless, onion packets can be incorporated in the current construction. Intuitively our construction leaks less data than Lightning for the same multi-hop payments, as intermediaries in our case are not notified on each payment, contrary to multi-hop payments in Lightning. Therefore a future extension can improve the privacy of the construction and formally demonstrate exact privacy guarantees.

8. Conclusion

In this work we presented Elmo, a construction for the establishment and optimistic teardown of payment channels without posting transactions on-chain. Such a virtual channel can be opened over a path of base channels of any length, i.e., the constructor is *variadic*.

The base channels themselves can be virtual, therefore our construction is *recursive*. A key performance characteristic of our construction is its optimal round complexity for on-chain channel closing: one transaction is required by any party to turn the virtual channel into a simple one and one more transaction is needed to close it.

We formally described the protocol in the UC setting, provided a suitable ideal functionality and finally proved the indistinguishability of the protocol and functionality, along with the balance security properties that ensure no loss of funds. This is achieved through the use of the `ANYPREVOUT` sighash flag, which is a feature that will in all likelihood be added in the next Bitcoin update.

6. https://github.com/lightning/bolts/blob/master/03-transactions.md#efficient-per-commitment-secret-storage

References

[1] Aumayr, L., Ersoy, O., Erwig, A., Faust, S., Hostáková, K., Maffei, M., Moreno-Sanchez, P., Riahi, S.: Generalized bitcoin-compatible channels. IACR Cryptol. ePrint Arch. p. 476 (2020)

[2] Aumayr, L., Maffei, M., Ersoy, O., Erwig, A., Faust, S., Riahi, S., Hostáková, K., Moreno-Sanchez, P.: Bitcoin-compatible virtual channels. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 901–918 (2021). https://doi.org/10.1109/SP40001.2021.00097

[3] Aumayr, L., Moreno-Sanchez, P., Kate, A., Maffei, M.: Breaking and fixing virtual channels: Domino attack and donner. In: 30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023. The Internet Society (2023), https://www.ndss-symposium.org/ndss-paper/breaking-and-fixing-virtual-channels-domino-attack-and-donner/

[4] Aumayr, L., Thyagarajan, S.A.K., Malavolta, G., Moreno-Sanchez, P., Maffei, M.: Sleepy channels: Bi-directional payment channels without watchtowers. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022. pp. 179–192. ACM (2022). https://doi.org/10.1145/3548606.3559370, https://doi.org/10.1145/3548606.3559370

[5] Avarikioti, G., Kogias, E.K., Wattenhofer, R., Zindros, D.: Brick: Asynchronous payment channels (2020)

[6] Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Timón, J., Wuille, P.: Enabling blockchain innovations with pegged sidechains (2014)

[7] Badertscher, C., Canetti, R., Hesse, J., Tackmann, B., Zikas, V.: Universal composition with global subroutines: Capturing global setup within plain UC. In: Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part III. pp. 1–30 (2020). https://doi.org/10.1007/978-3-030-64381-2_1

[8] Badertscher, C., Gaži, P., Kiayias, A., Russell, A., Zikas, V.: Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 913–930. ACM (2018)

[9] Badertscher, C., Hesse, J., Zikas, V.: On the (ir)replaceability of global setups, or how (not) to use a global ledger. In: Nissim, K., Waters, B. (eds.) Theory of Cryptography. pp. 626–657. Springer International Publishing, Cham (2021)

[10] Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: A composable treatment. In: Annual International Cryptology Conference. pp. 324–356. Springer (2017)

[11] Broder, A.Z., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., Wiener, J.L.: Graph structure in the web. Comput. Networks **33**(1-6), 309–320 (2000). https://doi.org/10.1016/S1389-1286(00)00083-9, https://doi.org/10.1016/S1389-1286(00)00083-9

[12] Burchert, C., Decker, C., Wattenhofer, R.: Scalable funding of bitcoin micropayment channel networks. In: The Royal Society (2018). https://doi.org/10.1098/rsos.180089

[13] Buterin, V.: On-chain scaling to potentially 500 tx/sec through mass tx validation. https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477

[14] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Paper 2000/067 (2000), https://eprint.iacr.org/2000/067, https://eprint.iacr.org/2000/067

[15] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA. pp. 136–145 (2001). https://doi.org/10.1109/SFCS.2001.959888

[16] Chakravarty, M.M.T., Coretti, S., Fitzi, M., Gazi, P., Kant, P., Kiayias, A., Russell, A.: Hydra: Fast isomorphic state channels. Cryptology ePrint Archive, 2020/299

[17] Chakravarty, M.M.T., Kireev, R., MacKenzie, K., McHale, V., Müller, J., Nemish, A., Nester, C., Peyton Jones, M., Thompson, S., Valentine, R., Wadler, P.: Functional blockchain contracts (2019), https://iohk.io/en/research/library/papers/functional-blockchain-contracts/

[18] Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E.G., et al.: On scaling decentralized blockchains. In: Financial Cryptography and Data Security. pp. 106–125. Springer (2016)

[19] Danezis, G., Goldberg, I.: Sphinx: A compact and provably secure mix format. In: Security and Privacy, 2009 30th IEEE Symposium on. pp. 269–282. IEEE (2009)

[20] Decker, C., Russell, R., Osuntokun, O.: eltoo: A simple layer2 protocol for bitcoin

[21] Decker, C., Wattenhofer, R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In: Symposium on Self-Stabilizing Systems. pp. 3–18. Springer (2015)

[22] Dong, M., Liang, Q., Li, X., Liu, J.: Celer network: Bring internet scale to every blockchain (2018)

[23] Dunford, R., Su, Q., Tamang, E.: The pareto principle. The Plymouth Student Scientist **7**, 140–148 (2014). https://doi.org/10.4135/9781412950596.n394, http://hdl.handle.net/10026.1/14054

[24] Dziembowski, S., Eckey, L., Faust, S., Hesse, J., Hostáková, K.: Multi-party virtual state channels. In: Ishai, Y., Rijmen, V. (eds.) International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT. pp. 625–656 (2019). https://doi.org/10.1007/978-3-030-17653-2_21

[25] Dziembowski, S., Eckey, L., Faust, S., Malinowski, D.: Perun: Virtual payment hubs over cryptocurrencies. In: IEEE Symposium on Security and Privacy (SP). pp. 344–361. IEEE Computer Society, Los Alamitos, CA, USA (May 2019). https://doi.org/10.1109/SP.2019.00020

[26] Dziembowski, S., Fabiański, G., Faust, S., Riahi, S.: Lower bounds for off-chain protocols: Exploring the limits of plasma. Cryptology ePrint Archive, Report 2020/175

[27] Dziembowski, S., Faust, S., Hostáková, K.: General state channel networks. In: Computer and Communications Security, CCS. pp. 949–966 (2018). https://doi.org/10.1145/3243734.3243856

[28] Egger, C., Moreno-Sanchez, P., Maffei, M.: Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks. In: Conference on Computer and Communications Security, SIGSAC. pp. 801–815. CCS ’19 (2019). https://doi.org/10.1145/3319535.3345666

[29] Gaži, P., Kiayias, A., Zindros, D.: Proof-of-stake sidechains. In: Symposium on Security and Privacy, SP. pp. 677–694 (2019). https://doi.org/10.1109/SP.2019.00040

[30] Green, M., Miers, I.: Bolt: Anonymous payment channels for decentralized currencies. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. p. 473–489. CCS ’17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134093

[31] Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: Sok: Layer-two blockchain protocols. In: Financial Cryptography and Data Security FC. pp. 201–226 (2020). https://doi.org/10.1007/978-3-030-51280-4_12

[32] Harris, J., Zohar, A.: Flood & loot: A systemic attack on the lightning network. In: Conference on Advances in Financial Technologies. pp. 202–213. AFT (2020). https://doi.org/10.1145/3419614.3423248

[33] Jourenko, M., Larangeira, M., Tanaka, K.: Lightweight virtual payment channels. In: Cryptology and Network Security. pp. 365–384 (2020)

[34] Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings. pp. 477–498 (2013). https://doi.org/10.1007/978-3-642-36594-2_27

[35] Khalil, R., Gervais, A.: Revive: Rebalancing off-blockchain payment networks. In: Conference on Computer and Communications Security, CCS. pp. 439–453 (2017). https://doi.org/10.1145/3133956.3134033

[36] Kiayias, A., Litos, O.S.T.: A composable security treatment of the lightning network. In: Computer Security Foundations Symposium, CSF. pp. 334–349 (2020). https://doi.org/10.1109/CSF49147.2020.00031

[37] Kiayias, A., Zindros, D.: Proof-of-work sidechains. In: Workshop on Trusted Smart Contracts. pp. 21–34 (2019). https://doi.org/10.1007/978-3-030-43725-1_3

[38] Konstantopoulos, G.: Plasma cash: Towards more efficient plasma constructions (2019)

[39] Lee, J., Kim, S., Park, S., Moon, S.M.: Routee: A secure payment network routing hub using trusted execution environments (2020)

[40] Liao, J., Zhang, F., Sun, W., Shi, W.: Speedster: An efficient multi-party state channel via enclaves. In: Asia Conference on Computer and Communications Security, ASIACCS. pp. 637–651 (2022). https://doi.org/10.1145/3488932.3523259

[41] Lind, J., Eyal, I., Pietzuch, P.R., Sirer, E.G.: Teechan: Payment channels using trusted execution environments. CoRR **abs/1612.07766** (2016)

[42] Lind, J., Naor, O., Eyal, I., Kelbert, F., Sirer, E.G., Pietzuch, P.: Teechain: A secure payment network with asynchronous blockchain access. In: Symposium on Operating Systems Principles. pp. 63–79. SOSP '19 (2019). https://doi.org/10.1145/3341301.3359627

[43] Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Anonymous multi-hop locks for blockchain scalability and interoperability. In: Network and Distributed System Security Symposium, NDSS (2019)

[44] Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple schnorr multi-signatures with applications to bitcoin. Des. Codes Cryptogr. **87**(9), 2139–2164 (2019). https://doi.org/10.1007/s10623-019-00608-x, https://doi.org/10.1007/s10623-019-00608-x

[45] Milgram, S.: The small world problem. Psychology Today **1**, 61–67 (May 1967)

[46] Miller, A., Bentov, I., Kumaresan, R., Cordi, C., McCorry, P.: Sprites and state channels: Payment networks that go faster than lightning (2017), arXiv:1702.05812

[47] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)

[48] Nick, J., Ruffing, T., Seurin, Y.: Musig2: Simple two-round schnorr multi-signatures. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12825, pp. 189–221. Springer (2021). https://doi.org/10.1007/978-3-030-84242-0_8, https://doi.org/10.1007/978-3-030-84242-0_8

[49] Optimism: Optimistic rollup overview. https://github.com/ethereum-optimism/optimistic-specs/blob/0e9673af0f2cafd89ac7d6c0e5d8bed7c67b74ca/overview.md

[50] Poon, J., Buterin, V.: Plasma: Scalable autonomous smart contracts

[51] Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf (January 2016)

[52] Powers, D.M.W.: Applications and explanations of Zipf's law. In: New Methods in Language Processing and Computational Natural Language Learning (1998)

[53] Prihodko, P., Zhigulin, S., Sahno, M., Ostrovskiy, A., Osuntokun, O.: Flare: An approach to routing in lightning network (2016)

[54] Sivaraman, V., Venkatakrishnan, S.B., Alizadeh, M., Fanti, G.C., Viswanath, P.: Routing cryptocurrency with the spider network. CoRR **abs/1809.05088** (2018)

[55] Spilman, J.: Anti dos for tx replacement. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html (April 2013)

[56] Wood, G.: Ethereum: A secure decentralised generalised transaction ledger

[57] Zhao, L., Shuang, H., Xu, S., Huang, W., Cui, R., Bettadpur, P., Lie, D.: Sok: Hardware security support for trustworthy execution (2019)

Appendix A. Further Related Work

Various attacks have been identified against LN. The wormhole attack [43] against LN allows colluding parties in a multi-hop payment to steal the fees of the intermediaries between them and Flood & Loot attacks [32] analyses an attack in which too many channels are forced to close in a short amount of time, harming blockchain liveness and enabling a malicious party to steal off-chain funds.

To the best of our knowledge, no formal treatment of the privacy of LN exists. Nevertheless, it intuitively improves upon the privacy of on-chain Bitcoin transactions, as LN payments do not leave a permanent record: only intermediaries of each payment are informed. It can be argued that Elmo further improves privacy, as payments are hidden from the intermediaries of a virtual channel.

Payment routing [54], [53], [39] is another research area that aims to improve network efficiency without sacrificing privacy. Actively rebalancing channels [35] can further increase network efficiency by reducing unavailable routes due to lack of well-balanced funds.

Bolt [30] constructs privacy-preserving payment channels enabling both direct payments and payments with a single untrusted intermediary. Sprites [46] leverages the scripting language of Ethereum to decrease the time collateral is locked compared to LN.

State channels are a generalisation of payment channels, which enable off-chain execution of any smart contract supported by the underlying blockchain, not just payments. Generalized Bitcoin-Compatible Channels [1] enable the creation of state channels on Bitcoin, extending channel functionality from simple payments to arbitrary Bitcoin scripts. Since Elmo only pertains to payment, not state, channels, we choose not to build it on top of [1]. State channels can also be extended to more than two parties [40], [24].

BDW [12] shows how pairwise channels over Bitcoin can be funded with no on-chain transactions by allowing parties to form groups that can pool their funds together off-chain and then use those funds to open channels. Such proposals are complementary to virtual channels and, depending on the use case, could be more efficient. In comparison to Elmo, BDW is less flexible: coins in a BDW pool can only be exchanged with members of that pool. ACMU [28] allows for multi-path atomic payments with reduced collateral, enabling new applications such as crowdfunding conditional on reaching a funding target.

TEE-based [57] solutions [41], [42], [40], [39] improve the throughput and efficiency of PCNs by an order of magnitude or more, at the cost of having to trust TEEs. Brick [5] uses a partially trusted committee to extend PCNs to fully asynchronous networks.

Donner [3] is technically insecure since any state update to a base channel invalidates the corresponding tx^r. There is a straightforward fix, which however adds an overhead to each payment over a base channel: On every payment, the two base channel parties must update their tx^r to spend the α output of the new state. Potential intermediaries must consider this overhead and possibly increase the fees they require from the endpoints. This per-payment overhead can be avoided by using `ANYPREVOUT` in the α output.

Appendix B. Functionality $\mathcal{G}_{\text{Chan}}$

Functionality $\mathcal{G}_{\text{Chan}}$ – general message handling rules

- On receiving input (`msg`) by $\mathcal{E}$ addressed to $P \in \{Alice, Bob\}$, handle it according to the corresponding rule in Fig. 12, 13, 14, 15 or 16 (if any) and subsequently send (RELAY, `msg`, P, $\mathcal{E}$, input) to $\mathcal{A}$.
- On receiving (`msg`) by party R addressed to $P \in \{Alice, Bob\}$ by means of `mode` $\in \{\text{output}, \text{network}\}$, handle it according to the corresponding rule in Fig. 12, 13, 14, 15 or 16 (if any) and subsequently send (RELAY, `msg`, P, $\mathcal{E}$, `mode`) to $\mathcal{A}$. // all messages are relayed to $\mathcal{A}$
- On receiving (RELAY, `msg`, P, R, `mode`) by $\mathcal{A}$ (`mode` $\in \{\text{input}, \text{output}, \text{network}\}$, $P \in \{Alice, Bob\}$), relay `msg` to R as P by means of `mode`. // $\mathcal{A}$ fully controls outgoing messages by $\mathcal{G}_{\text{Chan}}$
- On receiving (INFO, `msg`) by $\mathcal{A}$, handle (`msg`) according to the corresponding rule in Fig. 12, 13, 14, 15 or 16 (if any). After handling the message or after an "ensure" fails, send (HANDLED, `msg`) to $\mathcal{A}$. // (INFO, `msg`) messages by $\mathcal{S}$ always return control to $\mathcal{S}$ without any side-effect to any other ITI, except if $\mathcal{G}_{\text{Chan}}$ halts
- $\mathcal{G}_{\text{Chan}}$ keeps track of two state machines, one for each of $Alice$, Bob. If there are more than one suitable rules for a particular message, or if a rule matches the message for both parties, then both rule versions are executed. // the two rules act on different state machines, so the order of execution does not matter

Figure 11

Note that in UCGS [7], just like in UC, every message to an ITI may arrive via one of three channels: input, output and network. In the session of interest, input messages come from the environment $\mathcal{E}$ in the real world, whereas in the ideal world each input message comes from the corresponding dummy party, which forwards it as received by $\mathcal{E}$. Outputs may be received from any subroutine (local or global). This means that the "sender field" of inputs and outputs cannot be tampered with by $\mathcal{E}$ or $\mathcal{A}$. Network messages only come from $\mathcal{A}$; they may have been sent from any machine but are relayed (and possibly delayed, reordered, modified or even dropped) by $\mathcal{A}$. Therefore, in contrast to inputs and outputs, network messages may have a tampered "sender field".

Functionality $\mathcal{G}_{\text{Chan}}$ – open state machine

```
P ∈ {Alice, Bob}
1: On first activation: // before handing the message
2:     $pk_P \leftarrow \bot$; $balance_P \leftarrow 0$; $State_P \leftarrow$ UNINIT
3:     $enabler_P \leftarrow \bot$ // if we are a virtual channel, the ITI of P's base channel
4:     $host_P \leftarrow \bot$ // if we are a virtual channel, the ITI of the common host of this channel and P's base channel

5: On (BECAME CORRUPTED OR NEGLIGENT, P) by $\mathcal{A}$ or on output (ENABLER USED REVOCATION) by $host_P$ when in any state:
6:     $State_P \leftarrow$ IGNORED

7: On (INIT, pk) by P when $State_P =$ UNINIT:
8:     $pk_P \leftarrow pk$
9:     $State_P \leftarrow$ INIT

10: On (OPEN, x, "ledger", ...) by Alice when $State_A =$ INIT:
11:     store x
12:     $State_A \leftarrow$ TENTATIVE BASE OPEN

13: On (BASE OPEN) by $\mathcal{A}$ when $State_A =$ TENTATIVE BASE OPEN:
14:     $balance_A \leftarrow x$
15:     $layer_A \leftarrow 0$
16:     $State_A \leftarrow$ OPEN

17: On (BASE OPEN) by $\mathcal{A}$ when $State_B =$ INIT:
18:     $layer_B \leftarrow 0$
19:     $State_B \leftarrow$ OPEN

20: On (OPEN, x, hops $\neq$ "ledger", ...) by Alice when $State_A =$ INIT:
21:     store x
22:     $enabler_A \leftarrow$ hops[0].left
23:     add $enabler_A$ to Alice's kindred parties
24:     $State_A \leftarrow$ PENDING VIRTUAL OPEN

25: On output (FUNDED, host, ...) to Alice by $enabler_A$ when $State_A =$ PENDING VIRTUAL OPEN:
26:     $host_A \leftarrow$ host[0].left
27:     $State_A \leftarrow$ TENTATIVE VIRTUAL OPEN

28: On output (FUNDED, host, ...) to Bob by ITI $R \in \{\mathcal{G}_{\text{Chan}}, \text{LN}\}$ when $State_B =$ INIT:
29:     $enabler_B \leftarrow R$
30:     add $enabler_B$ to Bob's kindred parties
31:     $host_B \leftarrow$ host
32:     $State_B \leftarrow$ TENTATIVE VIRTUAL OPEN

33: On (VIRTUAL OPEN) by $\mathcal{A}$ when
```

```
    $State_P$ = TENTATIVE VIRTUAL OPEN:
34:     if $P = Alice$ then $\texttt{balance}_P \leftarrow x$
35:     $\texttt{layer}_P \leftarrow 0$
36:     $State_P \leftarrow$ OPEN
```

Figure 12

Functionality $\mathcal{G}_{\mathrm{Chan}}$ – payment state machine

```
$P \in \{Alice, Bob\}$
 1: On (PAY, $x$) by $P$ when $State_P$ = OPEN: // $P$ pays $\bar{P}$
 2:     store $x$
 3:     $State_P \leftarrow$ TENTATIVE PAY

 4: On (PAY) by $\mathcal{A}$ when $State_P$ = TENTATIVE PAY: // $P$
    pays $\bar{P}$
 5:     $State_P \leftarrow$ (SYNC PAY, $x$)

 6: On (GET PAID, $y$) by $P$ when $State_P$ = OPEN: // $\bar{P}$ pays
    $P$
 7:     store $y$
 8:     $State_P \leftarrow$ TENTATIVE GET PAID

 9: On (PAY) by $\mathcal{A}$ when $State_P$ = TENTATIVE GET PAID: //
    $\bar{P}$ pays $P$
10:     $State_P \leftarrow$ (SYNC GET PAID, $x$)

11: When $State_P$ = (SYNC PAY, $x$):
12:     if $State_{\bar{P}} \in \{$IGNORED, (SYNC GET PAID, $x$)$\}$ then
13:         $\texttt{balance}_P \leftarrow \texttt{balance}_P - x$
14:         // if $\bar{P}$ honest, this state transition happens
    simultaneously with l. 21
15:         $State_P \leftarrow$ OPEN
16:     end if

17: When $State_P$ = (SYNC GET PAID, $x$):
18:     if $State_{\bar{P}} \in \{$IGNORED, (SYNC PAY, $x$)$\}$ then
19:         $\texttt{balance}_P \leftarrow \texttt{balance}_P + x$
20:         // if $\bar{P}$ honest, this state transition happens
    simultaneously with l. 15
21:         $State_P \leftarrow$ OPEN
22:     end if
```

Figure 13

Functionality $\mathcal{G}_{\mathrm{Chan}}$ – funding state machine

```
$P \in \{Alice, Bob\}$
 1: On input (FUND ME, $x$, ...) by ITI $R \in \{\mathcal{G}_{\mathrm{Chan}}$, LN$\}$
    when $State_P$ = OPEN:
 2:     store $x$
 3:     add $R$ to $P$'s kindred parties
 4:     $State_P \leftarrow$ PENDING FUND
```

```
 5: When $State_P$ = PENDING FUND:
 6:     if we intercept the command "define new VIRT ITI
    host" by $\mathcal{A}$, routed through $P$ then
 7:         store host
 8:         $State_P \leftarrow$ TENTATIVE FUND
 9:         continue executing $\mathcal{A}$'s command
10:     end if

11: On (FUND) by $\mathcal{A}$ when $State_P$ = TENTATIVE FUND:
12:     $State_P \leftarrow$ SYNC FUND

13: When $State_P$ = OPEN:
14:     if we intercept the command "define new VIRT ITI
    host" by $\mathcal{A}$, routed through $P$ then
15:         store host
16:         $State_P \leftarrow$ TENTATIVE HELP FUND
17:         continue executing $\mathcal{A}$'s command
18:     end if
19:     if we receive a RELAY message with msg = (INIT,
    ..., fundee) addressed from $P$ by $\mathcal{A}$ then
20:         add fundee to $P$'s kindred parties
21:         continue executing $\mathcal{A}$'s command
22:     end if

23: On (FUND) by $\mathcal{A}$ when $State_P$ = TENTATIVE HELP FUND:
24:     $State_P \leftarrow$ SYNC HELP FUND

25: When $State_P$ = SYNC FUND:
26:     if $State_{\bar{P}} \in \{$IGNORED, SYNC HELP FUND$\}$ then
27:         $\texttt{balance}_P \leftarrow \texttt{balance}_P - x$
28:         $\texttt{host}_P \leftarrow \texttt{host}$
29:         // if $\bar{P}$ honest, this state transition happens
    simultaneously with l. 38
30:         $\texttt{layer}_P \leftarrow \texttt{layer}_P + 1$
31:         $State_P \leftarrow$ OPEN
32:     end if

33: When $State_P$ = SYNC HELP FUND:
34:     if $State_{\bar{P}} \in \{$IGNORED, SYNC FUND$\}$ then
35:         $\texttt{host}_P \leftarrow \texttt{host}$
36:         // if $\bar{P}$ honest, this state transition happens
    simultaneously with l. 31
37:         $\texttt{layer}_P \leftarrow \texttt{layer}_P + 1$
38:         $State_P \leftarrow$ OPEN
39:     end if
```

Figure 14

Functionality $\mathcal{G}_{\mathrm{Chan}}$ – force close state machine

```
$P \in \{Alice, Bob\}$
 1: On (FORCECLOSE) by $P$ when $State_P$ = OPEN:
 2:     $State_P \leftarrow$ CLOSING

 3: On input (BALANCE) by $R$ addressed to $P$ where $R$ is
    kindred with $P$:
 4:     if $State_P \notin \{$UNINIT, INIT, PENDING VIRTUAL OPEN,
    TENTATIVE VIRTUAL OPEN, TENTATIVE BASE OPEN,
        IGNORED, CLOSED} then
5:          reply (MY BALANCE, $\mathtt{balance}_P$, $pk_P$, $\mathtt{balance}_{\bar{P}}$, $pk_{\bar{P}}$)
6:      else
7:          reply (MY BALANCE, 0, $pk_P$, 0, $pk_{\bar{P}}$)
8:      end if

9:  On (FORCECLOSE, $P$) by $\mathcal{A}$ when $State_P \notin$ {UNINIT, INIT, PENDING VIRTUAL OPEN, TENTATIVE VIRTUAL OPEN, TENTATIVE BASE OPEN, IGNORED}:
10:     input (READ) to $\mathcal{G}_{\text{Ledger}}$ as $P$ and assign ouput to $\Sigma$
11:     $\mathtt{coins} \leftarrow$ sum of values of outputs exclusively spendable or spent by $pk_P$ in $\Sigma$
12:     $\mathtt{balance} \leftarrow \mathtt{balance}_P$
13:     for all $P$'s kindred parties $R$ do
14:         input (BALANCE) to $R$ as $P$ and extract $\mathtt{balance}_R$, $pk_R$ from response
15:         $\mathtt{balance} \leftarrow \mathtt{balance} + \mathtt{balance}_R$
16:         $\mathtt{coins} \leftarrow \mathtt{coins}$ + sum of values of outputs exclusively spendable or spent by $pk_R$ in $\Sigma$
17:     end for
18:     if $\mathtt{coins} \geq \mathtt{balance}$ then
19:         $State_P \leftarrow$ CLOSED
20:     else // balance security is broken
21:         halt
22:     end if
```

Figure 15

Functionality $\mathcal{G}_{\text{Chan}}$ – cooperative close state machine

```
$P \in \{Alice, Bob\}$
1:  On (COOP CLOSING, $P$, $x$) by $\mathcal{A}$ when $State_P$ = OPEN:
2:      store $x$
3:      $State_P \leftarrow$ COOP CLOSING

4:  On (COOP CLOSED, $P$) by $\mathcal{A}$ when $State_P$ = COOP CLOSING:
5:      if $\mathtt{layer}_P = 0$ then // $P$'s channel, which is virtual, is cooperatively closed
6:          $State_P \leftarrow$ COOP CLOSED
7:      else // the virtual channel for which $P$'s channel is base is cooperatively closed
8:          $\mathtt{layer}_P \leftarrow \mathtt{layer}_P - 1$
9:          $\mathtt{balance}_P \leftarrow \mathtt{balance}_P + x$
10:         $State_P \leftarrow$ OPEN
11:     end if
```

Figure 16

Appendix C. Liveness & Replaceability

Proposition 2. Consider a synchronised honest party that submits a transaction $\mathtt{tx}$ to the ledger functionality [8] by the time the block indexed by h is added to $\mathtt{state}$ in its view. Then $\mathtt{tx}$ is guaranteed to be included in the block range $[h+1, h+s]$, where $s = (2+q)\mathtt{windowSize}$ and $q = \lceil(\mathtt{maxTime}_{\mathtt{window}} + \frac{\mathtt{Delay}}{2})/\mathtt{minTime}_{\mathtt{window}}\rceil$.

Proof: Consider τ_h^U to be the round that a party U becomes aware of the h-th block in the $\mathtt{state}$. It follows that $\tau_h \leq \tau_h^U$ where τ_h is the round block h enters $\mathtt{state}$. Note that by time $\tau_h + \mathtt{maxTime}_{\mathtt{window}}$ another $\mathtt{windowSize}$ blocks are added to $\mathtt{state}$ and thus $\tau_h^U \leq \tau_h + \mathtt{maxTime}_{\mathtt{window}}$.

Suppose U submits the transaction $\mathtt{tx}$ to the ledger at time τ_h^U. Observe that as long as $\tau_h + \mathtt{maxTime}_{\mathtt{window}}$ is $\mathtt{Delay}/2$ before the time that block with index $h + t - 2\mathtt{windowSize}$ enters $\mathtt{state}$, then $\mathtt{tx}$ is guaranteed to enter the $\mathtt{state}$ in a block with index up to $h + t$ where since $\mathtt{advBlcks}_{\mathtt{window}} < \mathtt{windowSize}$. It follows we need $\tau_h + \mathtt{maxTime}_{\mathtt{window}} < \tau_{h+t-2\mathtt{windowSize}} - \frac{\mathtt{Delay}}{2}$. Let $r = \lceil(\mathtt{maxTime}_{\mathtt{window}} + \frac{\mathtt{Delay}}{2})/\mathtt{minTime}_{\mathtt{window}}\rceil$. Recall that in a period of $\mathtt{minTime}_{\mathtt{window}}$ rounds at most $\mathtt{windowSize}$ blocks enter $\mathtt{state}$. As a result $r \cdot \mathtt{windowSize}$ blocks require at least $r \cdot \mathtt{minTime}_{\mathtt{window}} \geq \mathtt{maxTime}_{\mathtt{window}} + \frac{\mathtt{Delay}}{2}$ rounds. We deduce that if $t \geq (2+r)\mathtt{windowSize}$ the inequality follows. □

Since we use a global setup, proving UC-emulation is not enough. We further need to prove that all ideal global subroutines are *replaceable*, i.e., they can be replaced with their real counterparts. This guarantees that a real deployment will offer the same security guarantees as its idealized description.

For any $i \in [n]$, the individual global subroutines $\mathcal{G}_{\text{Ledger}}, \mathcal{G}^1_{\text{Chan}}, \dots, \mathcal{G}^i_{\text{Chan}}$ can be merged (as per Def. 4.1 of [9]) into the "global setup" $\mathcal{G}^i$. Likewise, the realisation Π_{Ledger} of $\mathcal{G}_{\text{Ledger}}$ [10] and $\Pi^1_{\text{Chan}}, \dots, \Pi^i_{\text{Chan}}$ can be merged into Π^i.

Lemma 3. For all $i \in [n]$, Π^i UC-emulates $\mathcal{G}^i$.

Proof: [Proof of Lemma 3] The following facts hold (note the inversion of the order of indices compared to the formulation of Theorem 4.3): $\mathcal{G}_{\text{Ledger}}$ and Π_{Ledger} are *subroutine respecting* (Def. A.6 [9]) as they do not accept/pass inputs or outputs from/to parties outside their session (this can be verified by inspection). $\mathcal{G}^1_{\text{Chan}}$ and Π^1_{Chan} are $\mathcal{G}_{\text{Ledger}}$- and Π_{Ledger}-subroutine respecting respectively (Def. 2.3 [9]), as they they do not accept/pass inputs or outputs from/to parties outside their session apart from $\mathcal{G}_{\text{Ledger}}$ and Π_{Ledger} respectively. For any $j \in [i] \setminus \{1\}$, $\mathcal{G}^j_{\text{Chan}}$ and Π^j_{Chan} are $\mathcal{G}^{j-1}$- and Π^{j-1}-subroutine respecting respectively, as they they do not accept/pass inputs or outputs from/to parties outside their session apart from $\mathcal{G}^{j-1}$ and Π^{j-1} respectively. Theorem 4.3 of [9] then implies the required result. □

Theorem 4 (Full Replacement). For all $i \in [n]$, the ideal global setup $\mathcal{G}^i$ can be replaced with Π^i.

Proof: Our simulator $\mathcal{S}$ is $\mathcal{G}$*-agnostic* (Def. 3.4 of [9]) as $\mathcal{S}$ is a relay between parties and the environment $\mathcal{E}$. Thus Theorem 3.5 of [9], full replacement, applies inductively to $\mathcal{G}^i$ and Π^i. □