# Non-Atomic Payment Splitting in Channel Networks

Stefan Dziembowski and Paweł Kędzior
University of Warsaw

***Abstract*—*Off-chain channel networks* are one of the most promising technologies for dealing with blockchain scalability and delayed finality issues. Parties that are connected within such networks can send coins to each other without interacting with the blockchain. Moreover, these payments can be "routed" over the network. Thanks to this, even the parties that do not have a channel in common can perform payments between each other with the help of intermediaries.**

**In this paper, we introduce a new notion that we call *Non-Atomic Payment Splitting (NAPS) protocols* that allow the intermediaries in the network to split the payments recursively into several sub-payments in such a way that the payment can be successful "partially" (i.e. not all the requested amount may be transferred). This is in contrast with the existing splitting techniques that are "atomic" in the sense that they did not allow such partial payments (we compare the "atomic" and "non-atomic" approach in the paper). We define NAPS formally, and then present a protocol, that we call "ETHNA", that satisfies this definition. ETHNA is based on very simple and efficient cryptographic tools, and in particular does not use any expensive cryptographic primitives. We implement a simple variant of ETHNA in Solidity and provide some benchmarks. We also report on some experiments with routing using ETHNA.**

## I. INTRODUCTION

Blockchain technology [26] allows a large group of parties to reach consensus about contents of an (immutable) ledger, typically containing a list of transactions. In blockchain's initial applications these transactions were simply describing transfers of *coins* between the parties. One of the very promising extensions of the original Bitcoin ledger, are blockchains that allow to register and execute the so-called *smart contracts* (or simply "contracts"), i.e., formal agreements between the parties, written down in a programming language and having financial consequences (for more on this topic see, e.g., [6, 11, 21]). Probably the best-known example of such a system is *Ethereum* [34]. One of the main limitations of several blockchain-based systems is delayed finality, lack of scalability, and non-trivial transaction fees. For example, in Bitcoin it takes at least around 10 minutes to confirm a transaction, at most 7 transactions per second can be processed, and the average transaction fee is currently typically over 1 USD.

*Off-chain channels* [7, 30, 31] are a powerful approach for dealing with these issues. The simplest example of this technology are the so-called "*payment* channels". Informally, such a channel between Alice and Bob is an object in which both parties have some coins. A channel has a corresponding smart contract on the blockchain that can be used for resolving conflicts between the parties. The parties *open* a channel by depositing some coins in it. They can later change the *balance* of the channel (i.e. information on how the channel's coins are distributed between Alice and Bob, respectively) just by exchanging messages, and without interacting with the blockchain The channel can be *closed* by Alice or Bob, in which case the last channel's balance is used to determine how many coins are transferred to each of them. Since updates do not require blockchain participation (they are done "off-chain"), each individual update is immediate (its time is determined by the network speed) and at essentially no cost. The only operations that involve blockchain are: "opening" and "closing" the channel. Hence, this approach also significantly improves scalability. All these advantages hold only if Alice and Bob are cooperating. In the "pessimistic" case (when one of them is malicious) there are no benefits of using this technology, and the only thing that is guaranteed is that the honest party does not loose her coins. This is ok, since in practice, it is expected that in a vast majority of cases the parties are cooperating (i.e. "behaving optimistically"). We provide more background on the off-chain channels in the next section. As we explain there, channels can form *networks* which can serve for sending coins between the parties that do not have a channel between each other. The main contribution of this work is a novel algorithm for sending such payments.

## II. BACKGROUND AND OUR CONTRIBUTION

In order to explain our contribution we need to provide an introduction to channel networks. This is done in Sec. II-A. Readers familiar with this topic can go quickly over it, just paying attention to some terminology and notation that we use(in particular: "pushing", "acknowledging" payments, "cash functions", "$n$¢", "$P \multimap\!\!\circ\; P'$"). We then outline our contribution in Sec. II-B. In this informal description we assume that the maximal blockchain reaction time is 1 hour (we often write "$h$" for an hour).

### A. Introduction to channel and their networks

As mentioned above, a payment channel is *opened* when Alice and Bob deploy a smart contract on the ledger, and deposit some number of coins (say: $x$, and $y$, respectively) into it. The initial *balance* of this channel is: "$x$¢ in Alice's account, $y$¢ in Bob's account" (or $[\text{Alice} \mapsto x, \text{Bob} \mapsto y]$ for short). We model amounts of coins as non-negative integers, and write "$n$¢" to denote $n$ coins. This balance can be *updated* (to some new balance $[\text{Alice} \mapsto x', \text{Bob} \mapsto y']$, such that $x' + y' = x + y$) by just exchanging messages between the parties. The corresponding smart contract guarantees that each party can at any time *close* the channel and get the money that correspond to her latest balance. Only the opening and closing operations require interaction with the blockchain. For more on how it is done see, e.g., [10]).

Now, suppose we are given a set of parties $P_1, \ldots, P_n$ and channels between some of them. These channels naturally form an (undirected) *channel graph*, which is a tuple

$\mathcal{G} = (\mathcal{P}, \mathcal{E}, \Gamma)$ with the set of vertices $\mathcal{P}$ equal to $\{P_1, \ldots, P_n\}$ and set $\mathcal{E}$ of edges being a family of two-element subsets of $\mathcal{P}$. The elements of $\mathcal{P}$ will be typically denoted as "$P_i \multimapboth P_j$" (instead of $\{P_i, P_j\}$). Every $P_i \multimapboth P_j$ represents a channel between $P_i$ and $P_j$, and the *cash function* $\Gamma$ determines the amount of coins available for the parties in every channel. More precisely, every $\Gamma(P_i \multimapboth P_j)$ is a function $f$ of a type $f : \{P_i, P_j\} \to \mathbb{Z}_{\geq 0}$. We will often write $\Gamma^{P_i \multimapboth P_j}$ to denote this function. The value $\Gamma^{P_i \multimapboth P_j}(P)$ denotes the amount of coins that $P$ has in her *account* in channel $P_i \multimapboth P_j$. A *path* (in $\mathcal{G}$) is a sequence $P_{i_1} \rightarrow \cdots \rightarrow P_{i_t}$ such that for every $j$ we have $P_{i_j} \multimapboth P_{i_{j+1}} \in \mathcal{E}$. In this paper, for the sake of simplicity, we assume that (a) the channel system is deployed with some initial value of $\Gamma_0$, which evolves over time, resulting in functions $\Gamma_1, \Gamma_2, \ldots$, (b) once a channel system is established, no new channels are created (i.e., $\mathcal{E}$ remains fixed), and (c) no coins are added to the the existing channels, i.e., the total amount of coins available in every channel $e = P_i \multimapboth P_j$ never exceeds the total amount available in it initially.

Channel graphs can serve for secure payment sending. Let us recall how this works in the most popular payment channel networks, such as *Lightning* or *Raiden*. Our description is very high-level (for the details, see, e.g., [30]). Consider the following example: we have three parties: $P_1, P_2$, and $P_3$ and two channels: $P_1 \multimapboth P_2$ and $P_2 \multimapboth P_3$ between them. Now, suppose the *sender* $P_1$ wants to send $v$¢ to the *receiver* $P_3$ over the path $P_1 \rightarrow P_2 \rightarrow P_3$, with $P_2$ being an *intermediary* that *routes* these coins. This is done as follows. First, party $P_1$ asks $P_2$ to forward $v$¢ in the direction of $P_3$ (we call such a request *pushing* coins from $P_1$ to $P_2$). The proof that $P_3$ received these coins has to be presented by $P_2$ within 2 hours (denote this proof with $\pi$ — we will discuss how $\pi$ looks like in a moment). If $P_2$ manages to do it by this deadline, then she gets these coins in her account in the channel $P_1 \multimapboth P_2$. To guarantee that this will happen, $P_1$ initially blocks these coins in the channel $P_1 \multimapboth P_2$. These coins can be claimed back by $P_1$ if the 2 hours have passed, and $P_2$ did not claim them. In a similar way, $P_2$ pushes these coins to $P_3$, i.e., she offers $P_3$ to claim (by providing proof $\pi$ within 1 hour) 6¢ in the channel channel $P_3 \multimapboth P_4$. Now, suppose that party $P_3$ claims her $v$¢ in channel $P_2 \multimapboth P_3$. This can only be done by providing a proof $\pi$ that she received these coins. We call this process *acknowledging* the payment. Party $P_2$ and $P_2$ can now claim her coins in channel $P_1 \multimapboth P_2$ by submitting an acknowledgment containing the proof $\pi$.

In the above example the amount of coins that can be pushed via a channel $P_i \multimapboth P_{i+1}$ is upper-bounded by the amount of coins that $P_i$ has in this channel. Therefore the maximal amount of coins that can be pushed over path $P_1 \rightarrow P_2 \rightarrow P_3$ is equal to the minimum of these values. We will call this value the *capacity* of a given path.

On the technical level, in the Lightning network the proof $\pi$ is constructed using so-called *hash-locked transactions*, and "smart contracts"[1] that guarantee that nobody looses money. This is possible thanks to the way in which the $n$ hours" deadlines in the channels $P_1 \multimapboth P_2$ and $P_2 \multimapboth P_3$ are chosen. An interesting feature of this protocol is that proof $\pi$ serves not only for internal purposes of the routing algorithm, but can also be viewed as the output of the protocol which can be used by $P_1$ as a receipt that she transferred some coins to $P_4$. In other words: $P_1$ can use $\pi$ to resolve disputes between with $P_4$, either in some smart contract (that was deployed earlier, and uses the given PCN for payments), or outside of the blockchain.

### B. Our contribution

One of the main problems with the existing PCNs is that sending a payment between two parties requires a path from the sender to the receiver that has sufficient capacity. This problem is amplified by the fact that capacity of potential paths can change dynamically, as several payments are executed in parallel. Although usually the payments are very fast, in the worst case they can be significantly delayed since each "hop" in the network can take as long as the pessimistic blockchain reaction time. Therefore it is hard to predict exactly what will be the capacity of a given path even in very close future. This is especially a problem if capacity of a given channel is close to being completely exhausted (i.e. it is close to zero, because of several ongoing payments). Some research [8] suggests that while Lightning is very efficient in transferring small amount of coins, transferring the larger ones is much harder, and in particular transfers of coins worth \$200 succeed with probability 1%. A natural idea for solving this problem is to split the payments along the way into several sub-payments. This was described in several recent papers (see, e.g., [12, 13, 27, 28, 32]). However, up to our knowledge all these papers considered so-called "*atomic* payment splitting", meaning that either all the sub-payments got through, or none of them. In this paper we prose a new, alternative technique that we call "*non*-atomic payment splitting" that does not have this feature, and hence is more flexible. (We provide a comparison between atomic and non-atomic splitting in Sec. III-2.) More concretely, our contribution can be summarized as follows.

*1)* We introduce the concept of *non-atomic payment splitting* by defining formally a notion of *Non-Atomic Payment Splitting (NAPS)* protocols. In our definition we require that splitting is done ad-hoc by the intermediaries, possibly in a reaction to dynamically changing capacity of the paths, or to the fees. Perhaps the easiest way to describe NAPS is to look at the payment networks as tools for outsourcing payment delivery. For example, in the scenario from Sect. II-A party $P_1$ outsources to $P_2$ the task of delivering 6¢ to $P_4$, and gives $P_2$ three hours to complete it (then $P_2$ outsources this task to $P_3$ with a more restrictive deadline). The sender might not be interested in *how* this money is transferred, and the only thing that matters to her is that it is indeed delivered to the receiver, and that she gets the receipt. In particular, the sender may not care if the money gets split on the way to the receiver, i.e., if the coins that he sends are divided into smaller amounts that are transferred independently over different paths. In many cases the sender may also be OK with

[1] Recall that Lightning is built over Bitcoin, which has a very limited "smart contract" support, hence these "smart contracts" are different that the ones considered in this paper, see [30].

not all the money being transferred at once. More precisely, suppose that he intends to transfer $u$¢ to the receiver. Then he can also accept the fact that $v < u$ coins were transferred (due to network capacity limitations), and try to transfer the remaining $u - v$ coins later (in another "installments"). Also, in many cases (e.g. BitTorrent-type file sharing) the goods that the seller delivers in exchange for the payment can be divided into very small units, and sent to the buyer depending on how many coins have been transferred so far. Finally, in several cases (e.g. depositing coins in so-called "cryptocurrency exchanges") uploading a non-full amount is "better than nothing". NAPS protocol permits such recursive non-atomic payment splitting into "sub-payments" and partial transfers of the coins.

*2)* We construct a protocol that we call ETHNA (see Appx. A for an explanation of this name choice) that satisfies the NAPS definition. In ETHNA the "sub-receipts" for sub-payments are aggregated by the intermediaries into one short sub-receipt, so that their size does not grow with the number of aggregated sub-receipts. This is done very efficiently, and in particular avoiding using advanced and expensive techniques such as non-interactive zero knowledge or homomorphic signature schemes and hash functions. Instead, we rely on a technique called "fraud proofs" in which an honest behavior of parties is enforced by a punishing mechanism (this method was used before, e.g., in [11, 17, 29, 33]). We stress that the amount of data that is passed between two consecutive parties on the path does *not* depend on the number of sub-payments in which the payment is later divided. The same applies to the data that these two parties send to the blockchain in case there is a conflict between them. We summarize the complexity of ETHNA in Sec. VI-B.

*3)* We provide a formal security analysis of ETHNA. More precisely we prove that ETHNA satisfies the the NAPS definition. We also analyze ETHNA's complexity.

*4)* We also implement ETHNA contracts in Solidity (the standard language for programming smart contracts in Ethereum), and we provide some routing experiments. We describe this implementation and provide some benchmarks in Sec. VII-1. We stress, however, that routing algorithms are *not* the main focus of this work, and further research on designing algorithm that exploit non-atomicity of payment splitting.

### *C. Other related work and organization of the paper*

Some of the related work was mentioned already before. Off-chain channels are a topic of intensive research, and there is no space here to describe all the recent exciting developments [1, 4, 9, 10, 11, 13, 14, 19, 20, 22, 23, 24, 25] in this area. The reader can also consult SoK papers on off-chain techniques [15, 16]. Partial coin transfers were considered in [28], but with no aggregation techniques and ad-hoc splitting. In a recent, very interesting paper Bagaria et al. [2] proposed a *Boomerang* system which allows to split the payments (by the sender) into multiple parts in a "redundantly" and tolerate the fact that only some of them succeed. The papers [2, 12, 28] focus on routing techniques, which is not the main focus of the paper.

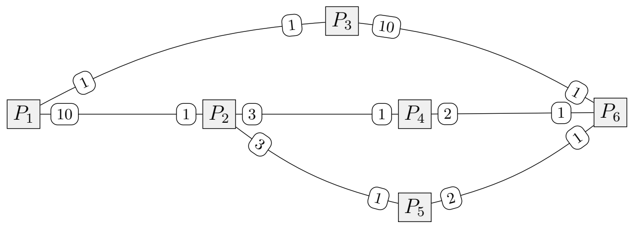
*Organization of the rest of the paper:* The next two section contain an informal description of our ideas: in Sec. III we provide an overview of NAPS definition, and in Sec. IV we describe the main design principles of ETHNA. Then, in Sec. V we provide the formal NAPS definition, and in Sec. VI the detailed description of ETHNA, together with security proof. Hence, in some sense Sec. V contains the "formal details" of Sec. III, and Sec. VI – the details corresponding to Sec. IV. An overview of our implementation and the simulations is presented in Sec. VII.

*Notation:* For standard definitions of cryptographic algorithms such as signature schemes or hash functions, see, e.g., [18]. When we say that a message is "signed by some party" we mean that it was signed using some fixed signature scheme that is existentially unforgeable under chosen-message attack. Natural numbers are denoted with $\mathbb{N}$. We will also use the notion of *nonces*. Their set is denoted with $\mathcal{N}$. We assume that $\mathcal{N} = \mathbb{N}$. We use some standard notation for functions, string operations, and trees. For completeness it is presented in Appx. B.

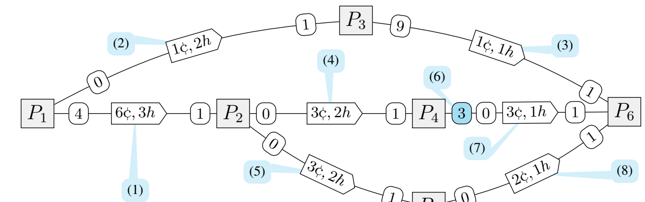
## III. OVERVIEW OF THE NAPS DEFINITION

Let us now explain informally the NAPS protocol features (for a formal definition see Sec. V). As highlighted above, the main advantage of NAPS protocols over the existing PCNs is that they allow ad-hoc splitting of a payment into sub-payments Throughout this paper we use the following convention: our protocols are run by a set of parties denoted $\mathcal{P} = \{P_1, \ldots, P_n\}$, where $P_1$ be the *sender*, $P_2, \ldots, P_{n-1}$ be the *intermediaries*, and $P_n$ be the *receiver*. Moreover, let $v$ be the amount of coins that $P_1$ wants to send to $P_n$, and let $t$ be the maximal time until when the transfer of coins should happen. Since in general $P_1$ can perform multiple payments to $P_n$, we assume that each payment comes with a nonce $\mu \in \mathcal{N}$ that can be later used to identify this payment. Sometimes we will simply call it "payment $\mu$". In this paper we present our protocol in a stand-alone way, i.e., we do not take into account possible parallel executions of the same protocol (e.g., with $P_2$ being the sender and $P_1$ being one of the intermediaries) and other ones. This is done purely for the sake of simplicity, and we conjecture that our protocol satisfies such stronger "composability" [3] properties. We leave analyzing this as an open research direction.

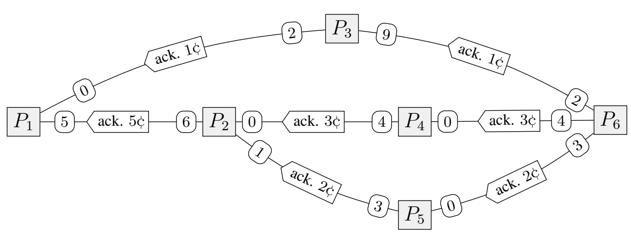
*1) NAPS behavior when everybody is honest:* For simplicity we start with an informal description of how NAPS protocols operate when all the parties are honest. The security properties (taking into account malicious behavior of the parties) are described informally in Sec. III-3, and formally defined in Sec. V. The easiest way to understand NAPS is to look a the example on Fig. 1. We provide a more general description below. Let us start by describing how the protocol looks like from the point of view of the sender $P_1$. Let $P_{i_1}, \ldots, P_{i_t}$ be the neighbors of $P_1$, i.e., parties with which $P_1$ has channels. Suppose the balance of each channel $P_1 \multimapboth P_{i_j}$ is $[P_1 \mapsto x_i, P_{i_j} \mapsto y_j]$ (meaning that $P_1$ and $P_{i_j}$ have $x_i$ and $y_j$ coins in their respective accounts in this channel). Now, $P_1$ chooses to push some amount $v_j$ of coins to $P_n$ via some




(a) The channel graph with the initial coin distribution.




(b) The sender $P_1$ wants to send 7¢ to the receiver $P_6$. She splits these coins into two amounts: 6¢ pushed to $P_2$ and 1¢ is pushed to $P_3$. This is indicated with labels (1) and (2) respectively. Then (3) party $P_3$ simply pushes 1¢ further to $P_6$. Party $P_2$ splits 6¢ into 3¢ + 3¢, and pushes 3¢ to both $P_4$ (4) and $P_5$ (5). Path $P_4 \rightarrow P_6$ initially had capacity 2 only (see Fig. (a) above), but luckily in the meanwhile 1¢ got unlocked (6) for $P_4$ in channel $P_4 \multimap P_6$, and hence (7) party $P_4$ pushes all 3¢ to $P_6$. No coins got unlocked in channel $P_5 \multimap P_6$, so $P_5$ pushes only 2¢ to $P_6$. The channel balances correspond to the situation *after* the coins are pushed (except of channel $P_4 \multimap P_6$ where we also indicated the fact that 1¢ got unlocked (6)).
Each party $P$ can also decide on her own about the timeout $t$ of each sub-payment that she pushes (this timeout in hours is indicated with "$h$"). The only restriction is that $t$ has to come at least 1 hour before the time she has to acknowledge that sub-payment back. This is because $P$ needs this "safety margin" of 1 hour in case $P'$ is malicious, and the acknowledgment has to be done "via the blockchain".




(c) Party $P_6$ acknowledges sub-payment of 1¢ to $P_3$, which, in turn acknowledges it to $P_1$. Party $P_6$ also acknowledges sub-payment of 3¢ to $P_4$ and 2¢ to $P_5$, who later acknowledge them to $P_2$. Once $P_2$ receives both acknowledgments she "aggregates" them into a single acknowledgment (for 5¢) and sends it to $P_1$. As a result 5¢ + 1¢ = 6¢ are transferred from $P_1$ to $P_6$. The channel balances correspond to the situation *after* the coins were acknowledged. Note that these actions can happen concurrently, e.g., acknowledgments along the path $P_6 \rightarrow P_3 \rightarrow P_1$ can be arbitrarily interleaved with what is done in the other parts of the graph (even before steps (4) and (5) on Fig. (b) above started)

Fig. 1: An example of a NAPS protocol execution. An edge "$P_i$ $x$ — $y$ $P_j$" denotes the fact that there exists a channel between $P_i$ and $P_j$, and the parties have $x$ and $y$¢ in it, respectively. We stress that actions from Figs. (b) and (c) can be interleaved (see caption under Fig. (c) for an example)

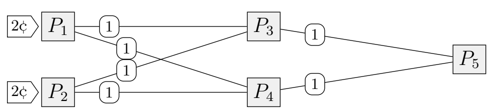
$P_{i_j}$, and set up a deadline $t_j$ for this (we will also call $v_j$ a *sub-payment* of payment $\mu$). This results in: (a) the balance $[P_1 \mapsto x_i, P_{i_j} \mapsto y_j]$ changing to $[P_1 \mapsto x_i - v_j, P_{i_j} \mapsto y_j]$, (b) the amount of coins that $P_1$ still wants to transfer to $P_n$ being decreased as follows: $v := v - v_j$, and (c) $P_{i_j}$ holding "$v_j$ coins that she should transfer to $P_n$ within time $t_j$.

It is also OK if $P_{i_j}$ transfers only some part $v'_j < v_j$ of this amount (this can happen, e.g., if the paths that lead to $P_n$ via $P_j$ do not have sufficient capacity). In this case, $P_1$ has to be given back the remaining ("non-transferred") amount $r = v_j - v'_j$. More precisely, before time $t_j$ comes, party $P_{i_j}$ acknowledges the amount $v'_j$ that she managed to transfer. This results in (1) changing the balance of the channel $P_1 \multimap P_{i_j}$ by crediting $v'_j$ coins to $P_{i_j}$'s account in it, and (2) $r$ coins to $P_1$'s account. Moreover (3) $P_1$ adds back the non-transferred amount $r$ to $v$, by letting $v := v + r$. Here (1) corresponds to the fact that $P_{i_j}$ has to be given the coins that she transferred (and hence "lost" in the other channels"), and (2) comes from the fact that not all the coins were transferred (if $P_{i_j}$ managed to transfer all the coins, then, of course, $r = 0$). Finally, (3) is used for $P_1$'s "internal bookkeeping" purposes, i.e., $P_1$ simply writes down the fact that $r$ coins "were returned" and still need to be transferred.

While party $P_1$ waits for $P_{i_j}$ to complete the transfer that it requested, she can also contact some other neighbor $P_{i_k}$ asking her to transfer some other amount $v_k$ to $P_n$. This is done in exactly the same way as transferring coins via $P_{i_j}$ (described above). In particular, the effects on the balance of the channel $P_1 \multimap P_{i_k}$ are as before (with subscript "$j$" replaced with "$k$"). In the example on Fig. 1 party $P_1$ splits 7 coins into 6 (that she pushes to $P_2$) plus 1 that she pushes to $P_3$. In more advanced cases several such transfers can be done in parallel with other neighbors of $P_1$. Moreover, $P_1$ can push several sub-payments (of payment $\mu$) to one neighbor. For example, $P_1$ can push again some new amount to $P_{i_j}$ hoping that maybe this time there will be more capacity available for routing payments via this party.

This process can be repeated by the intermediaries. Let $P$ be a party that holds some coins that were "pushed" to it by some $P'$ (and that originate from $P_1$ a have to be delivered to $P_n$). Now, $P$ can split them further, and moreover she can decide on her own how this splitting is done depending, e.g., on the current capacity of the possible paths leading to $P_n$. For instance, $P_{i_j}$ can decide to split $v_{i_j}$ further to between its neighbors in the same way as $P_1$ split $u$ between its neighbors. The payment splitting can be done in an arbitrary way, except of two following restrictions. First of all, we do not allow are "loops" (i.e. paths that contain the same party more than once), as it is hard to imagine any application of such a feature. In the basic version of the protocol we assume that the number of times a given payment sub-payment is split by a single party $P$ is bounded by a parameter $\delta \in \mathbb{N}$, called *arity* (for example arity on Figs. 1 is at most 2). In Appx. E2 we present an improved protocol where $\delta$ is unbounded (at a cost of a mild increase of the pessimistic number of rounds of interaction). As already mentioned, the most important feature of NAPS is the *non-atomicity of payments*. We discuss it further below.

*2) Atomic vs. non-atomic payment splitting:* As already highlighted in Sec. II-B the previous protocols on payment splitting always required payments to be atomic, meaning that in order for a payment to succeed all the sub-payments had to reach the receiver. Technically, this means that in order to issue a receipt for *any* of the sub-payments (this receipt is typically a pre-image of a hash function, see, e.g., [12]) all of them need to reach the receiver. This has several disadvantages: (1) the coins remain blocked in every path at least until the last sub-payment arrives to the receiver, (2) the success of a given sub-payment dependents not only on the subsequent intermediaries, but also on the other "sibling"paths (this problem was observed in [12] where it is argued that this risk may lead to intermediaries rejecting sub-payments that were split before, see Sec. 3.1 of [12]). Finally, atomic payments may result in the "deadlock" situations in the network. Since this may be of independent interest, we describe it in more detail below. Consider a channel graph as below (for simplicity we do not specify the coin amounts on the right-hand-sides of the channels, as they are irrelevant to this example).




Now suppose that $P_1$ and $P_2$ decide to send 2¢ each to $P_5$ via $P_3$ and $P_4$. If now $P_1$ pushes 1¢ to $P_3$ and at the same time $P_2$ pushes 1¢ to $P_4$, then none of the payments can be completed (since the channels $P_3$ ⊶ $P_5$ and $P_4$ ⊶ $P_5$ do not have sufficient capacity). On the other hand: if we allow *non*-atomic payments then each payment will partially succeed (i.e. each sender will send 1¢ to the receiver $P_5$). They may then try to send the remaining amounts after some time when new capacity in these channels is available. This is of course a very simple scenario, but it can be generalized to much larger graphs, and to more complicated "deadlocks".

On the other hand, "atomicity" and even "fine grained atomicity" can be also obtained in ETHNA by a small modification of the protocol. We write more about it in Sec. E1. Let us also remark that atomic payment splitting in general seems to be easier to achieve, which is probably the reason why there has been more focus on them in the literature (with papers focusing more on other aspects of this problem, such as routing algorithms, e.g. [12]). Finally, let us stress that we do not claim that non-atomicity is in any way superior to atomicity. We think that both solutions have their advantages and disadvantages, and there exist applications where each of them is better than the other one.

*3) NAPS security properties:* In the description in Sec. III-1 we assumed that all the parties are behaving honestly. Like all the other PCNs, we require that NAPS protocols work also if the parties are malicious, and in particular, no honest party $P$ can loose money, even if all the other parties are not following the protocol and are working against $P$. The corrupt parties can act in a coalition, which is modeled by an adversary Adv. For the sake of simplicity we assume synchronous network, with Adv being rushing. Formal security definition appears in Sec. V Let us now informally list the security requirements, which are quite standard, and hold for most PCNs (including Lightning).

The first property is called "fairness for the sender". To define it, note that as a result of payment $\mu$ (with timeout $t$), the total amount of coins that each party $P$ has in the channel with other parties typically changes. Let $net_\mu(P)$ denote the amount of coins that $P$ gained in all the channels. Of course $net_\mu(P)$ can be negative if $P$ lost $-net_\mu(P)$ coins. We require that by the time $t$ an honest $P_i$ holds a receipt of a form

$$\text{“an amount } v \text{ of coins has been transferred from } P_1 \text{ to } P_n \text{ as a result of payment } \mu\text{”,} \tag{1}$$

with $v \leq u$. Moreover, under normal circumstances, i.e. when everybody is honest, $v$ is equal to $-net_\mu(P_1)$ (i.e. the sum of the amounts that $P_1$ lost in the channels). In case some parties (other than $P_1$) are dishonest, the only thing that they can do is to behave irrationally, and let $v \geq -net_\mu(P_1)$, in which case $P_1$ holds a receipt for transferring *more* coins than she actually lost in the channels. Note that introducing receipts makes our model stronger than the models that have no receipts (e.g. [30]). This is because the "no receipts" settings makes sense only under the assumption that the sender and the receiver trust each other, and in particular the receiver is not corrupt (which is is a *stronger* security assumption that the one that we use in our paper). A receipt can be later used in another smart contract (e.g., a contract that delivers some digital goods whose amount depends on $v$). "Fairness for the receiver" is defined analogously, i.e.: if $P_1$ holds a receipt (1) then typically $v = net(P_n)$, and if some parties (other than $P_n$) are dishonest, then they can make $v \leq net_\mu(P_n)$. In other words, $P_1$ cannot get a receipt for an amount that is higher than what $P_n$ actually received in the channels. Finally, we require that the following property called "balance neutrality for the intermediaries" holds: for every honest $P \in \{P_2, \ldots, P_{n-1}\}$ we have that $net_\mu(P_n) \geq 0$. Again: if everybody else is honest then we have equality instead of inequality.

## IV. OVERVIEW OF THE ETHNA PROTOCOL

After presenting NAPS definition, let us now explain the main ideas behind the protocol ETHNA protocol that realizes it (for a formal description of the construction see Sec. VI, and for an overview of the implementation see Sec. VII-1). A very important feature of ETHNA is that it permits "sub-receipt aggregation", by which we mean the following. Consider some payment $\mu$. Each time after $P_n$ receives some sub-payment $v$ that reached it via some path $\Pi = P_1 \twoheadrightarrow P_{i_1} \twoheadrightarrow \cdots \twoheadrightarrow P_{i_t}$ it issues a sub-receipt and sends it to $P_{n-1}$. Each intermediary that received more than one sub-receipt can aggregate them into one short sub-receipt that she sends further in the direction of $P_1$. Finally, $P_1$ also produces one short receipt for the entire payment. This results in small communication complexity, and in particular, the pessimistic gas costs are low. We discuss this in more detail in Secs. VI-B and VII-1. One option for doing this would be to let the sub-receipt be signed using a homomorphic signature scheme, and then exploit this homomorphism to aggregate the sub-receipts. In this paper we use a simpler solution that can be efficiently and easily implemented in the current smart-contract platforms.

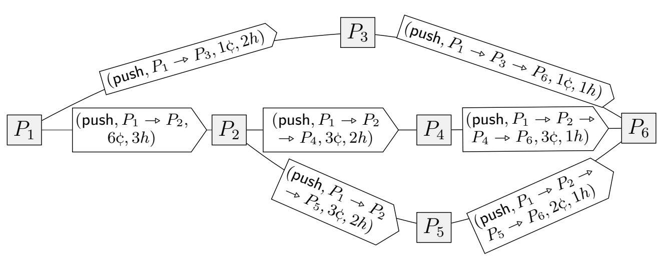
Very informally speaking, we ask $P_n$ to perform the "sub-payment aggregation herself" (this is done at the moment of signing a sub-receipt, and does not require any further interaction with $P_n$). Then, we just let the other parties verify that this aggregation was performed correctly. If any "cheating by $P_n$" is detected (i.e. some party discovers that $P_n$ did not behave honestly) then a proof of this fact (called a "fraud proof") will count as a receipt that a full amount has been transferred to $P_n$. From the security point of view this is ok, since an honest $P_n$ will never cheat (and hence, no "fraud proof" against him will ever be produced). Thanks to this approach, we completely avoid using any expensive advanced cryptographic techniques (such as homomorphic signatures, or non-interactive proofs). Below we explain the main idea of ETHNA by considering the example from Fig. 1.

### A. The "everybody is honest" case

Again, we start with describing how the protocol works when everybody is honest, and then (in Sec. IV-B) we show how the malicious behavior is prevented.

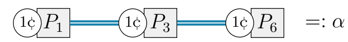
*1) Invoice sending:* The protocol starts with the receiver $P_n$ sending to $P_1$ an "invoice" that specifies (among other things) the identifier $\mu$ of the payment, and the maximal amount $v$ of coins that $P_n$ is willing to accept. As we explain below, this invoice may be later used together with "fraud proofs" to produce a proof that all the $v$ coins were transferred to $P_n$ (if she proves to be malicious).

*2) Pushing sub-payments:* Pushing sub-payments is done by sending messages containing information about the path that the sub-payment "traveled" so far (together with the amount of coins to be pushed and a timeout information), and simultaneously blocking coins in the underlying channels. The messages sent between the parties on Fig. 1a are presented on the picture below.




Whenever a message "$(\mathsf{push}, \pi, v, t)$" is sent from $P$ to $P'$, the party $P$ blocks $v$ coins in channel $P \multimap\!\!\circ P'$ for time $t$. These coins can be claimed by $P'$ is she provides a corresponding sub-receipt within time $t$. Otherwise it can be reclaimed back by $P$.

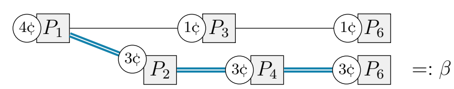
*3) Acknowledging sub-payments by the receiver:* The receiver $P_6$ acknowledges that sub-payments by sending signed sub-receipts back to the intermediaries, and simultaneously claiming the coins that were blocked in the corresponding channels. Simultaneously the receiver $P_6$ creates a graph called "payment tree" that is stored locally by $P_6$ and grows with each acknowledged sub-payment. Consider now Fig. 1c. As explained before, the order of message acknowledgment can be arbitrary. In what follows we assume that $P_6$ first acknowledges the sub-payment that came along the path $P_1 \twoheadrightarrow P_3 \twoheadrightarrow P_6$. This means that $P_6$ "accepts" that 1¢ will be transferred to her from $P_1$ via path $P_1 \twoheadrightarrow P_3 \twoheadrightarrow P_6$, or, in other words: 1¢ will be "passed" through each of $P_1, P_3$, and $P_6$ (note that we included here the sender $P_1$ and the receiver $P_n$). This can be depicted as the following graph that consists of a single path that we denote $\alpha$:



(2)

In order to acknowledge the sub-payment that was pushed along the path $P_1 \twoheadrightarrow P_3 \twoheadrightarrow P_6$ party $P_6$ signs $\alpha$ and sends it to $P_3$. Such signed information (in a slightly generalized form) will be called a "sub-receipt" (see Sec. VI). By providing this sub-receipt party $P_6$ also gets 1¢ in the $P_3 \multimap\!\!\circ P_4$ (that were blocked by $P_3$ in this channel when the "push" message was sent). The graph from Eq. (2) is the first version of the payment tree that, as mentioned above, the receiver $P_6$ stores locally.

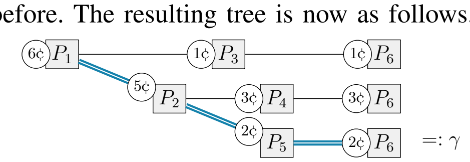
Now, suppose the next sub-payment that $P_6$ wants to acknowledge is the one that came along the path $P_1 \twoheadrightarrow P_2 \twoheadrightarrow P_4 \twoheadrightarrow P_6$, i.e., $P_6$ accepts that 3¢ will be transferred to her from $P_1$ via path $P_1 \twoheadrightarrow P_2 \twoheadrightarrow P_4 \twoheadrightarrow P_6$. The receiver $P_6$ now modifies the payment tree as follows:



(3)

Analogously to what we saw before, this tree represents the total amounts of coins that will be "passed" through different parties from $P_1$ to $P_6$ after the acknowledgment of this sub-payment is completed. On Eq. (3) the thick line (denoted $\beta$) corresponds to the "new" path, and the thin one is taken from Eq. (2), except that $P_1$ is labeled with "4¢". This is because the total amount of coins that will be passed through $P_1$ is equal to the sum of the coins passed before (1¢) and now (3¢). Observe also that $P_6$ appears in "two copies" on Eq. (3). The is because the graph that we construct is a *tree* (actually every leaf of a payment tree will be labeled by the receiver). Party $P_6$ now signs path $\beta$ to create a sub-receipt that she sends to $P_4$ in order to claim 3¢ in the channel $P_4 \multimap\!\!\circ P_6$.

Finally, $P_6$ acknowledges the sub-payment that came along the path $P_1 \twoheadrightarrow P_2 \twoheadrightarrow P_5 \twoheadrightarrow P_6$. This is done similarly to what we did before. The resulting tree is now as follows.



(4)

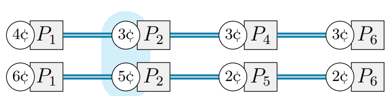
Note that we performed "summing" in two places on Eq. (4): at the node $P_1$ (where we computed 6¢ as 4¢ + 2¢) and an $P_2$ (where 5¢ = 2¢ + 3¢). Labeled path $\gamma$ is now signed by $P_6$ and sent to $P_5$ as sub-receipt in order to claim 2¢.

The payment trees whose examples we saw on Eqs. (2)–(4) are defined formally (in a slightly more general version) in Sec. VI-A2 on p. 11. Their main feature is that value of coins in the label on each node $P$ is equal to the sum of the labels of the children of $P$. By a standard recursive application of this observation this implies that the coin value in a label of $P$ is equal to the sum of labels in the leaves of the sub-tree rooted in $P$. In particular: the label in the root of the entire tree is equal to the sum of the values in the leaves.

*4) Acknowledging sub-payments by the intermediaries:* We now show how the intermediaries $P_2, \ldots, P_5$ acknowledge the sub-payments. On a high level this is done by propagating the sup-receipts (issued by $P_6$) from right to left. Note, that each party may receive several such sub-receipts (if she decided to split a given sub-payment). Let $\mathcal{S}$ be the set of such sub-receipts (such sets will be called "payment reports", see Sec. VI for their formal definition). When a party $P$ wants to acknowledge the sub-payment she chooses (in a way that we explain below) one of the sub-receipts $\zeta$ from her set $\mathcal{S}$. She then forwards it back in the left direction to the party $P'$ that pushed the given sub-payment to her. As a result $P$ gets $v¢$ in the channel $P' \multimap\!\!\circ\, P$. To determine the value of $v¢$ the following rule is used: it is defined to be the label of $P$ on the path $\zeta$. Given this, the rule for choosing $\zeta \in \mathcal{S}$ is pretty natural: $P$ simply chooses such the $\zeta$ that maximizes $v$. Such $\zeta$ will be called a "leader" of $\mathcal{S}$ (at node $P$). See Sec. VI for the formal definition of this notion. To illustrate it let us look again at out example from Fig. 1.

First, observe that $P_3$ holds only one sub-receipt (i.e.: the signed path $\alpha$). She simply forwards it to $P_1$ and receives $1¢$ in the channel $P_1 \multimap\!\!\circ\, P_3$. Note that this is exactly equal to the value that she "lost" in the channel $P_3 \multimap\!\!\circ\, P_6$, and hence the balance neutrality property holds. The situation is a bit more complicated for $P_2$ since she holds two paths signed by the receiver: $\beta$ (defined on Eq. (3)) and $\gamma$ (from Eq. (4)). By applying the rule described above $P_2$ chooses the leader $\zeta$ at $P_2$ to be equal to $\gamma$ (since $5¢ > 3¢$). This is depicted below (the shaded area indicates the labels that are compared).

$\beta =$
$\gamma =$




(5)

What remains is to argue about balance neutrality for $P_2$, i.e. that number of coins received by $P_2$ in the channel $P_1 \multimap\!\!\circ\, P_2$ is equal to the sum of coins that she "lost on the right-hand side". In this particular example it can be easily verified just by looking at Eq. (5) ($5¢$ are "gained", and $2¢+3¢$ are "lost"). In the general case the formal proof is based on the property that that value of coins in the label on each node $P$ in a payment tree is equal to the sum of the labels of the children of $P$. See Sec. VI, and particular Claim 1, for the details.

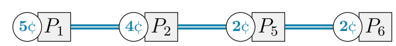
*5) Final receipt produced by $P_1$:* Once all the sub-payments are over $P_1$ decides to conclude the procedure and obtain the final receipt for the entire payment (see Eq. (1) on page 5). Again, $P_1$ holds a "payment report" $\mathcal{S}$, i.e. a set of paths signed by $P_6$. In the case of our example these paths are: $\alpha$ (sent to $P_1$ by $P_3$) and $\gamma$ (sent by $P_2$). Party $P_1$ chooses her "receipt" in a similar way as the intermediaries choose which sub-receipt to forward. More precisely, let $\zeta$ be the path that is the leader of $\mathcal{S}$ at node $P_1$. This path becomes the final receipt. The amount of coins that are transferred is equal to the label of $P_1$ in $\zeta$. In our case, the leader $\zeta$ is clearly $\gamma$ (since its label at $P$ is "6¢", while the label of $\gamma$ at $P$ is "1¢", cf. Eqs (2) and (4)). Hence, $\gamma$ becomes the final receipt for the payment of 6 coins.

"Fairness for the sender" follows from the same argument as the "balance neutrality for the intermediaries". For "fairness for the receiver" observe that $\zeta$ is signed by the receiver, and is taken from the payment tree (created and maintained by the receiver). To finish the argument recall that: (a) as observed before the label in the root of such a tree is always equal to the sum of the labels in its leaves, and (b) this sum is exactly equal to the total amount of coins that the receiver received from its neighbors during this payment procedure. For the details see Lemma 2 on page 12.

## *B. Dealing with malicious behavior*

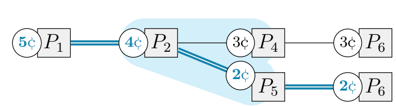
The main type of malicious behavior that we have to deal with is cheating by the receiver $P_n$ whose goal could be to get more coins than appears on the final receipt held by the sender $P_1$. This could potentially be done at the cost of $P_1$ or some of the intermediaries. So far, we have not described how to guarantee that $P_n$ produces the sub-receipts correctly. As already highlighted, our trick is to let the malicious $P_n$ produce the sub-receipts in an arbitrary way, and later let other parties verify $P_n$'s operation. This is based on the idea of "fraud proofs": if an intermediary $P$ finds a proof that $P_n$ is cheating she can automatically claim all the coins that were pushed to her by forwarding this proof "to the left". In this way the cheating proof reaches the sender $P_1$ who can now use it as the receipt for transferring the full amount that was requested (recall that $P_1$ holds an "invoice" from $P_n$).

Suppose, e.g., that in our scenario $P_6$ cheats by sending to $P_5$, instead of $\gamma$ (see Eq. (4)), the following sub-receipt:

$\widehat{\gamma} :=$ 

(6)

The receiver does it in order to make $P_1$ hold a receipt for $5¢$, while in fact receiving $6¢$. Party $P_5$ has no way to discover this fraud attempt (since from her local perspective everything looks ok), so $2¢$ get transferred to $P_6$ in the channel $P_5 \multimap\!\!\circ\, P_6$. Party $P_5$ forwards $\widehat{\gamma}$ to $P_2$ and gets $2¢$ in the channel $P_2 \multimap\!\!\circ\, P_5$ (hence the "balance neutrality" property for her holds). Now look at this situation from the point of view of $P_2$. In addition to $\widehat{\gamma}$ she got one more sub-receipt, namely $\beta$ (see, e.g., Eq. (5)). Party $P_2$ preforms a "consistency check" by combining $\widehat{\gamma}$ and $\beta$. This is done by trying to locally reconstruct the part of the payment tree that concerns $P_2$. This is done as follows. First observe that the value on the label of $P_1$ in $\beta$ is $4¢$, which is smaller than the label of $P_1$ in $\widehat{\gamma}$ (which is equal to $5¢$). This means that $\beta$ had to be signed by $P_6$ *before* she signed $\widehat{\gamma}$. Hence $P_2$ first writes down $\beta$, and then on top of it she writes $\widehat{\gamma}$ (possibly overwriting some values). Normally (i.e. when $P_6$ is honest) this should result in a sub-tree of the tree from Eq. (4). However, since $P_6$ was cheating the resulting graph is different. Namely, $P_2$ reconstructs the following:




(7)

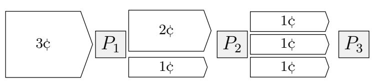
It is now obvious that $P_6$ is cheating, since the labels on the children of $P_2$ sum up to $5¢$, which is larger than $4¢$ (the label of $P_2$). This "inconsistency" is marked as a shaded region on Eq. (7). Hence the set $\{\beta, \widehat{\gamma}\}$ is a "fraud proof" against $P_6$. As described above, once we get such a proof we are "done":

simply each intermediary can use it to claim all the money that was blocked for her, and the receiver can use it as a receipt that *all* the coins were transferred. Let us stress that, of course, none of the parties knows a priori if $P_6$ is cheating or not, and therefore in reality the above "consistency check" is performed always.

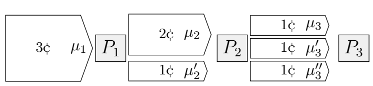
## V. NAPS FORMAL SECURITY DEFINITION

We now proceed to the formal exposition of the ideas already presented informally in Sec. III. Below $\Delta$ denotes maximal blockchain reaction time (typically: $\Delta \gg 1$).

*1) Payment routes:* We start with defining a generalization of the term "payment paths" that were introduced in Sec. III. As already explained, to be as general as possible, the NAPS definition permits that several sub-payment of the same payment $\mu$ are routed via the same party independently. Moreover, we allow using the same path for more than one sub-payment of the same payment. Consider, e.g., the following scenario: 3¢ is sent on a path $P_1 \twoheadrightarrow P_2 \twoheadrightarrow P_3$. This amount is first split by $P_1$ as: 2¢ + 1¢. The 2¢ is split again as: 1¢ + 1¢ and sent to $P_3$, while 1¢ is just delivered directly to $P_3$ without being split further. Pictorially:




Obviously all the 3 coins above traveled along the same path, but nevertheless they have to be considered as separate sub-payments. In order to uniquely identify each of them, we introduce a concept of "payment *routes*" that are very similar to "payment paths", except that they contain additional information that makes them unique (in the situations as above). More concretely, a "route" is a "path" with nonces added in every hop. For example, the nonces added to the scenario above are as follows.




This results in the following routes: $\langle(P_1,\mu_1),(P_2,\mu_2),(P_3,\mu_3)\rangle$, $\langle(P_1,\mu_1),(P_2,\mu_2),(P_3,\mu_3')\rangle$, and $\langle(P_1,\mu_1),(P_2,\mu_2'),(P_3,\mu_3'')\rangle$. We can think of every $\mu_i$ as being "contributed" by $P_i$. Moreover, we assume that $\mu_1$ ("contributed" by the sender $P_1$) is equal to the nonce that identifies the entire payment. Formally, for a channel graph $\mathcal{G} = (\mathcal{P}, E, \Gamma)$ a string $\pi = \langle(P_{i_1},\mu_1),\ldots,(P_{i_{|\pi|}},\mu_{|\pi|})\rangle$ is a *payment route over* $\mathcal{G}$ *for payment* $\mu$ if each $\mu_i \in \mathcal{N}$ is a nonce and $P_{i_1} \twoheadrightarrow \cdots \twoheadrightarrow P_{i_{|\pi|}}$ is a path in $\mathcal{G}$ that has at least two elements, its first element is equal to $P_1$, its last element is equal to $P_n$, and it has no loops (i.e. every element from $\mathcal{P}$ appears in $\pi$ at most once). We assume that a payment route corresponding to a payment $\mu$ will always start with $(P_1,\mu)$ (hence, as already mentioned above $\mu_1 := \mu$). We say that $P$ *appears on* $\pi$ *(at position* $j$*)* if we have that $P = P_{i_j}$. A *payment route prefix (over* $\mathcal{G}$*)* is a string $\pi'$ that is a prefix of some payment route over $\mathcal{G}$.

*2) Modeling parties and channels:* Suppose $\mathcal{G} = (\mathcal{P}, \mathcal{E}, \Gamma)$ is a channel graph. In our formal modeling every edge $e = P \multimap\!\!\!\circ\, P' \in \mathcal{E}$ has a corresponding machine denoted $C^e$. We assume that $C^e$ has two special registers denoted $C^e.\mathsf{cash}(P)$ and $C^e.\mathsf{cash}(P')$. The values in these registers are non-negative integers, and $C^e.\mathsf{cash}(P)$ denotes the amount of coins that $P \in e$ has in her account in $C^e$. Recall also that $C^{P \multimap\!\circ P'}.\mathsf{cash}$ can be viewed as a function $C^{P \multimap\!\circ P'}.\mathsf{cash} : \{P, P'\} \to \mathbb{Z}_{\geq 0}$. Channel machines can interact with $P$ and $P'$, and have a state (for this reason we will refer to the as "*state channels*"). See, e.g., [5, 11] (or ETHNA implementation described in Sec. VII-1) on how to implement state channels in real life.

Another "special" machine is the *receipt verification machine* RVM. The role of RVM is to model the fact that the receipts produced by $P_1$ need to be publicly-verifiable, so, e.g., they can be used later in another smart contract, see Sec. III-3 (it plays a role similar to the so-called "validation function" defined in Sec. 2.3 of [12]). We stress that the RVM has very limited interaction with the other machines. In fact, the only interaction that happens is: $P_1$ sends a message to RVM, and RVM decides if it is a valid receipt and outputs information on how many coins were transferred within a given payment. Hence, we can think of RVM as an efficiently computable ("non-interactive") function.

*3) The adversary and the environment:* The protocol is attacked by a polynomial-time rushing adversary $\mathsf{Adv}$ who can *corrupt* some parties (when a party is corrupt $\mathsf{Adv}$ learns all its secrets and takes a full control over it). The adversary can also send messages to the honest parties that influence their behavior in the protocol, and receive messages from them. A party that has not been corrupt is called *honest*. We assume that $\mathsf{Adv}$ gets $\mathcal{G}$ as input.

To model the fact that the parties can make internal decisions about the protocol actions we use a concept of an *environment* [3] that is responsible for "orchestrating" the execution. We model it by a poly-time interactive machine $\mathsf{Env}$. The party machines interact with the environment $\mathsf{Env}$ via messages starting with "env-" prefix. The environment sends the following messages to the parties: "env-send" and "env-receive" — sent simultaneously to $P_1$ and $P_n$ (respectively) and used to initiate a payment $\mu$, messages "env-push" — to push sub-payments further, and messages "env-acknowledge" — to acknowledge a payment. The parties respond with messages "env-pushed" and "env-acknowledge"— to signal that a sub-payment was pushed and acknowledged (respectively). The reason to have the env-pushed and env-acknowledged messages is purely technical[2] For reference, these messages and their syntax are summarized on a cheat sheet on Fig. 4 (see p. 16). We assume that the environment gets the channel graph $\mathcal{G}$ as input. The environment $\mathsf{Env}$ is called *admissible* if it satisfies certain criteria presented on Fig. 2.

It maintains a set $\Omega$ of "open push requests" (see Fig. 2) and functions *sent*, *value*, and *timeout* that are used to store information about these requests. We say that a party $P$ has an *open* push request if there exists $\pi \in \Omega$ such that $P$ appears

[2]Under the normal circumstances, if $\mathsf{Env}$ asked to $P$ to push a sub-payment to $P'$ then in the next round she will receive an "env-pushed" message from $P'$. Of course, this does not need to be the case when $P$ or $P'$ are corrupt and hence the the "env-pushed" message is needed. The same applies to "env-acknowledge" and "env-acknowledged".

Env takes as input a channel graph $\mathcal{G} = (\mathcal{P}, \mathcal{E}, \Gamma)$, where $\Gamma$ will be treated as a variable that will be changing throughout the execution of Env (the values $\mathcal{P}, \mathcal{E}$ will remain constant). It also defines the following variables:

- $\Omega$ — a set of payments routes (initially empty) called the *open* push *requests*. When we say that we *open a* push *request* $\pi$, we mean that we add $\pi$ to $\Omega$. When we say that we *close a* push *request* $\pi$ we mean that we remove $\pi$ from $\Omega$.
- $\mathit{sent}, \mathit{value}, \mathit{timeout}$ — functions of a type $\Omega \to \mathbb{Z}_{\geq 0}$.

The environment interacts with the parties in an arbitrary way, as long as certain restrictions are satisfied. We specify "conditions" on when a message that Env receives is valid (if they are not met, then the message is ignored). Both the outgoing and incoming messages can result in modifications of the variables (we call these modifications the "side effects").

- Env sends a $(\text{env-send}, v, \mu, t)$ to $P_1$ and $(\text{env-receive}, v, \mu, t)$ to $P_n$.
*Restrictions:* (a) these messages have to be sent simultaneously, (b) the nonce $\mu$ has not been used before in the env-send and env -receive messages.
- Env receives $(\text{env-pushed}, (\pi || (P, \mu)), v, t)$ from a party $P$.
*Restrictions:* $t > \tau$, where $\tau$ is the current time.
*Side effects:* open a push request $(\pi || (P, \mu))$ and let $\mathit{sent}((\pi || (P, \mu))) := 0$ and $\mathit{value}((\pi || (P, \mu))) := v$ and $\mathit{timeout}((\pi || (P, \mu))) := t$.
We require that in time $t$ the latest Env sends (env -acknowledge, $(\pi || (P, \mu))$) (see below) to party $P$.
- Env receives $(\text{env-acknowledged}, (\pi || (P, \mu) || (P', \mu')), v)$ from a party $P$.
*Condition:* a push request $(\pi || (P, \mu) || (P', \mu'))$ is open.
*Side effects:* let $\mathit{sent}((\pi || (P, \mu))) := \mathit{sent}(\pi || (P, \mu))) + v$ and $\Gamma^{P \multimap\!\circ P'}(P) := \Gamma^{P \multimap\!\circ P'}(P) + v' - v$ and $\Gamma^{P \multimap\!\circ P'}(P') := \Gamma^{P \multimap\!\circ P'}(P') + v$, where $v' := \mathit{value}((\pi || (P, \mu) || (P', \mu'))$.
- Env sends $(\text{env-push}, (\pi || (P, \mu) || (P', \mu')), v, t)$ to a party $P$.
*Restrictions:* (a) no env-push request with the same argument $(\pi || (P, \mu) || (P', \mu'))$ has been sent before, (b) a push request $(\pi || (P, \mu))$ is open, (c) at most $\delta - 1$ requests env-push with the argument $(\pi || (P, \mu))$ have been sent before, (d) $t \leq \mathit{timeout}(\pi || (P, \mu)) - \Delta$, (e) $v \leq \mathit{value}(\pi || (P, \mu)) - \mathit{sent}(\pi || (P, \mu))$, and (f) $v \leq \Gamma^{P \multimap\!\circ P'}(P)$.
*Side effects:* let $\Gamma^{P \multimap\!\circ P'}(P) := \Gamma^{P \multimap\!\circ P'}(P) - v$.
- Env sends $(\text{env-acknowledge}, (\pi || (P, \mu) || (P', \mu')))$ to a party $P'$.
*Restriction:* a push request $(\pi || (P, \mu) || (P', \mu'))$ is open. No push request $(\pi || (P, \mu) || (P', \mu') || \pi')$ (for $\pi' \neq \epsilon$) is open.
*Side effects:* close this push request.

Fig. 2: Admissible Env for ETHNA with arity $\delta$.

on $\pi$. The main idea behind the "admissible environment" is that it restricts us to the environments that satisfy some natural correctness requirements, such as "do not push more coins than you hold in a given sub-payment". These conditions are called "restrictions". Most of the actions result in some modification of the internal variables of Env. These restrictions are called "side effects". The environment is also responsible for terminating the protocol. More concretely, we say that the protocol *terminated* is Env stops. The environment can only do it if there are no open push requests.

*4) The network model:* We assume a synchronous communication network, i.e., the execution of the protocol happens in rounds. The notion of rounds is just an abstraction which simplifies our model, and was used frequently in this area in the past (see, e.g., [10, 11]). Whenever we say that some operation (e.g. sending a message or simply staying in idle state) *takes at most* $\tau \in \mathbb{N} \cup \{\infty\}$ *rounds* we mean that it is up to the adversary to decide how long this operation takes (as long as it takes at most $\tau$ rounds). We assume that every machine is activated in each round. The communication between each two parties $P$ and $P'$ takes 1 round. Communicating with the state channel machines is a bit more subtle and it is therefore described in a separate section below.

*5) Immediate vs. non-immediate messages to the state channel machines:* One subtle point in modeling the state channels as machines is their response time. In real life executing state channel machines is done via an update procedure (see, e.g., [11]) where the parties mutually sign the new state of the channel. This procedure takes two rounds of communication in the optimistic case (i.e. when both parties are honest). The pessimistic case is a bit more tricky, since in general each update may require interacting with the blockchain. In some cases, however, even in the pessimistic case we can think of the update as being immediate. This happens when a party $P$ updates the channel $P \multimap\!\circ P'$ in a way that is *beneficial* for $P'$ (e.g., $P$ transfers money to $P'$). In this case $P$ just sends a new signed state of the channel and she does *not* need to wait for $P'$'s confirmation (and therefore the whole update procedure takes just 1 round, even if $P'$ is dishonest $P'$). The same situation happens in channels. In particular, if $P$ pushes a sub-payment to $P'$ she does not need to hold a confirmation that $P'$ received this message. In the worst case (if $P'$ is dishonest) simply this sub-payment not result in any coins being transferred. The situation is of course different for acknowledgments: here $P'$ needs to get a confirmation from $P$ that she received the acknowledgment. It is therefore convenient to distinguish between *immediate* and *non-immediate* updates to a channel $P \multimap\!\circ P'$. The "immediate" take 1 round (since no confirmation is needed), and the "non-immediate" ones take 2 rounds if both parties are honest, and they may take up to $\Delta$ rounds if one of $\{P, P'\}$ is dishonest.

*6) The definition:* A *Non-Atomic Payment Splitting (NAPS) protocol* $\Pi$ *for a channel graph* $\mathcal{G} = (\mathcal{P}, \mathcal{E}, \Gamma)$ is a tuple consisting of: the party machines $P_1, \ldots, P_n$, the state channel machines $C^e$ (for every $e \in \mathcal{E}$), and the receipt verification machine RVM that for every $\mathcal{A}$ and Env satisfy the *functionality* and *security* requirements described below.

*Functionality requirements:* The following must hold for every NAPS protocol with overwhelming probability. *Guaranteed sending:* Suppose $P_1$ and $P_n$ are honest, and they both simultaneously receive messages $(\text{env-send}, v, \mu, t)$ and $(\text{env-receive}, v, \mu, t)$ (respectively) from Env. Then in the next round $P_1$ sends $(\text{env-pushed}, (P_1, \mu), v, t)$ to Env. *Guaranteed pushing:* Suppose $P$ and $P'$ are honest, and $P$ receives a message $(\text{env-push}, (\pi || (P, \mu) || (P', \mu')), v, t)$ from Env (for some $\pi, v, t, \mu$, and $\mu'$). Then in the next round $P'$ sends a message $(\text{env-pushed}, (\pi || (P, \mu) || (P', \mu')), v, t)$ to Env. This is the only case when $P'$ sends an env-pushed message to Env with this route prefix. *Guaranteed acknowledgment by $P' \in \{P_2, \ldots, P_n\}$:* Suppose $P$ and $P'$ are honest, and $P'$ receives a message (env-acknowledge, $(\pi||(P,\mu)||(P',\mu'))$) from Env (for some $\pi,\mu$ and $\mu'$), and let $v := \mathit{sent}((\pi||(P,\mu)||(P',\mu'))$. Then in the next round $P$ sends a message (env-acknowledged, $(\pi||(P,\mu)||(P',\mu')), v$) to Env. This is the only case when $P$ sends an env-pushed message to Env with this route prefix. *Guaranteed acknowledgment by $P_1$:* Suppose $P_1$ is honest and it receives a message (env-acknowledge, $(P_1,\mu)$) from Env (for some $\mu$). Let $v := \mathit{sent}((P_1,\mu))$. Then in the next round the receipt verification machine outputs $(v,\mu)$.

*Security requirements:* Suppose some execution was performed and terminated. Let $\widehat{\Gamma}$ be a cash function describing the amount of coins in the state channels after this execution, i.e, let every $\widehat{\Gamma}(e)$ be equal to a function $f : e \to \mathbb{Z}_{\geq 0}$ such that $f(P) := C^e.\mathsf{cash}(P)$. Now, look at this execution from a perspective of some party machine $P$. Let $\mathcal{U}$ be the set of all parties that have a channel with $P$, i.e., let $\mathcal{U} = \{P' :$ such that $(P \multimap\!\circ P') \in \mathcal{E}\}$. The *net result of* $P$ in this execution (so far) is defined as $\mathit{net}(P) := \sum_{P' \in \mathcal{U}} \widehat{\Gamma}^{P \multimap\circ P'}(P) - \Gamma^{P \multimap\circ P'}(P)$. This can be extended to the state channels, namely, the *net result of channel* $e$ in this execution (so far) is defined as $\mathit{net}(e) := \sum_{P \in e} \widehat{\Gamma}^e(P) - \Gamma^e(P)$. Let us also define the *total transmitted* sum of coins until this moment as $\sum_{(\mu,v)\in\mathcal{W}} v$, where $\mathcal{W}$ is the set of outputs of RVM. The following requirements (already discussed informally in Sec. III-3) must hold for every NAPS protocol with overwhelming probability. *Fairness for the sender $P_1$:* Suppose that $P_1$ is honest and has no open push request. Then $\mathit{net}(P_1) + v \geq 0$. *Fairness for the receiver $P_n$:* Suppose that $P_n$ is honest. Then $\mathit{net}(P_n) - v \geq 0$. *Balance neutrality of the intermediaries:* Suppose that $P \in \{P_2, \ldots, P_{n-1}\}$ is honest and has no open push request, then $\mathit{net}(P) \geq 0$. *"No money printing" in the state channel machines:* For every channel $P \multimap\!\circ P'$ we have that $\mathit{net}(P \multimap\!\circ P') \leq 0$.

## VI. Formal description of the EthNA protocol

Let us start with providing formal description of some of the terms that were already informally introduced in Sec. IV. For a graph $\mathcal{G}$ and a nonce $\mu$, a *sub-receipt (over $\mathcal{G}$, for payment $\mu$)* is a pair $(\pi,\lambda)$ signed by $P_n$ such that $\pi$ is a payment route over $\mathcal{G}$ (for payment $\mu$), and $\lambda$ is a non-increasing sequence of positive integers, such that $|\lambda| = |\pi|$. We will denote it with $\wr\pi,\lambda\int$. A *payment report for* $\mu$ is a set $\mathcal{S}$ of sub-receipts for $\mu$ such that $\pi$ identifies a member of $\mathcal{S}$ uniquely, i.e.: $(\wr\pi,\lambda\int \in \mathcal{S}$ and $\wr\pi,\lambda'\int \in \mathcal{S})$ implies $\lambda = \lambda'$. For example, $\alpha,\beta$, and $\gamma$ in Sec. IV-A are sub-receipts, and the set $\{\beta,\gamma\}$ (see Eq. (5)) is a payment report (except that in that informal description we omitted the nonces). For a payment report $\mathcal{S}$ a sub-receipt $\wr(\pi,\lambda\int$ is a *leader of $\mathcal{S}$ at node $P$* if $P$ appears on $\pi$ at some position $i$, and for every $\wr\pi',\lambda'\int \in \mathcal{S}$ we have that $\lambda[i] \geq \lambda'[i]$. This notion was already discussed in Sec. IV, where in particular we said that the leader of a payment report $\{\alpha',\gamma\}$ (on Eq. (5)) is $\gamma$. In normal cases (i.e. if $P_n$ is honest) the leader is always unique, and is equal to the *last* sub-receipt of a from $\wr(\pi||\sigma'),\lambda'\int$ signed by $P_n$, however in general this does not need to be the case. When we talk about *the* leader of $\mathcal{S}$ at $P$ we mean $\wr(\pi||P||\sigma),\lambda\int$ that is the smallest according to some fixed linear ordering.

As already mentioned in Sec. II-B, EthNA is constructed using "fraud proofs". Formally, a *fraud proof (for $\mu$)* is a payment report $\mathcal{Q}$ for $\mu$ of a form $\mathcal{Q} = \{\wr(\sigma||\pi_i),\lambda_i\int\}_{i=1}^m$, where all the $\pi_i[1]$'s are pairwise distinct[3], such that the following condition holds: $\max_{i:=1,\ldots,m} \lambda_i[|\sigma|] < \sum_{i:=1}^m \lambda_i[|\sigma|+1]$ For an example of a fraud proof (with nonce missing from the picture) see Eq. (7). If EthNA has arity at most $\delta$ (see Sec. III) then we require that $m \leq \delta$. Informally speaking this conditions means simply that in $\mathcal{Q}$ the largest label of $\sigma$ is at smaller than the sum of all labels of $\sigma$'s children. If none of the subsets of a payment report $\mathcal{S}$ is a fraud proof then we say that $\mathcal{S}$ *is consistent*. As we show later (cf. Lemma 1) if $P_n$ is honest then $\mathcal{S}$ is always consistent.

*1) Size of the fraud proofs.:* Note that the description of set $\mathcal{Q}$ as defined above can be quite large (it is of size $O(\delta \cdot (\ell + \kappa))$, where $\delta$ is EthNA's arity, $\ell$ is the maximal length of payment routes, and $\kappa$ is the security parameter (we need this to account for the signature size). Luckily, there is a simple way the "compress" it to $O(\delta \cdot \kappa)$ (where $\kappa$ is the security parameter) by exploiting the fact that the only values that are needed to prove cheating are the positions on the indices $|\sigma|$ and $|\sigma|+1$ of the $\lambda$'s. We describe further compression ideas in Appx. E2.

### A. The actual protocol

The formal description of EthNA appears on Figs. 3 (it uses a sub-routine algorithm $\mathsf{Add}_\Phi$ that we describe below). Let us first comment on the types of messages that are sent within the protocol (see also the cheat sheet on Fig. 4 on p. 16 in the appendix). The parties communicate with each other only via the state channels (except of the first "invoice" message sent from $P_n$ to $P_1$). The messages that are used are: "push" to push a sub-payment (the corresponding message sent by the channel to the other party is "pushed"), "acknowledge" to acknowledge a sub-payment (the corresponding message is "acknowledged"), and "fraud-signal" to signal fraud (the corresponding message is "fraud-signalled"). The messages sent by $P_1$ to the RVM are either "acknowledge" (if everything went ok), or "fraud-signalled". Let us now describe the individual procedures. In our description we make several simplifications, e.g., we ignore some special, but rare cases (like $P_n$ not acknowledging some payments at all). Let $\mathcal{G}$ be the channel graph. As already mentioned before, the main idea is to let the sender $P_n$ perform the payment aggregation herself, and to "punish" her in case she cheats. Cheating will be proven using the fraud proofs defined above. Of course, if $P_n$ is honest then nobody can produce a valid fraud proof (we prove it in Lemma 1). Therefore the punishment for cheating can be arbitrarily severe. As explained before, in our settings we simply let a fraud proof serve as a receipt (see Eq. (1)) that all the coins were transferred.

Going a bit more into the details, the protocol for every new payment $\mu$ of value $v$ starts when $P_1$ and $P_n$ receive "env-send" and "env-receive" messages from the environment (with parameters $v,\mu$, and $t$, where $t$ specifies the maximal

[3] In other words: the paths in $\mathcal{Q}$ form a tree with exactly one vertex $\pi$ that has more than on child.

time when the payment has to be completed). As a reaction $P_n$ sends a signed pair $(\mathsf{invoice}, \mu, u, t)$ (called an *invoice*) to $P_1$. If later $P_1$ obtains a fraud proof $\mathcal{Q}$ for $\mu$ then $(\mathsf{invoice}, \mu, u, t)$ together with $\mathcal{Q}$ will serve as a receipt that *all* the $u$¢ were transferred in payment $\mu$. This is ok, since the protocol is constructed in such a way that $P_n$ never pushes more coins than $u$ to her neighbors (within payment $\mu$). Let us now provide some more information on how the coins are pushed and acknowledged.

*1) Pushing payments.:* Initially no coins have been transferred within payment $\mu$, so $P_1$ holds all $u$ of them. Pushing payments is done in a recursive way. Suppose $P$ holds some number $v$ of coins that were pushed to $P$ via some path $\pi$ (in case $P = P_1$ this path is simply $\langle (P_1, \mu) \rangle$). Let $t$ be the deadline until this payment has to be completed. Party $P$ holds a variable $\mathcal{S}^\pi$ that she uses for bookkeeping purposes. Variable $\mathcal{S}^\pi$ contains a payment report and is initially empty. Upon receiving a message $\mathsf{env\text{-}push}(\pi || (P', \mu'), v', t')$ from the environment party $P$ pushes $v'$¢ to a neighbor $P'$ of hers. This is done by sending a push message in the state channel $P \multimapboth P'$ and blocking $v'$¢ of $P$ in it. This message comes with a parameter $(\pi || (P', \mu'))$ (where $\mu'$ is some fresh nonce) and a deadline $t' < t - \Delta$ until when this payment has to be completed. As in the case of Lightning (see Sec. II-A) this message is immediate since it imposes no commitments on $P'$ . Before describing how the payments are acknowledged by the intermediaries, and how the final receipt is produced by $P_1$ let us present the procedure for the receiver $P_n$.

*2) Payment acknowledgment by $P_n$.:* For every payment $\mu$ party $P_n$ maintains a payment tree $\Phi^\mu$ that is initially empty. Payment trees were already discussed in Sec. IV-A (in particular: Eqs. (2)–(4) on p. 6 contain examples of such trees). For a formal definition consider some fixed $\mu$ and $\mathcal{G}$. During the execution of ETHNA for $\mathcal{G}$ and $\mu$, several sub-payments are delivered to $P_n$. Let $\pi^1, \ldots, \pi^t$ denote the consecutive paths over which these sub-payments go (of course they need to be distinct), and let $v^i \in \mathbb{Z}_{>0}$ be the amount of coins transmitted with each $\pi^i$. Let $\mathcal{R} := \{(\pi^i, v^i)\}_{i=1}^t$. Formally a *payment tree* $\mathsf{tree}(\mathcal{R})$ is a pair $(T, \mathcal{L})$, where $T$ is the set of all prefixes of the $\pi^i$'s, i.e., $T := \bigcup_i \mathsf{prefix}(\pi^i)$, (for the standard notation for the trees see Appx. B). If ETHNA has arity $\delta$ then the arity of $T$ in every node $\pi || (P, \mu)$ is at most $\delta$. Then for every $\pi \in T$ we let $\mathcal{L}(\pi) := \sum_{i:\pi \in \mathsf{prefix}(\pi^i)} v^i$. In other words: every payment route prefix $\pi$ gets labeled by the arithmetic sum of the value of the payments that were "passed through it". Obviously, the label $\mathcal{L}(\varepsilon)$ of the root node of $\mathsf{tree}(\mathcal{R})$ is equal to the sum of all $v^i$'s, and hence it is equal to the total number of coins transferred by the sub-payments in $\mathcal{R}$. We also have that for every payment route prefix $\sigma$ $\mathcal{L}(\sigma) = \sum_{\pi \text{ is a child of } \sigma} \mathcal{L}(\pi)$. It is also easy to see that $\mathsf{tree}(\mathcal{R})$ can be constructed "dynamically" by processing elements of $\mathcal{R}$ one after another. More precisely, this is done as follows. We start with an empty tree $\Phi$, and then iteratively apply the algorithm $\mathsf{Add}_\Phi$ (see Alg. 1) for $(\pi^1, v^1), (\pi^2, v^2), \ldots$.
From the construction of the algorithm it follows immediately that of $P_n$ starts with $\Phi$ being an empty tree, and then iteratively applies $\mathsf{Add}_\Phi$ to $(\pi^i, v^i)$'s for $i = 1, \ldots, t$, then the final state of $\Phi$ is equal to $\mathsf{tree}(\mathcal{R})$. For example, if $P_n$ applies this procedure to the situation on Fig. 1c she obtains the trees depicted on Eqs. (2)–(4). For a payment tree $\Phi = (T, \mathcal{L})$ and $\pi \in T$ define $\mathsf{labels}(\Phi, \pi)$ as the sequence (of length $|\pi|$) of all labels leading from the tree root to $\pi$, i.e., for every $i = 1, \ldots, |\pi|$ let $\mathsf{labels}(\Phi, \pi)[i] := \mathcal{L}(\pi|_i)$. The following lemma (whose proof appears in Appx. C) shows that if $P_n$ applies the $\mathsf{Add}_\Phi$ algorithm correctly, then the resulting sets $\mathcal{S}$ are never inconsistent (and hence no "fraud proof" will ever be produced against an honest $P_n$).

**Algorithm 1:** $\mathsf{Add}_\Phi(\pi, v)$

**assumption:** $v \in \mathbb{Z}_{>0}$ and $\pi \notin T$

*This algorithm operates on a global state $\Phi = (T, \mathcal{L})$. Its side effect is a change of the global state.*

**for** $j = 1, \ldots, |\pi|$ **do**
  **if** $\pi|_j \in T$ **then**
    **let** $\mathcal{L}(\pi|_j) := \mathcal{L}(\pi|_j) + v$
  **else**
    **let** $T := T \cup \{\pi|_j\}$ **let** $\mathcal{L}(\pi|_j) := v$

**output** $\langle \mathcal{L}(\pi[1]), \ldots, \mathcal{L}(\pi_{|\pi|}) \rangle$ *(the labels on path $\pi$)*

**Lemma 1.** *Suppose a party $P_n$ executes $\mathsf{Add}_\Phi$ multiple times (for some payment $\mu$, and starting from $\Phi = \emptyset$) and signs every output. Let $\mathcal{S}$ be the set of sub-receipts signed by party $P_n$ during the execution of the $\mathsf{Add}_\Phi$ algorithm. Then $\mathcal{S}$ is consistent.*

Party $P_n$ waits for push requests. Each such a message arrives from one of $P_n$'s neighbors in $\mathcal{G}$ and is transmitted via some state channel $P \multimapboth P_n$ of $\mathcal{G}$ . They all come with parameters $\pi, v$, and $t$, where $\pi$ is a payment route (starting with $(P_1, \mu)$ and with $P_n$ appearing as its last element), $v$ is the number of pushed coins, and $t$ is a timeout for this sub-payment. The receiver now decides on the number $v' \leq v$ of coins that she is willing to accept from this sub-payment. She then runs $\mathsf{Add}_{\Phi^\mu}(\pi, v')$. Recall that this results in updating state $\Phi^\mu$ and producing an output $\lambda$ (equal to the labels on path $\pi$ *after* updating the state). Party $P_n$ acknowledges the sub-receipt of $v'$¢ by sending a signed pair $\wr \pi, \lambda \int$ back to the state channel $P \multimapboth P_n$, and claims $v'$¢ from the amount locked in $P \multimapboth P_n$ by $P$. As in Lightning, this message is not immediate. Party $P$ learns $\wr \pi, \lambda \int$ within 1 hour. Observe that from the fact that $\mathcal{L}(\sigma) = \sum_{\pi \text{ is a child of } \sigma} \mathcal{L}(\pi)$ (see above) we get that $\lambda[1]$ is equal to the sum of all the coins that were so far transmitted to $P_n$ within payment $\mu$.

*3) Payment acknowledgment by the intermediaries.:* Let us now go back to party $P$ that pushed some coins to $P'$ via channel $P \multimapboth P'$ and waits receive acknowledgment from $P'$ (via the same channel). For a moment suppose the $P$ is an intermediary. Let $P''$ be the party that earlier pushed $v$¢ to $P$. In the most likely case $P$ receives some $\wr \phi, \lambda \int$ (with $\pi$ being a prefix of $\phi$). In this case she adds $\wr \phi, \lambda \int$ to $\mathcal{S}^\pi$. The state channel is constructed in such a way that $\phi[|\pi| + 1]$ coins (from those that were locked by $P$) are transferred to $P'$, while the rest goes back to $P$. Once $P$ gets an $(\mathsf{env\text{-}acknowledge}, \pi))$ message (this can only happen if there are no open push request for sub-payments of $\pi$) she looks at $\mathcal{S}^\pi$. If it is consistent then she finds the leader $\wr(\pi||\widehat{\sigma}), \widehat{\lambda}\int$ of this set at $P$. Party $P$ acknowledges $\pi$ by sending back $\wr(\pi||\widehat{\sigma}), \widehat{\lambda}\int$ to $P'' \multimap\!\!\circ P$. At the same time she claims $\lambda[|\pi|]¢$ from the coins locked in this channel. Observe that since $\mathcal{S}^\pi$ is consistent, thus $\lambda[|\pi|]$ is at least as large as the sum $\sum_{\wr(\pi||\sigma),\lambda\int \in \mathcal{S}^\pi} \lambda[|\pi|+1]$, and this sum is exactly equal to the total number of coins that $P$ "payed" to the parties to which she pushed this payment. Hence she never looses money.

The second option is that $\mathcal{S}^\pi$ is inconsistent. Let $w$ be the fraud proof. Party $P$ simply sends $w$ back to $P''$ (over the channel $P'' \multimap\!\!\circ P$). Think of it as "throwing an exception" in recursive application of "pushing" procedure. In some sense $w$ is a "wild card" that allows to claim *all* the $v¢$ that were pushed to a party that presents it. Since it works "universally" no honest party looses money. In particular, although $P''$ has to accept that all the coins were "transferred" to $P$, she can later use the same $w$ to claim all the coins that were blocked by the party that pushed this payment to her.

*4) Receipt by the sender.:* For the sender $P_1$ the protocol works similarly, except that $P_1$ does not "push" messages back, but simply outputs them as a receipt. More precisely if $\mathcal{S}^{(P_1,\mu)}$ is consistent and no fraud proof have been received, then let $\wr\widehat{\phi}, \widehat{\lambda}\int$ be the leader of $\mathcal{S}^{(P_1,\mu)}$ at $P_1$. In this case case party $P_1$ concludes that $\widehat{\lambda}[1]¢$ were transferred, and $\wr\widehat{\phi}, \widehat{\lambda}\int$ is the receipt. Otherwise let $w$ be the fraud proof. Then $P_1$ concludes that all $u¢$ were transferred and a pair $(w, \wr\mu, u\int)$ is the receipt (the "$\wr\mu, u\int$" component is needed to demonstrate what was the maximal transmitted value that $P_n$ agreed for).

### B. Analysis

We already argued informally about ETHNA's security while presenting it. Formal security analysis of this protocol is given in the proof of the following lemma.

**Lemma 2.** *Assuming that the underlying signature scheme is existentially unforgeable under a chosen message attack, the* ETHNA *is a Non-Atomic Payment Splitting protocol for every channel graph* $\mathcal{G} = (\mathcal{P}, \mathcal{E}, \Gamma)$.

*Proof sketch.* We show that the functionality and security requirements from Sec. V hold in presence of an arbitrary adversary $\mathsf{Adv}$ and any admissible $\mathsf{Env}$. The functionality requirements follow easily from the construction of the protocol, so it remains is to argue about the security requirements. First, it is easy to see that at a moment if some party $P$ gets a "fraud proof" (either by finding it herself, or because of receiving it from some other party) then her security is guaranteed. This is because if such a $P$ is an intermediary, then she can claim all the coins that were pushed to her (and she never pushes forward more coins than this), so balance neutrality holds for her. If $P$ is the sender, then she simply uses this proof as her receipt that *all* the coins were transferred, and therefore fairness for the sender holds. On the other hand, as proven in Lemma 1, if $P_n$ is honest then no fraud proof will ever be produced, so this mechanism constitutes no risk to the fairness of the receiver. Hence, for every $P$ we can assume that she does not get a fraud proof. For $P \neq P_n$ this means that the part of the payment report that she gets is always consistent, which means that what she transfer in the sub-payments to the other parties is at most what she gets herself (for the party that pushed a given sub-payment to her). Hence balance neutrality for the intermediaries holds. Using a very similar argument we can show that fairness for the sender is also provided. Fairness for the receiver follows from the fact that in the payment tree the label in the root is equal to the sum of the labels in the leaves (which is equal total amount of coins that $P_n$ "looses" in the channels). The complete proof is moved to Appx. D. $\square$

When analyzing security of the off-chain protocols one typically considers the *optimistic* scenario (when the parties are cooperating) and the *pessimistic* one when the malicious parties slow down the execution.

*Time complexity:* In the optimistic case the payments are almost immediate. It takes 1 for a payment to be pushed, and 2 rounds to be acknowledged (since for acknowledgment the messages sent to state channels are not immediate). Hence in the most optimistic case the time for executing a payment is $3 \cdot \ell$ (where $\ell$ is the depth of the payment tree). During the acknowledgment every malicious party can delay the process by time at most $\Delta$. Hence, the maximal pessimistic time is $(1 + \Delta) \cdot \ell$.

*Blockchain costs:* The second important measure are the blockchain costs, i.e., the fees that the parties need to pay. Below we provide a "theoretical" analysis of such costs (by this we mean that we abstract away from practical features of Ethereum). For results of concrete experiments see VII-1. Note that in the optimistic case these the only costs are channel opening and closing, and hence they are independent of the tree depth and of its arity. In the pessimistic case all the messages in state channels need to be sent "via the blockchain". This is especially unpleasant, since its not clear whose fault it was, and who should pay the fees (in other words: this fault is "non-uniquely attributable" and can lead to "griefing", see, e.g. [10, 13] for an explanation of these notions). Let us consider two cases. In the first case there is no fraud proof. Then, the only message that is sent via the blockchain is $\mathsf{acknowledge}(\wr\phi, \lambda\int)$, which has size linear $O(\ell+\kappa)$ (where $\ell$ is as above, and $\kappa$ is the security parameter, and corresponds to space needed to store a signature). The situation is a bit different if a fraud proof appears. As remarked in Sec. VI-1 the size of a fraud proof is $O(\delta \cdot (\ell+\kappa))$, where $\delta$ is ETHNA's arity, $\ell$ is the maximal length of payment routes, and $\kappa$ is the security parameter. Note that the fraud proof is "propagated", i.e., even if a given intermediary decided to keep its arity small (i.e.: not to split her sub-payments in too many sub-payments), she may be forced to pay fees that depend on some (potentially larger) arity. This could result in griefing attacks and it is the reason why we introduced a global bound on the arity. There are many ways around this. First of all, we could modify the protocol in such a way that the fraud proofs by $P_n$ are posted directly in a smart contract on a blockchain in such a way that all the other parties do not need to re-post it, and can just refer to it. This would mean that the fees are payed only by the first party that discovers the fraud proof. She could then be compensated from a deposit put aside before the

**(a) The parties**

**Party $P_1$**

Wait to receive messages $(\textsf{env-send}, u, \mu, t)$ from the environment Env. Handle each such a message as follows.

> If in the next round you receive a message $(\textsf{invoice}, \mu, u, t)$ signed by $P_n$ then store it, and execute the *route handling procedure* procedure $\textsf{handle-route}(\langle(P_1, \mu)\rangle, v, t)$ defined below.
> If the output of this procedure is $(\textsf{fraud-signal}, w)$ then send $(\textsf{fraud-signal}, w, \lbag \mu, u \rbag)$ to RVM. Otherwise (i.e. if it was a "acknowledge" message) simply send this output to RVM.

**Party $P_i \in \{P_2, \ldots, P_{n-1}\}$**

Wait to receive messages $(\textsf{pushed}, \pi, v, t)$ from some $C^{P \multimap\!\!\circ P_i}$. Handle each such a request by the route handling procedure $\textsf{handle-route}(\pi, v, t)$ defined below.

> $\underline{\textsf{handle-route}(\pi, v, t)}$
>
> Let $\mathcal{S}^\pi$ be a variable containing a set of sub-receipts that initially is empty, send $(\textsf{env-pushed}, \pi, v)$ to Env and wait for the following messages from Env:
>
> - $(\textsf{env-push}, (\pi||(P', \mu')), v', t')$ — handle each such a message by executing the following *push handling procedure*:
>
>   > $\underline{\textsf{handle-push}((\pi||(P', \mu')), v', t')}$
>   >
>   > Send a message $(\textsf{push}, (\pi||(P', \mu')), v', t')$ to $C^{P_i \multimap\!\!\circ P'}$, and wait to receive one of the following messages from $C^{P_i \multimap\!\!\circ P'}$:
>   >
>   > – $(\textsf{acknowledged}, (\pi||(P', \mu')), \textsf{empty})$ — then send a message $(\textsf{env-acknowledged}, (\pi||(P', \mu')), 0)$ to Env,
>   >
>   > – $(\textsf{acknowledged}, \lbag \psi, \lambda \rbag)$, where $\psi$ is such that $(\pi||(P', \mu'))$ is a prefix of $\psi$ — then store $\lbag \psi, \lambda \rbag$ in $\mathcal{S}^\pi$ by letting $\mathcal{S}^\pi := \mathcal{S}^\pi \cup \{\lbag \psi, \lambda \rbag\}$. Let $\widehat{v} := \lambda[|\pi| + 1]$. Send $(\textsf{env-acknowledged}, (\pi||(P', \mu')), \widehat{v})$ to Env, and
>   >
>   > – $(\textsf{fraud-signalled}, w)$ — then store $w$ and send a message $(\textsf{env-acknowledged}, (\pi||(P', \mu')), v')$ to Env.
>   >
>   > Then end the $\textsf{handle-push}$ procedure.
>
> - $(\textsf{env-acknowledge}, \pi)$ — do the following
>
> – If you stored $(\textsf{fraud-signalled}, w)$ (for some $(P', \mu')$) or if $\mathcal{S}^\pi$ is inconsistent and $w$ is the fraud proof — then output $(\textsf{fraud-signal}, w)$.
>
> – Otherwise: if $\mathcal{S}^\pi$ is empty then output $(\textsf{acknowledge}, \pi, \textsf{empty})$.
>
> – Otherwise let $\lbag \psi, \lambda \rbag$ be the leader of $\mathcal{S}^\pi$ at $\widetilde{P}$, where $\widetilde{P}$ is the last party on $\pi$. Output $(\textsf{acknowledge}, \pi, \lbag \psi, \lambda \rbag)$.
>
> After producing the output end the $\textsf{handle-route}$ procedure.

After this procedure terminates send its output back to $C^{P \multimap\!\!\circ P_i}$.

**Party $P_n$**

Wait to receive messages $(\textsf{env-receive}, u, \mu, t)$ from the environment Env. Handle each such a request as follows.

> First, sign a message $(\textsf{invoice}, \mu, u, t)$ and send it to $P_1$. Let $\mathcal{S}^\mu$ be a variable containing a payment report that initially is empty, Wait to receive messages $(\textsf{pushed}, \pi, v, t)$ from some $C^{P \multimap\!\!\circ P_n}$. Handle each such a message as follows. Once you receive it send $(\textsf{env-pushed}, \pi, v, t)$ to Env and wait to receive $(\textsf{env-acknowledge}, \pi, v')$ from Env. Once this happens, execute $\textsf{Add}_{\mathcal{S}^\mu}(\pi, v')$. Let $\lbag \pi, \lambda \rbag$ be the output of this procedure. Send a message $(\textsf{acknowledge}, \lbag \pi, \lambda \rbag)$ to $C^{P \multimap\!\!\circ P_n}$.

**(b) The state channel machine $C^{P_i \multimap\!\!\circ P_j}$**

Recall that the values of registers $C^{P_i \multimap\!\!\circ P_j}.\textsf{cash}(P_i)$ and $C^{P_i \multimap\!\!\circ P_j}.\textsf{cash}(P_j)$ were pre-loaded before the execution started. Wait for the messages $(\textsf{push}, (\pi||(P, \mu)||(P', \mu')), v, t)$ from $P \in P_i \multimap\!\!\circ P_j$ (where $P' := \textsf{other-party}(P)$) such that (a) $t \leq \tau + \Delta$ (where $\tau$ is the current time), (b) $(\pi||(P, \mu)||(P', \mu'))$ is a payment route prefix, (c) $v \leq C^{P_i \multimap\!\!\circ P_j}.\textsf{cash}(P)$, and (d) you have not previously received a push request with the same parameters. Upon receiving such a message let $C^{P_i \multimap\!\!\circ P_j}.\textsf{cash}(P) := C^{P_i \multimap\!\!\circ P_j}.\textsf{cash}(P) - v$, and send $(\textsf{pushed}, (\pi||(P, \mu)||(P', \mu')), v, t)$ to $(P', \mu')$. Then wait until one of the following happens:

- you receive a message $(\textsf{acknowledge}, \lbag \psi, \lambda \rbag)$ from $(P', \mu')$ where $\psi$ is a route with a prefix $(\pi||(P, \mu)||(P', \mu'))$ — then let $\widehat{v} := \lambda[|\pi| + 2]$. Let $C^{P_i \multimap\!\!\circ P_j}.\Gamma(P') := C^{P_i \multimap\!\!\circ P_j}.\Gamma(P') + \widehat{v}$, and $C^{P_i \multimap\!\!\circ P_j}.\Gamma(P) := C^{P_i \multimap\!\!\circ P_j}.\Gamma(P) + v - \widehat{v}$, then send $(\textsf{acknowledged}, \lbag \psi, \lambda \rbag)$ to $P$,
- you receive a message $(\textsf{fraud-signal}, w)$ from $P'$ where $w$ is an fraud proof — then let $C^{P_i \multimap\!\!\circ P_j}.\Gamma(P') := C^{P_i \multimap\!\!\circ P_j}.\Gamma(P') + v$ and forward this message to $P$,
- time $t$ comes — then let $C^{P_i \multimap\!\!\circ P_j}.\textsf{cash}(P) := C^{P_i \multimap\!\!\circ P_j}.\textsf{cash}(P) + v$ and send a message $(\textsf{acknowledged}, (\pi||(P, \mu)||(P', \mu')), \textsf{empty})$ to $P$.

**(c) The receipt verification machine RVM**

Wait for one of the following messages from $P_1$:

- $(\textsf{acknowledge}, ((P_1, \mu), \textsf{empty}))$ — then output $(\mu, 0)$.
- $(\textsf{acknowledge}, \lbag ((P_1, \mu)||\psi), (v||\lambda) \rbag)$ — then output $(\mu, v)$.
- $(\textsf{fraud-signal}, w, \lbag \mu, u \rbag)$, where $w$ is a fraud proof for some route $\sigma$, and $\nu$ is the nonce in the first element of $\sigma$ — then output $(\mu, u)$.

For a given $\mu$ output a pair that contains it only once (i.e. after outputting $(\mu, v)$ ignore all the future calls that would lead to outputting $(\mu, v')$ for some $v'$).

Fig. 3: The ETHNA protocol

protocol starts. Moreover, the proof size can be significantly reduced using techniques described in Appx. E2.

## VII. PRACTICAL ASPECTS

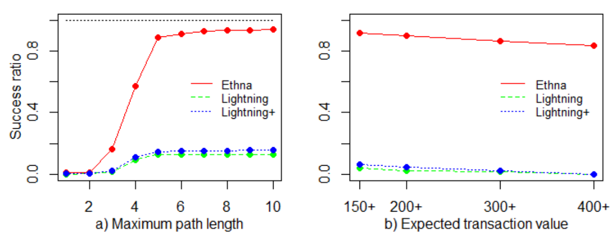
In this section we provide information about practical experiments of ETHNA implementation. The source code is available at github.com/Sam16450/NAPS-EthNA.

*1) Implementation in Solidity:* We implemented a simple version of ETHNA in Solidity. Compared to the version described in this paper, this preliminary version lacks the ability to add nonces. The following table summarizes the execution costs *in terms of thousands of gas*, and depending on the arity and the maximal path length.

| arity | path length | constru-ctor | close | addState | addChea-tingProof | add-Comple-ted-Transa-ction | close-Disagree-ment |
|---|---|---|---|---|---|---|---|
| 5 | 10 | 2,391 | 14 | 93 | 1,053 | 155 | 14 |
| 5 | 5 | 2,249 | 14 | 94 | 871 | 145 | 14 |
| 2 | 5 | 2,088 | 14 | 93 | 779 | 145 | 14 |
| 2 | 3 | 2,191 | 14 | 93 | 590 | 140 | 14 |

Above, `constructor` denotes the procedure for deploying a channel, `close` corresponds to closing a channel without disagreement, `addState` is used to register the balance in case of disagreement, `addCheatingProof` is used to add a fraud proof, `addCompletedTransaction` — to add a sub-receipt when no cheating was discovered, and `closeDisagreement` – to finally close a channel after disagreement. Assuming cost 1,000 gas = \$0.00018 (according to `ethgasstation.info` this is the average rate as of Jan 21st, 2020) we get that the most expensive action (deploying a channel, `addCheatingProof`) costs \$0.43.

*2) Simulation results:* Although routing algorithms are not the main topic of this work, we also performed some experiments with a routing algorithm built on top of ETHNA. In our experiments we used the following approach. The network graph was taken from the Lightning network (from the website gitlab.tu-berlin.de/rohrer/discharged-pc-data) with aprrox. $6,000$ nodes and $30,000$ channels. Channel's capacities are chosen according to the normal distribution $\mathcal{N}(200, 50)$. Each transaction was split by applying the following rules. The sender and the intermediaries look at the channel graph and search for the set $\mathcal{X}$ of shortest paths that lead to the receiver (and have different first element). Then they split the payment in values that are proportional to the capacity of the first channel in the path. In our simulations we performed $100,000$ transaction. The results are below.




Above, the "success ratio' denotes the probability of success of an average payment. "Lightning" refers to standard Lightning routing, and "Lightning+" to the Lightning algorithm that attempts to push payments multiple time. Transaction values are chosen uniformly from set $(x_0, x_1)$, where in (a) we have $(x_0, x_1) = (10, 500)$ and in (b) we have $x_0 = 150, 200, 300,$ and 400 and $x_1$ always set to 500. Our experiments show that for such large payments even this simple routing algorithm for ETHNA works much better than Lightning. We leave designing better routing algorithms for ETHNA as an direction for future work.

## ACKNOWLEDGMENTS

This work was supported by the Foundation for Polish Science Team grant 2016/1/4 ) co-financed with the support of the EU Smart Growth Operational Programme (PO IR) and by the *Ethereum Foundation* grant FY18-0023.

## REFERENCES

[1] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostáková, M. Maffei, P. Moreno-Sanchez, and S. Riahi. "Bitcoin-Compatible Virtual Channels". In: *IACR Cryptol. ePrint Arch.* (2020).

[2] V. K. Bagaria, J. Neu, and D. Tse. "Boomerang: Redundancy Improves Latency and Throughput in Payment Networks". In: *CoRR* (2019). arXiv: 1910.01834.

[3] R. Canetti. "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*. 2001.

[4] M. M. T. Chakravarty, S. Coretti, M. Fitzi, P. Gazi, P. Kant, A. Kiayias, and A. Russell. "Hydra: Fast Isomorphic State Channels". In: *IACR Cryptol. ePrint Arch.* (2020).

[5] J. Coleman, L. Horne, and L. Xuanji. *Counterfactual: Generalized State Channels*. https://l4.ventures/papers/statechannels.pdf. 2018.

[6] C. Dannen. *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners*. 1st. USA, 2017.

[7] C. Decker and R. Wattenhofer. "A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels". In: *Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings*. 2015.

[8] Diar. *Lightning Strikes, But Select Hubs Dominate Network Funds*. https://diar.co/volume-2-issue-25/. (Accessed on 01/20/2020).

[9] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková. "Multi-party Virtual State Channels". In: *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*. 2019.

[10] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski. "Perun: Virtual Payment Hubs over Cryptocurrencies". In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. 2019.

[11] S. Dziembowski, S. Faust, and K. Hostáková. "General State Channel Networks". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. 2018.

[12] L. Eckey, S. Faust, K. Hostáková, and S. Roos. "Splitting Payments Locally While Routing Interdimensionally". In: *IACR Cryptol. ePrint Arch.* (2020).

[13] C. Egger, P. Moreno-Sanchez, and M. Maffei. "Atomic Multi-Channel Updates with Constant Collateral in Bitcoin-Compatible Payment-Channel Networks". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. 2019.

[14] M. Green and I. Miers. "Bolt: Anonymous Payment Channels for Decentralized Currencies". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. 2017.

[15] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais. "SoK: Off The Chain Transactions". In: *IACR Cryptology ePrint Archive* (2019).

[16] M. Jourenko, K. Kurazumi, M. Larangeira, and K. Tanaka. "SoK: A Taxonomy for Layer-2 Scalability Related Protocols for Cryptocurrencies". In: *IACR Cryptology ePrint Archive* (2019).

[17] H. A. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten. "Arbitrum: Scalable, private smart contracts". In: *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. 2018.

[18] J. Katz and Y. Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. 2007.

[19] R. Khalil and A. Gervais. "Revive: Rebalancing Off-Blockchain Payment Networks". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. 2017.

[20] A. Kiayias and O. S. T. Litos. "A Composable Security Treatment of the Lightning Network". In: *IACR Cryptology ePrint Archive* (2019).

[21] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. "Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts". In: *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*. 2016.

[22] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi. "Concurrency and Privacy with Payment-Channel Networks". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. 2017.

[23] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei. "Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability". In: *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. 2019.

[24] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei. "Multi-Hop Locks for Secure, Privacy-Preserving and Interoperable Payment-Channel Networks". In: *IACR Cryptol. ePrint Arch.* (2018).

[25] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry. "Sprites and State Channels: Payment Networks that Go Faster Than Lightning". In: *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*. 2019.

[26] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. http://bitcoin.org/bitcoin.pdf. 2009.

[27] O. Osuntokun. *[Lightning-dev] AMP: Atomic Multi-Path Payments over Lightning*. https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html. (Accessed on 01/19/2020). 2018.

[28] D. Piatkivskyi and M. Nowostawski. "Split Payments in Payment Networks". In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2018 International Workshops, DPM 2018 and CBT 2018, Barcelona, Spain, September 6-7, 2018, Proceedings*. 2018.

[29] J. Poon and V. Buterin. *Plasma: Scalable Autonomous Smart Contracts*. Accessed: 2017-08-10. 2017.

[30] J. Poon and T. Dryja. *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. Draft version 0.5.9.2, available at https://lightning.network/lightning-network-paper.pdf. 2016.

[31] J. Spilman. *[Bitcoin-development] Anti DoS for tx replacement*. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html. (Accessed on 01/20/2020). 2013.

[32] E. Tairi, P. Moreno-Sanchez, and M. Maffei. "A$^2$L: Anonymous Atomic Locks for Scalability and Interoperability in Payment Channel Hubs". In: *IACR Cryptol. ePrint Arch.* (2019).

[33] J. Teutsch and C. Reitwießner. *A scalable verification solution for blockchains*. https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf. 2017.

[34] G. Wood. *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. http://gavwood.com/paper.pdf. 2014.


## APPENDIX

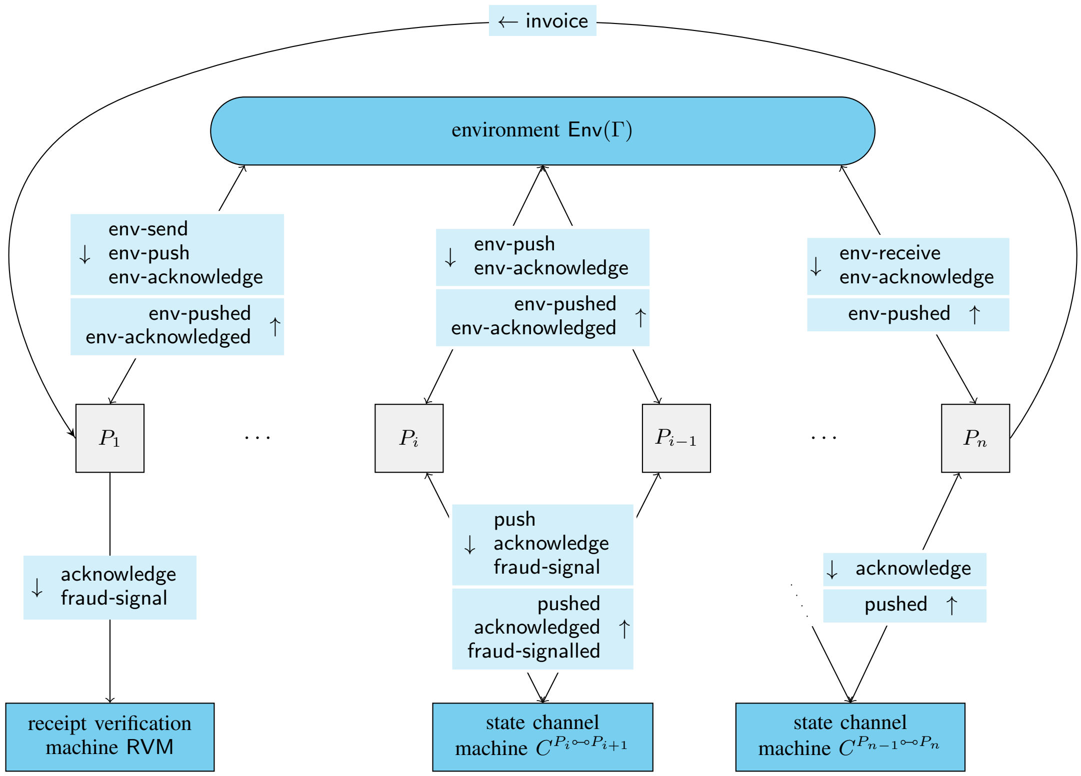
### A. ETHNA's name explanation

We call our protocol ETHNA, in reference to Etna, one of the highest active volcanoes in Europe. This is because the coin transfers in ETHNA resemble a lava flood (with large streams recursively bifurcating into small sub-streams). The letter "h" is added so that the prefix "Eth-" is reminiscent of ETH, the symbol of Ether (the currency used in Ethereum), and "NA" stands for "Non-Atomic".

### B. Standard function and string notation

By $[a_i \mapsto x_1, \ldots, a_m \mapsto x_m]$ we mean a function $f : \{a_i, \ldots, a_m\} \to \{x_1, \ldots, x_m\}$ such that for every $i$ we have $f(a_i) := x_i$. Let $A$ be some finite alphabet. Strings $\delta \in A^*$ are frequently denoted using angle brackets: $\delta = \langle \delta_1, \ldots, \delta_m \rangle$. Let $\delta$ be a string $\langle \delta_1, \ldots, \delta_n \rangle$. For $i = 1, \ldots, n$ let $\delta[i]$ denote $\delta_i$. Let $\varepsilon$ denote an empty string, and "$||$" denote concatenation of strings. We overload this symbol, and write $\delta||a$ and $a||\delta$ to denote $\delta||\langle a \rangle$ and $\langle a \rangle||\delta$, respectively (for $\delta \in A^*$ and $a \in A$). For $k \leq n$ let $\delta|_k$ denote $\delta$'s prefix of length $k$. A set of prefixes of $\delta$ is denoted prefix$(\delta)$ (note that it includes $\varepsilon$).

We define trees as prefix-closed sets of words over some alphabet $A$. Formally, a *tree* is a subset $T$ of $A^*$ such that for every $\delta \in T$ we have that any prefix of $\delta$ is also in $T$. Any element of $T$ is called a *node* of this tree. For two nodes $\delta, \beta \in T$ such that $\beta = \delta||a$ (for some $a$) we say that $\delta$ is the *parent* of $\beta$, and $\beta$ is a *child* of $\delta$. A *labeled tree over $A$* is a pair $(T, \mathcal{L})$, where $T$ is a tree over $A$, and $\mathcal{L}$ is a function from $T$ to some set of *labels*. For $\delta \in T$ we say that $\mathcal{L}(\delta)$ is the *label of* $\delta$.




## Message syntax

**Types of variables**

- $v$ — a positive integer denoting amounts of coins,
- $\mu$ — a nonce,
- $\pi$ — payment path prefix over $\mathcal{G}$, and
- $t$ — time.

**Messages sent and received by** Env

The environment Env sends the following messages to the parties:

- (env-send, $v, \mu, t$) (this message is sent only to $P_1$),
- (env-receive, $v, \mu, t$),
- (env-push, $\pi, v, t$), and
- (env-acknowledge, $\pi$).

The environment Env also receives the following messages from the parties:

- (env-pushed, $\pi, v, t$), and
- (env-acknowledged, $\pi, v$).

**Messages exchanged between the parties**

Party $P_n$ sends to party $P_1$ a message:

- (invoice, $\mu, u, t$) (singed by $P_n$).

**Messages exchanged between the parties and the state channel machines**

The parties send the following messages to the state channel machines:

- (push, $\pi, v, t$),
- (acknowledge, $R$), where $R$ is either equal to ($\pi$, empty) (where "empty" is a keyword) or it is equal to $\wr \psi, \lambda \int$, where $(\psi, \lambda)$ is a sub-receipt over $\mathcal{G}$, and
- (fraud-signal, $w$), where $w$ is an fraud proof,

The state channel machines send the following messages to the parties:

- (acknowledged, $R$), where $R$ is as above, and
- (fraud-signalled, $w$), where $w$ is an fraud proof.

**Messages send by $P_1$ to** RVM

Party $P_1$ sends the following messages to the receipt verification machine RVM:

- (acknowledged, $R$), where $R$ is as above, and
- (fraud-signal, $w, \wr \mu, u \int$), where $w$ is a fraud proof.

Fig. 4: The flow of messages exchanged in the system, and their syntax.

### C. Proof of Lemma 1

Take an arbitrary payment route prefix $\sigma$ and an arbitrary set $\mathcal{Q} \subseteq \mathcal{S}$ that has a form $\mathcal{Q} = \{\wr(\sigma||\pi_i), \lambda_i\int\}_{i=1}^{m}$. Without loss of generality assume paths in $\mathcal{Q}$ are sorted according to the time by which the paths in this set were signed (starting from the first). From the fact that in the Add algorithm the values in the labels can only increase we get that

$$\max_{i=1,\dots,m} \lambda_i[|\sigma|] = \lambda_m[|\sigma|].$$

From the fact that $\mathcal{L}(\sigma) = \sum_{\pi \text{ is a child of } \sigma} \mathcal{L}(\pi)$ (see Sec. VI-A2) we know that the time when path $\wr(\sigma||\pi_m), \lambda_m\int$ was signed all the children on $\sigma$ in the tree $T$ were labeled by values that sum up to $\lambda_m[|\sigma|]$. The sum $\sum_{j:=1}^{m} \lambda_i[|\sigma|+1]$ is *at most* equal to this value. This is because (a) it is a *subset* of the set of all children of $\sigma$, and (b) these paths were signed *earlier* than when $\wr(\sigma||\pi_m), \lambda_m\int$ is signed (here we again use the fact that in the Add algorithm the values in the labels can only increase). Altogether we get that

$$\max_{i:=1,\dots,m} \lambda_i[|\sigma|] \geq \sum_{i:=1}^{m} \lambda_i[|\sigma|+1],$$

and hence $\mathcal{Q}$ cannot be a fraud proof (see Sec. VI for the definition of fraud proofs). Therefore $\mathcal{S}$ does not have fraud proofs, and hence it is consistent. □

### D. Proof of Lemma 2

We need to show that the functionality and security requirements from Sec. V hold in presence of an arbitrary adversary Adv and any admissible Env.

The functionality requirements follows easily from the construction of the protocol. Let us now argue about the security requirements. We start with showing the balance neutrality for the intermediaries. Suppose an honest party $P_i \in \{P_2, \dots, P_{n-1}\}$ starts a handle-route$(\pi, v, t)$ procedure (see Fig. 3). During this execution she initiates a number of handle-push procedures. Let us look at the execution of some handle-push$((\pi||(P',\mu)), v', t')$. At the beginning $P_i$ sends a message (env-push, $(\pi||(P',\mu)), v', t'$) to $C^{P_i \multimap\!\!\circ P'}$. As a result, $C^{P_i \multimap\!\!\circ P'}$ removes $v$ coins from $P_i$'s account. From the construction of the state channel machine it is clear that in time $t'+\Delta$ the latest party $P$ receives one of the following messages back from $C^{P_i \multimap\!\!\circ P'}$ (each of them results in transferring back to her account in $C^{P_i \multimap\!\!\circ P'}$ some amount $z$ of coins):

- a message (acknowledged, $(\pi||(P',\mu'))$, empty) — in this case $z = v$,
- a message (acknowledged, $\wr\psi, \lambda\int$) (where $\pi$ is a prefix of $\psi$) — in this case $z$ is equal to the last element of $\lambda$.
- a message (fraud-signalled, $w$) — in this case $z = 0$.

Call $(v-z)$ the *coins gained by* $P_i$ *in effect of the* handle-push *procedure* and denote it with $gained_{P_i}(\pi)$.

The handle-route$(\pi, v, t)$ procedure ends when $P_i$ receives a message (env-acknowledge, $\pi$) from Env (from the construction of Env it follows that this message must be sent by Env in time $t$ the latest). Once this happens, party $P_i$ sends one of the following messages to $C^{P \multimap\!\!\circ P_i}$ (each of them results in transferring to her account in $C^{P_i \multimap\!\!\circ P_i}$ some amount of $y$¢):

- a message (fraud-signal, $w$) — in this case $y = v$,
- a message (acknowledge, $\pi$, empty) — in this case $y = 0$, or
- a message (acknowledged, $\wr\psi, \lambda\int$) — in this case $y = \widehat{v}$, where $\widehat{v}$ is equal to the last element of $\lambda[|\pi|+1]$.

We will call $y$ the *coins lost by* $P_i$ *in effect of the* handle-push *procedure* and denote it with $lost_{P_i}(\pi)$.

**Claim 1.** *For every honest* $P \in \{P_2, \dots, P_{n-1}\}$ *let* $\pi$ *be some payment route such that a* handle-route$(\pi, v, t)$ *procedure has been executed (for some* $v$ *and* $t$*), and let* $\Pi$ *be the set of all payment routes* $(\pi||(P',\mu))$ *such that* handle-push$((\pi||(P',\mu)), v', t')$ *had been executed. Then we have*

$$gained_{P_i}(\pi) \geq \sum_{\pi' \in \Pi} lost_{P_i}(\pi') \tag{8}$$

*Proof.* First, observe that if $P_i$ sends to $C^{P \multimap\!\!\circ P_i}$ a message (fraud-signal, $w$) then Eq. (8) must hold, because in this case $gained_{P_i}(\pi) = v$, while $\sum_{\pi' \in \Pi} lost_{P_i}(\pi') \leq v$ (this follows from the fact that an admissible Env never asks $P_i$ to push more coins in total than $v$, see Fig. 2). Hence, what remains is to consider the case when no cheating was detected by $P_i$ and in particular $\mathcal{S}^{\pi}$ is consistent. Let $\mathcal{S} = \{\wr\phi_i, \lambda_i\int\}_{i=1}^{m}$. From the construction of the protocol we get that

$$gained_{P_i}(\pi) := \lambda(|\pi|),$$

where $\wr\psi, \lambda\int$ is the leader of $\mathcal{S}^{\pi}$ at $P_i$. This, from the consistency of $\mathcal{S}^{\pi}$ is at least equal to

$$\sum_{j:=1}^{m} \lambda_i[|\sigma|+1],$$

which, in turn is equal to $lost_{P_i}(\pi')$. This finishes the proof the claim. □

Let us now go back to the proof of Lemma 2. It is easy to see that for every $P \in \{P_1, \dots, P_{n-1}\}$ we have that

$$net(P) = \sum_{\pi} gained_P(\pi) - \sum_{\sigma} lost_P(\sigma),$$

where the sums are taken over all $\pi$'s such that handle-route$((\pi||P), v, t)$ (for some $v$ and $t$) has been executed, and all $\sigma$'s such that handle-push$((\pi||P||P'), v, t)$ (for some $v, t$, and $P'$) has been executed. Hence, by applying Claim 1 we obtain that $net(P) \geq 0$, and the balance neutrality holds.

To show fairness for the sender observe that the procedure for $P_1$ is very similar to the procedure for the intermediaries. Essentially, the only differences are as follows. First of all $P_1$, instead of receiving an (pushed, $\pi, v, t$) message from a state channel machine, receives an (env-send, $v, \mu, t$) message from Env and (in the next round) a signed message (invoice, $\mu, u, t$) from $P_n$. Secondly, the fraud-signal message has a different syntax (see Fig. 3 (a)). Thirdly, RVM does *not* transfer any coins to $P_1$'s account (in fact, there are not "accounts" in this machine). Instead RVM outputs $(\mu, y)$. Despite of these differences, the proof is essentially the same as the one for the intermediaries. The main difference is that the $gained_{P_1}$ is now defined with respect to the values output by the receipt verification machine RVM. Namely, once this machine outputs $(v, \mu)$ we let

$$\mathit{gained}_{P_1}((P_1, \mu)) := (\mu, v)$$

(while the definition of $\mathit{lost}_{P_1}$ remains as for the other $P_i$'s). We can show that for every $\mu$ the total sum of coins that $P_1$ looses as a result of executing $\mathsf{handle\text{-}route}((P_1, \mu), v, t)$ in his channels with other parties, is not greater than $v'$, where $(\mu, v')$ is the value output by RVM. This, of course, implies that the *total* amount of coins that $P_1$ looses cannot be larger than the value of $\mathsf{transmitted}$. The proof goes along the same lines as above. In particular we use the fact that the $P_1$ cannot loose more coins that $u$ (this follows the construction of $\mathsf{Env}$), and therefore if $P_1$ detects inconsistency, the fairness for $P_1$ is guaranteed to hold, as $P_1$ can always make RVM output $(v, \mu)$, by sending to it the inconsistency proof together with $(\mathsf{invoice}, \mu, u, t)$.

To show fairness for the receiver, consider some nonce $\mu$ such that $P_n$ received a message $(\mathsf{env\text{-}receive}, v, \mu, t)$ from $\mathsf{Env}$ (for some $u$ and $t$). Recall (see Fig. 3 (a)) that $P_n$ constructs a payment tree $\Phi^\mu$ by executing $\mathsf{Add}_{\mathcal{S}^\mu}(\pi, v')$ each time when it receives a message $(\mathsf{env\text{-}acknowledge}, \pi, v')$. By Lemma 1 $\Phi^\mu$ is always consistent. Recall also that $P_n$ sends a message $(\mathsf{acknowledge}, \wr \pi, \lambda \int)$ to $C^{P \multimap P_n}$ (for some $P$). We have that $\lambda[|\pi|] := v'$, and therefore $P_n$ gains $v'$¢ in the channel $C^{P \multimap P_n}$. The following invariant has to holds. Let $S^\mu$ be equal to the total amount of coins that $P_n$ gained this way, and let $\wr \widehat{\psi}, \widehat{\lambda} \int$ be the leader of $\Phi^\mu$ at $P_n$. Then

$$S^\mu = \widehat{\lambda}[n].$$

Hence, no matter what a (potentially malicious) $P_1$ sends to the receipt verification machine RVM, this machine will never output $(v, \mu)$, with $v > S^\mu$. Hence, the fairness for the receiver holds.

Finally, it is also easy to see that the "no money printing" holds for every state channels machine $C^{P_i \multimap P_j}$. This is because each such a machine will add at most $v$¢ to the accounts of $P_i$ and $P_j$, and this will happen only after deducing $v$¢ from an account of one of them. □

### E. Extensions

In this section we show some extensions of ETHNA. Formal proof that such "extended ETHNAs" satisfy NAPS definition is will be presented in the full version of this paper.

*1) Obtaining atomicity and partial atomicity in* ETHNA*:* ETHNA can be easily converted into a payment system for atomic payments in the following way. Consider some payment $\mu$ for $v$¢. We simply let *any* sub-receipt for a sub-payment count as the receipt for the entire payment $\mu$, and at the same time we instruct the receiver $P_n$ to start acknowledging payments, i.e., signing such receipts only if she receives *all* the sub-payments (for the full amount $v$). This works since (a) as long as $P_n$ did not receive the full amount, there is no receipt that she receive any coins, and (b) once she does it it is in her own best interest to acknowledge *all* sub-payments (and claim all the coins). This can be naturally generalized further to obtain "partial atomicity" where, e.g., the receiver can either receive $0$¢, $v/2$¢, or the full amount of $v$¢. This way of obtaining atomicity may be used in the applications like the one described very recently in [12], where in Sec. 3.1 describe a way to obtain "unlinkability" in atomic payment splitting. The main idea there is to hide the fact that a given payment has been already split. The "atomic ETHNA" satisfies this property, while avoiding using homomorphic hash functions (used in [12]). We leave a full comparison of these two approaches as a direction for future work.

*2) Reducing the size of the fraud proofs:* Recall that a fraud proof is a payment report $\mathcal{Q}$ of a form $\mathcal{Q} = \{\wr(\sigma || \pi_i), \lambda_i \int\}_{i=1}^m$, all the $\pi_i[1]$'s are pairwise distinct, such that the following condition holds:

$$\max_{i:=1,\ldots,m} \lambda_i[|\sigma|] < \sum_{i:=1}^{m} \lambda_i[|\sigma| + 1]. \tag{9}$$

Hence, in the most straightforward implementation it is of length $\Omega(\delta \cdot (\ell + \kappa))$, where $\delta$ is ETHNA's arity, $\ell$ is the maximal length of payment routes, and $\kappa$ is the security parameter

We now show how to reduce this to $O(\delta \cdot \kappa)$. We do it by designing an algorithm that signs the sub-receipts $\wr \phi, \lambda \int$ in a different way. Let $H$ be a collision-resistant hash function, and let $(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$ be a signature scheme. Suppose $(\mathsf{sk}, \mathsf{pk}) \leftarrow_\$ \mathsf{KGen}(1^\kappa)$ is the key pair of $P_n$. To sign $(\phi, \lambda)$ we define a new signature scheme $(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$ (i.e. we later let $\wr \phi, \lambda \int := ((\phi, \lambda), \sigma)$, where $\sigma := \mathsf{Sig}'_{\mathsf{sk}}((\phi, \lambda))$). Let $\mathsf{KGen}' := \mathsf{KGen}$. To define $\mathsf{Sig}((\phi, \lambda))$ first define $\langle h^1, \ldots, h^{|\phi|} \rangle$ recursively as:

$$h^1 := H(\phi[1]),$$

and for $j := 2, \ldots, |\phi|$:

$$h^j := H(\phi[j], h^{j-1}).$$

Then let $\mathsf{Sig}((\phi, \lambda)) := \langle \sigma^1, \ldots, \sigma^{|\phi|} \rangle$, where for each $j$ we have:

$$\sigma^j := \mathsf{Sig}^{\mathsf{sk}}(h^j, \lambda[j])$$

Verification of this signature is straightforward. It is also easy to see that if $(\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$ is existentially unforgeable under chosen message attack, then so is $(\mathsf{KGen}', \mathsf{Sig}', \mathsf{Vf}')$, assuming the signed messages are of a form $(\phi, \lambda)$, where $\phi$ is the payment path[4]. For a message $M$ let $\{M\}_{P_n}$ denote $M$ signed with $(\mathsf{KGen}', \mathsf{Sig}', \mathsf{Vf}')$. It is easy to see that now a fraud proof from Eq. (9) can be compressed to a sequence

$$\left\{ \left( \left\{ h_i^{|\sigma|}, \lambda_i[|\sigma|] \right\}_{P_n}, \pi_i[1], \left\{ h_i^{|\sigma|+1}, \lambda_i[|\sigma|+1] \right\}_{P_n} \right) \right\}_{i=1}^{m}. \tag{10}$$

such that Eq. (9) holds (above "$\pi_i[1]$" is needed to check correctness of $h_i^{|\sigma|+1}$). Since all the signed values are of size linear in the security parameter, and $m \leq \delta$ we get that Eq. (10) is $O(\delta \cdot \kappa)$. Note that this requires the parties

[4]This assumption is needed since payment paths have a clearly marked "ending", namely they have to finish with $(P_n, \mu_n)$, for some $\mu_n$ Otherwise it would be possible to attack this scheme by taking a prefix of a signed message and a prefix of its signature.

(and, pessimistically, the state channel contract) to verify $m$ signatures. This can be reduced to $1$ signature by using signature aggregation techniques, the simplest one being the Merkle trees technique, where we hash all pairs $(h^j, \lambda[j])$ using Merkle hash and sign only the top of the tree. Note that this introduces additional data costs of size $O(\kappa \cdot \log \delta)$.

*Further proof size reduction using "bisection":* Finally, let us remark that the proof Eq. (10) can be further compressed by allowing interaction between the party that discovered cheating (denote it $P$) and $P_n$. This is similar to the bisection technique [17, 33]. Suppose $P$ realized that Eq. 9 does not hold. She can then divide the set of paths in $\mathcal{Q}$ into two halves For convince suppose $m$ is even and let

$$A := \sum_{i:=1}^{m/2} \lambda_i[|\sigma| + 1],$$

and

$$B := \sum_{i:=m/2+1}^{m} \lambda_i[|\sigma| + 1].$$

$P$ can now challenge $P_n$ (on the blockchain) to provide her own calculations of the above sums[5]. Let $A'$ and $B'$ be $P_n$ respective answers. Then one of the following has to hold:

- $\max_{i:=1,\ldots,m} \lambda_i[|\sigma|] < A' + B'$ – then $P$ obtains the fraud proof and we are done.
- $A' < A$ or $B' < B$ – then we can apply this procedure recursively.

It is easy to see that in logarithmic number o rounds $P$ obtains a fraud proof. Note that this fraud proof is short, so it can be easily propagated to other parties (who do not need to repeat the above "game" with $P_n$).

[5] Since elements of $\mathcal{Q}$ can be sorted such a challenge is short.