Get Me out of This Payment! Bailout: An HTLC Re-routing Protocol

Oğuzhan Ersoy[1,2], Pedro Moreno-Sanchez[3], and Stefanie Roos[2]

[1] Radboud University, The Netherlands
oguzhan.ersoy@ru.nl
[2] Delft University of Technology, The Netherlands
s.roos@tudelft.nl
[3] IMDEA Software Institute, Spain
pedro.moreno@imdea.org

Abstract. The Lightning Network provides almost-instant payments to its parties. In addition to direct payments requiring a shared payment channel, parties can pay each other in the form of multi-hop payments via existing channels. Such multi-hop payments rely on a 2-phase commit protocol to achieve balance security; that is, no honest intermediary party loses her coins. Unfortunately, failures or attacks in this 2-phase commit protocol can lead to coins being committed (locked) in a payment for extended periods of time (in the order of days in the worst case). During these periods, parties cannot go offline without losing funds due to their existing commitments, even if they use watchtowers. Furthermore, they cannot use the locked funds for initiating or forwarding new payments, reducing their opportunities to use their coins and earn fees.

We introduce `Bailout`, the first protocol that allows intermediary parties in a multi-hop payment to unlock their coins before the payment completes by re-routing the payment over an alternative path. We achieve this by creating a circular payment route starting from the intermediary party in the opposite direction of the original payment. Once the circular payment is locked, both payments are canceled for the intermediary party, which frees the coins of the corresponding channels. This way, we create an alternative route for the ongoing multi-hop payment without involving the sender or receiver. The parties on the alternative path are incentivized to participate through fees. We evaluate the utility of our protocol using a real-world Lightning Network snapshot. Bailouts may fail due to insufficient balance in alternative paths used for re-routing. We find that attempts of a node to bailout typically succeed with a probability of more than 94% if at least one alternative path exists.

1 Introduction

Payment channels have emerged as one of the most promising mitigations to the blockchain scalability problem [22]. A payment channel enables two users to perform many payments between them while requiring only two transactions to be published on the blockchain. In a bit more detail, Alice and Bob open a channel between each other by submitting a transaction to the blockchain that locks coins in a shared deposit. A (off-chain) payment only requires that Alice and Bob exchange an authenticated agreement of a new deposit's balance, i.e., the split of the funds in the deposit between the two. This off-chain payment operation can be repeated arbitrarily often until the channel is closed by publishing a transaction on the blockchain that releases the deposited coins according to the last authorized balance. However, opening a channel only pays off if parties transact with each other repeatedly.

To enable parties to conduct a transaction without establishing a new channel, payment channel networks (PCNs) [3–5,16,17,33,41] allow routing payments from a sender to a receiver via multiple channels. In such a *multi-hop* payment, each channel in the route is updated with the payment amount (and a fee) from the sender to the receiver. The most important requirement for a multi-hop payment protocol is balance security [5, 19, 33], i.e., no honest party other than the sender should lose coins and the sender should only lose the payment amount and the fees. While there exist several proposals to achieve balance security [5,19,34,41], *hash-time lock contracts* (HTLC) are currently implemented in the Lightning Network (LN).

An HTLC-based multi-hop payment works as follows: When agreeing to conduct a payment, the receiver chooses a random value and then gives the hash of that value to the sender. The sender decides on one payment path. The first node on each channel making up the path commits to paying the second node if the second node provides the preimage of the hash within a certain time. The time, which depends on the node's individual preference and its position in the path, is called the timelock of the conditional payment. More details on the HTLC construction and timelocks are given in Section 2. Once all the commitments are made, the receiver provides the preimage and the preimage is forwarded along the path back to the sender, concluding the promised payments.

While the protocol provides balance security, it causes issues with regard to the availability of coins. After a node has committed to a payment, neither the node nor their successor on the path can use the payment amount for concurrent payments, as it is not yet known whether the coins will be successfully transferred. The typical amount of time funds can be *locked* in this manner is in the order of seconds, assuming that all parties are responsive. However, there can sometimes be delays in the order of days [38].

The delays can be caused by nodes being offline or payment failure. Thus, the locked coins can severely limit a node's liquidity and prevent them both from initiating payments of their own and from forwarding other payments due to the lack of available funds, which can drastically reduce the ability of the network to conduct payments [38, 43]. Also, if there are several locked HTLCs, the parties may not able to accept new HTLCs (even if they have enough funds) because of the upper limit in the number of concurrent HTLC [12]. Moreover, it is important to note that intermediary parties cannot go offline until all the locked payments are released. This holds even with watchtowers, as there is no watchtower protocol that updates the channel state without the presence of the channel owner [7, 8, 15, 25, 28, 36].

These negative effects of unexpectedly long-locked coins give rise to the question: *Is it possible to unlock coins of an intermediary party if the multi-hop payment is not completed and the timelock has not expired?*

Our contributions. In this work, we positively answer this question by providing `Bailout`, which allows an intermediary party, who has locked her coins for an unfinished multi-hop payment, to unlock her coins before the expiration of the corresponding timelock. In a nutshell, `Bailout` allows the intermediary party to re-route the on-going multi-hop payment, so that other nodes with a better availability situation take over the payment, freeing up coins for the intermediary party to use in other payments. We incentivize the other parties to take over the payment through offering them extra fees, typically higher than the standard fee for routing a payment. In this manner, we offload payments from overloaded nodes to nodes with a low load and available funds. Our contributions are:

– We introduce `Bailout`, the first protocol that allows intermediary parties to unlock their coins from an ongoing HTLC payment and provably achieves balance security. `Bailout` re-routes the payment over an alternative path that connects the neighboring parties of the intermediary. It is compatible with HTLC-based multi-hop payments in Lightning: (i) it can be implemented with the scripting language of Bitcoin, (ii) it does not require any additional information than the existing knowledge in Lightning, e.g., the intermediary party knows only her neighbors on the payment path.

– We evaluate our protocol in the face of parties that want to go offline and bailout of their ongoing payments. The level of concurrency and the frequency of long delays determine the amount of locked collateral in the network and hence affect the ability of a party to find an alternative path with sufficient funds. Still, even for high concurrency and frequent delays, less than 6% of bailouts fail.

2 Building Blocks

Transactions and Ledger. In this work, we utilize a simplified version of Bitcoin to model transactions and the ledger as in [3]. The transactions are based on the *unspent transaction output* (UTXO) model, where the coins are represented by outputs. An output $\vec{\theta}$ is defined as a tuple (cash, θ) where cash denotes the number of coins in the output and θ is the corresponding spending condition. For readability, we extract away the details of the ledger functionality. We require that the ledger handles the notion of time in rounds, and the round number corresponds to the number of blocks on the ledger. Also, we assume that a valid transaction is included in a block on the ledger after at most Δ rounds. The details of transactions and ledger functionality are given in Appx. A.

Payment Channels. A payment channel is defined as a tuple of $\gamma := (id, \mathsf{users}, \mathsf{cash}, \mathsf{st})$ where $\gamma.id$ is the id of the channel between parties $P \in \gamma.\mathsf{users}$, $\gamma.\mathsf{cash}$ denotes the capacity of the channel and $\gamma.\mathsf{st} := (\vec{\theta}_1, \dots, \vec{\theta}_n)$ is the state of the channel. We denote channel between A and B as $\gamma_{A,B}$. A channel has three phases: (i) *create* where the channel is opened by publishing the funding transaction on the ledger, (ii) *update* where parties update the state of the channel, and (iii) *close* where parties close the channel by publishing the latest channel state on the ledger. The payment channel functionality is given in Appx. A.

Payment Channel Networks. A payment channel network is a network where parties are nodes and channels are edges. One can route payments from a payer to a payee along multiple channels without requiring a direct channel between them. A Multi-hop payment (MHP) is constructed over a path of channels $\mathsf{path} := (\mathsf{path}[0], \ldots, \mathsf{path}[n-1])$ and conditional payments $(\mathsf{MHP}[0], \ldots, \mathsf{MHP}[n-1])$ (one for each channel) where n is the payment route length. $\mathsf{path}[i]$ is the ith channel in the payment route and $\mathsf{path}[i].\mathsf{payer}$ (and also $\mathsf{MHP}[i].\mathsf{payer}$) denotes the ith party in the path who pays to the $(i+1)$th party, $\mathsf{path}[i].\mathsf{payee}$.

We present the ideal functionality of MHP $\mathcal{F}_{MHP}$ in Appx. E, which has two phases: Setup and Lock, and Pay or Revoke phases. In the Setup and Lock phase, the payment path is created and the channels on the path lock the corresponding amounts. More concretely, at each channel $\mathsf{path}[i]$, $\mathsf{amt}[i]$ coins of $\mathsf{path}[i].\mathsf{payer}$ are locked. Here, the order of the locking corresponds to the order of channels on the path, starting with the channel adjacent to the sender. If the locking fails in a channel on the path, then the locking stops. When all channels in the path are locked, this phase is finished. In the Pay (or Revoke) phase, for each channel of $\mathsf{path}[i]$, the locked coins are paid to $\mathsf{path}[i].\mathsf{payee}$. Unlike in the previous phase, the channel updates are executed in the order from the receiver to the sender. If the payment is not completed before $\mathsf{TL}[i]$, then the locked coins can be revoked and given back to the $\mathsf{path}[i].\mathsf{payer}$.

Lightning Network achieves multi-hop payments via the HTLC (hash time locked contract) protocol. An HTLC is a *conditional payment* where the receiving party can claim the payment amount by providing the preimage of the given hash value. If the preimage is not provided within a certain time, the payment amount returns to the sending party. We write an HTLC tuple with the following attributes $\mathsf{HTLC} := (\mathsf{mid}, \mathsf{cpid}, \gamma,\ \mathsf{payer} \rightarrow \mathsf{payee},\ \mathsf{cond}, TL, amt)$ where $\mathsf{HTLC.cpid}$ is the id of the HTLC in channel $\mathsf{HTLC}.\gamma$ between the payer $\mathsf{HTLC.payer}$ and the payee $\mathsf{HTLC.payee}$. If the HTLC is part of a multi-hop payment, then $\mathsf{HTLC.mid}$ stores the corresponding id, otherwise it is $\perp$. The payment amount of the HTLC is $\mathsf{HTLC}.amt$ that is locked for the condition $\mathsf{HTLC.cond}$. If the HTLC is part of a MHP, the amount is deducted from the available coins of $\mathsf{HTLC.payer}$. If a witness $\mathsf{witness}$ is provided s.t. $\mathcal{H}(\mathsf{witness}) = \mathsf{cond}$ until time $\mathsf{HTLC}.TL$, then the payment amount is given to $\mathsf{HTLC.payee}$. Otherwise, at time $\mathsf{HTLC}.TL$, the amount is returned to $\mathsf{HTLC.payer}$. Note that a channel γ can have several ongoing HTLCs at the same time. For readability, unless it is necessary, we skip the first three attributes of the HTLC tuple, also we omit the payer and payee in figures where they are visually ascertainable. The scripts of an HTLC are given in Appx. A.

As explained previously, a MHP in Lightning is done by locking HTLCs in the payment path from sender to receiver wrt. the condition cond chosen by the

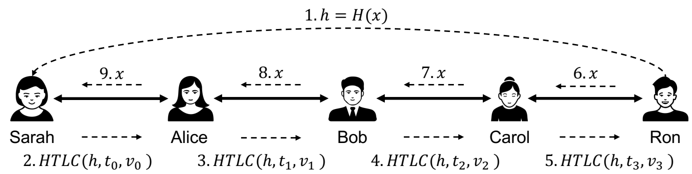


Fig. 1: A multi-hop payment with HTLCs. h denotes $\mathsf{MHP}[i].\mathsf{cond}$ where x is the corresponding preimage, and t_i and v_i represents $\mathsf{MHP}[i].TL$ and $\mathsf{MHP}[i].amt$.

receiver. Note that each intermediary party P_i plays the role of payee in the channel (of $\mathsf{MHP}[i]$) closer to sender, and the role of payer in the subsequent channel (of $\mathsf{MHP}[i+1]$), which is closer to the receiver. Party $\mathsf{MHP}[i+1].\mathsf{payer}$ accepts locking the conditional payment $\mathsf{MHP}[i+1]$ if the following conditions are satisfied: (i) the previous channel should be updated first with the same hash condition, $\mathsf{MHP}[i{+}1].\mathsf{cond} = \mathsf{MHP}[i].\mathsf{cond}$, (ii) the locked amount should be equal to the one in previous channel minus the fee, i.e, $\mathsf{MHP}[i].amt - \mathsf{MHP}[i+1].amt$ is equal to the fee amount chosen by the channel, and the locked amount can be at most the channel balance, and (iii) the timelock of the HTLC is less than or equal to the timelock of the previous channel plus the timelock of the channel chosen by the intermediary, $\mathsf{MHP}[i+1].TL = \mathsf{MHP}[i].TL - T_i$ where T_i is the timelock of the channel. In Lightning Network, the timelock and fee values of a channel is publicly known. An illustrative example of a MHP is given in Figure 1.

After the last channel before the receiver has been updated with an HTLC condition, the receiver reveals the preimage and obtains the payment. Subsequently, all intermediaries forward the preimage to their predecessor. If the receiver does not share the preimage, each channel returns to its initial state after the timelock. In this case, the coins in each channel will be locked and cannot be used until the timelock is over.

3 The `Bailout` protocol

Assume there is an ongoing multi-hop payment (MHP_0) including the channels from A to B and B to C (seen at the Initial State of Figure 2). Let HTLC_A and HTLC_C be the existing HTLCs with condition h and amounts amt_A and amt_C in channels $\gamma_{A,B}$ and $\gamma_{B,C}$:

$$\begin{aligned} \mathsf{HTLC}_A &:= (A \rightarrow B, h, TL_A, amt_A), \\ \mathsf{HTLC}_C &:= (B \rightarrow C, h, TL_C, amt_C). \end{aligned} \tag{1}$$

where $TL_C < TL_A$ and $amt_C < amt_A$. In both channels, coins have been locked for longer than expected by B. If the payment is not completed, B has to wait until the timelock of HTLC_C expires, which can be days.

Motivation. Here, we list some of the potential reasons that B may request to be removed from the long-lasting payment. First, B may want to go offline with minimal monitoring of the blockchain. If there are no ongoing payments locked, B only needs to monitor the blockchain (wrt. the channel timelock, once per day) for potential fraud of the other party of the channel, and this can even be delegated to a watchtower [28]. However, if there are ongoing HTLCs, the channel needs to be updated wrt. the outcome of them, and this cannot be delegated. Note that even if every party in the MHP is honest and online but B is offline, then the MHP cannot be completed until B is online again or the timelocks of B are expired. Thus, other parties also benefit from removing B from the ongoing payment as B's absence may delay the payment further.

Secondly, B may want to close his channels and spend the coins immediately. Even though, B can close the channel with ongoing payments, he needs to wait for them to be finalized. Thirdly, B may want to make an off-chain payment but due to the ongoing payment and the locked coins, there are not enough funds available. In the last scenario, B could also want to unlock his funds to participate in off-chain payments as an intermediary and make profits in the form of fees from other payments using the currently locked coins.

Security and Compatibility Requirements. Here, we aim to design a protocol that unlocks the coins of B, which is compatible with Bitcoin's scripting language and the Lightning Network. The protocol requires the participation of B's neighbors A and C as they need to be involved in unlocking previously made commitments. Without the cooperation of these neighbors, B cannot update the channels. The Lightning Network uses onion routing such that the intermediary only learns the identity of the previous and next node on the path. Thus, our protocol should also not require the identities of other parties on the path, in particular the sender and receiver. Finally, but most importantly, the protocol should provide balance security to every honest intermediary, meaning that no honest party should lose coins regardless of the acts of other parties.

3.1 Overview of `Bailout`

In this work, we design `Bailout` and show that it satisfies all the requirements given above. `Bailout` *re-routes* the ongoing locked HTLCs via an alternative path such that coins of B are released. In a nutshell, the idea is creating new HTLCs in the opposite direction with the same payment amounts and then cancelling them out. For that reason, we create a circular MHP (MHP_1) of length four starting from B that goes through A, D (party in the new route, called a *bailout party*), C and ends at B again (see Step 2 in Figure 2).[4] Once the new MHP is locked, both payments are canceled for B, which frees the coins of the corresponding channels, which is illustrated in the Step 3 of Figure 2. The re-routing of the original payment can be seen in the path difference between the Initial and Final State given in Figure 2.

Naive approach. A naive solution is creating a circular MHP_1 with the same condition as MHP_0, then HTLC_A and $\mathsf{MHP}_1[0]$ have the same amount and the

[4] Here, we require that there is an alternative path between Alice and Carol via only one intermediary, Dave. Later on, we generalize it to multiple intermediaries.

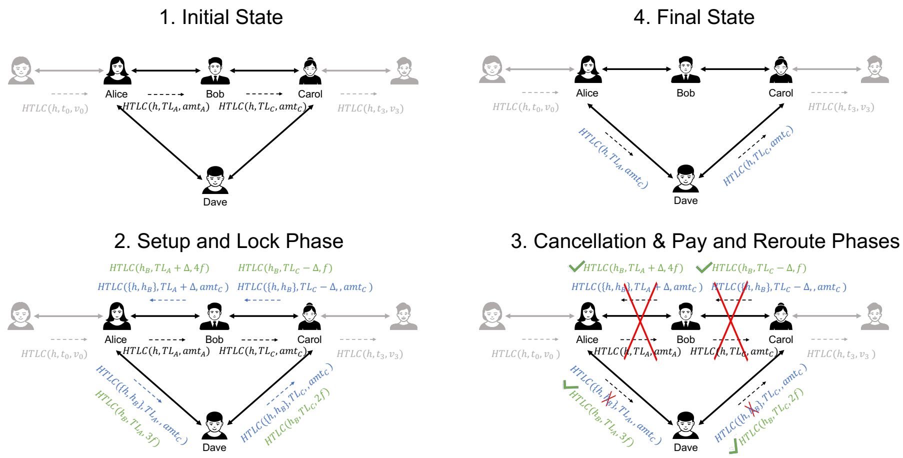


Fig. 2: Simplified protocol phases for the full cancellation/re-routing. In Setup and Lock Phase, the new multi-hop payments (MHP_1 and MHP_2) are locked. In Cancellation Phase, the HTLCs of B are cancelled in the channels with A and C. In Pay and Reroute Phase, MHP_2 is payed by sharing the preimage of h_B and the condition of MHP_1 is reduced to h. For simplification of the figure, we use a constant fee f, which can actually differ among parties. HTLCs of MHP_0, MHP_1 and MHP_2 are colored with black, blue and green respectively.

same hash condition but in opposite directions. Then, for the parties A and B, it would be the same if they cancel both of them, rather than waiting for the payments to be completed. It is similar for the channel between B and C. However, there is a security problem: if the preimage of h is known to A during the locking phase of MHP_1, then B loses his coins. More specifically, just after locking $\mathsf{MHP}_1[0]$, and before locking the other hops in MHP_1, if A knows the preimage[5], A can claim the payment in $\mathsf{MHP}_1[0]$ from B. Yet, if the last hop $\mathsf{MHP}_1[3]$ is not locked, then B is not be compensated in MHP_1.

To overcome the aforementioned problem, the conditional payments in MHP_1 should include an additional condition chosen by B, say h_B. In this way, if MHP_0 is completed during the process, then the new MHP (MHP_1) cannot be spent, and B does not lose his coins. In this case, MHP_1 is cancelled since there is no need to execute the protocol. With the additional condition, after re-routing, we need to ensure that parties A and C do not lose their coins because of the differences in conditions of MHP_0 and MHP_1. From A's perspective, since A is the payer for conditions (h, h_B) in $\mathsf{MHP}_1[1]$, and payee for h in MHP_0 (if she is not the sender), she is guaranteed that after paying in $\mathsf{MHP}_1[1]$, she can get paid in MHP_0. However, for C, it is the opposite. For that reason, we have an interim

[5] A can learn the preimage from B (or the other parties on the path if she is colluding with them).

step for the update between B and C where B needs to reveal the preimage of h_B, which we explain in more detail while presenting the protocol phases.

Incentives. Note that the reason of re-routing HTLCs of B in MHP_0 is that it was not completed in the expected time. The delay can be due to i) a node not forwarding the payment or preimage, ii) a node not peacefully settling the payment that she knows will fail and instead waiting for the timelock to expire, and iii) a receiver (intentionally) not providing the preimage, e.g., in a griefing attack. In case ii) and iii), the payment fails and the cancellation happens at the last possible moment, leading to very long delays. If the payment fails, intermediaries do not receive fees. As a consequence, the bailout party D is unlikely to agree to take over the payment if a fee is only paid when the original payment is successful. For this reason, there should be an additional incentive for D to be involved in the re-routing.

We introduce a secondary MHP, MHP_2 with the sole purpose of paying fees to the bailout party D, as well as A and C, for their involvement in the protocol. The condition of MHP_2 is h_B, which is revealed by B to C after the cancellation of HTLCs in their channel. Thus, the intermediary parties will get paid just after the HTLCs of B are cancelled, which is independent of the completion of MHP_0. D can negotiate its fee with B.

A simplified overview of `Bailout` steps is given in Figure 2. The locking of the new MHPs, MHP_1 and MHP_2, is done in the Setup and Lock phase. After that, the Cancellation phase starts. In this phase, the previous HTLCs, HTLC_A and HTLC_C, together with the new ones in MHP_1 belonging to channels $\gamma_{A,B}$ and $\gamma_{B,C}$ are cancelled, i.e., they are simultaneously revoked. Thus, the coins of B are released. Then, in the last phase, B reveals the secret x_B, so that each party can claim the payment in MHP_2 and also reduce the conditions of HTLCs in MHP_1 to only h.

Extension I - Multiple bailout parties and timelocks. So far we explained the protocol for only one bailout party D that connects A and C. However, such a party may not exist because of the network topology or insufficient balance. Thus, we extend the protocol to multiple bailout parties, D_i's. For the multiple case, the protocol steps do not change. The only concern of having multiple D_i's is that the timelocks of the re-routing payments (MHP_1) have to be divided by the number of new parties. In practice, a default timelock of a channel is either 40 or 144 blocks, with one block being published roughly every 10 minutes [38]. The average transaction confirmation time is not higher than one hour in the last three months (as of Oct. 17, 2022), yet, in the past, it had spikes higher than five days [10]. Thus, we assume the bailout parties can assess a safe timelock value regarding the transaction confirmation time at the moment, and whether they are willing to participate in the protocol with a lower timeout.

Extension II - Partial re-routing (or cancellation). Until now, `Bailout` is defined over the scenario where HTLC_A and HTLC_C of MHP_0 are completely cancelled and MHP_0 is re-routed over the bailout parties. Yet, it is also possible that the payment is partially re-routed and the HTLCs in $\gamma_{A,B}$ and $\gamma_{B,C}$ are updated accordingly. Let amt_{cxl} be the amount that party B aims to re-route via the new path. We can achieve partial re-routing by replacing the amount locked in MHP_1 with amt_{cxl} (instead of the amount in MHP_1). Then, during the cancellation phase, instead of completely cancelling the corresponding HTLCs in $\gamma_{A,B}$ and $\gamma_{B,C}$, we replace HTLC_A and HTLC_C with HTLC_A^{new} and HTLC_C^{new} with the only difference of amount reduction by amt_{cxl}. Hereby, we re-route the amount amt_{cxl} over the channels of bailout parties and keep the remaining in channels $\gamma_{A,B}$ and $\gamma_{B,C}$.

3.2 The Phases of `Bailout`

The formal protocol, Π_{BO}, written in the UC framework, is given in Appendix B. Here, we explain the three phases of `Bailout`: *Setup and Lock*, *Cancellation* and *Pay and Reroute*.

First, we should discuss the path of new multi-hop payments. The protocol requires existence of bailout parties, D_i's, that connect A and C. Here, finding an alternative path is not sufficient, it is also necessary that all channels on the new path have sufficient funds and the new bailout parties charge a fee that is acceptable. Also, as mentioned in the previous section, the more parties are involved, the lower the timelock values are. Thus, having only one bailout party is preferable to not shortening the timelock values. For completeness, we write the protocol for multiple ones.

Setup and Lock phase. In this phase, the new MHPs are created and locked wrt. to the initial HTLCs, HTLC_A and HTLC_C, given in Eqn. (1). B constructs the new MHPs of length n with $\mathsf{mhpInfo}_1 := (\mathsf{amt}_1, \mathsf{TL}, \mathsf{path})$ and $\mathsf{mhpInfo}_2 := (\mathsf{amt}_2, \mathsf{TL}, \mathsf{path})$ such that:

- $\mathsf{path}[0].\mathsf{payer} = \mathsf{path}[n-1].\mathsf{payee} = B$, $\mathsf{path}[0].\mathsf{payee} = \mathsf{path}[1].\mathsf{payer} = A$, $\mathsf{path}[n-2].\mathsf{payee} = \mathsf{path}[n-1].\mathsf{payer} = C$ and $\mathsf{path}[i].\mathsf{payee} = \mathsf{path}[i+1].\mathsf{payer} = D_i$ for $i \in [1, n-3]$.
- For $i \in [0, n-1]$, $\mathsf{amt}_1[i] := amt_{cxl} \leq amt_C$, and $\mathsf{amt}_2[i] = \sum_{j=i}^{n-1} f_j$ where f_j is the fee of channel $\mathsf{path}[j]$.
- $\mathsf{TL}[0] = TL_A + \Delta$, $\mathsf{TL}[n-1] = TL_C - \Delta$, and for $i \in [1, n-2]$, $\mathsf{TL}[i] = \frac{(n-2-i)}{n-3} \times (TL_A - TL_C) + TL_C$.

B chooses a random value x_B and computes $h_B = \mathcal{H}(x_B)$. Then, B computes the HTLCs of MHP_1 and MHP_2 (for $i \in [0, n-1]$):

$$\mathsf{MHP}_1[i] = (\mathsf{payer}_i \to \mathsf{payee}_i, \{h, h_B\}, \mathsf{TL}[i], \mathsf{amt}_1[i]),$$
$$\mathsf{MHP}_2[i] = (\mathsf{payer}_i \to \mathsf{payee}_i, \{h_B\}, \mathsf{TL}[i], \mathsf{amt}_2[i]),$$

where $\mathsf{payer}_i = \mathsf{path}[i].\mathsf{payer}$ and $\mathsf{payee}_i = \mathsf{path}[i].\mathsf{payee}$.

Once the HTLCs are created, starting from $i = 0$ to $n-1$, each channel of $\mathsf{path}[i]$ is locked with both $\mathsf{MHP}_1[i]$ and $\mathsf{MHP}_2[i]$. In the locking phase, parties follow the standard Lightning MHP locking procedure with the only difference being the two parallel HTLCs. If there is failure in any of them, the parties do not continue. Once both MHPs are successfully locked, the phase is completed.

Cancellation phase. In this phase, B updates his channels with both parties $P \in \{A, C\}$ by (partially or fully) canceling the existing HTLCs and unlocking the coins in his channels. B updates his channels $\gamma_{A,B}$ and $\gamma_{B,C}$. To ensure balance security of B, both channels are updated atomically. Also, the new states of both channels should not be publishable on the blockchain until the old ones are revoked. Otherwise, an old state of one channel (e.g., $\gamma_{A,B}$) and a new state of the other channel ($\gamma_{B,C}$) can be published. To achieve this, we use the idea presented in [4] where the updated states have an additional timelock condition. This additional timelock gives enough time for B to make sure that the previous state of both channels are revoked. If not, then he can publish the old states of both channels before the timelocks of the new states.

Another atomicity is required in the channel update of $\gamma_{B,C}$. The update of the channel $\gamma_{B,C}$ and revealing of x_B should be atomic. On the one hand, B should not share x_B with C before updating their channel. Otherwise, a malicious C can stop the update, and if x is revealed between $\mathsf{MHP}_1[n-1].TL$ and $\mathsf{MHP}_1[2].TL$, C can get paid by B from HTLC_C of MHP_0 without paying $\mathsf{MHP}_1[n-1]$. On the other hand, C should not update the channel without learning x_B. Otherwise, if a malicious B does not share x_B, then C might pay for MHP_0 when receiving x (assuming C is not the receiver of MHP_0), but cannot claim the payment from D_{n-3} in $\mathsf{MHP}_1[n-2]$. For that reason, we have an additional condition payment HTLC'_C that updates the channel where B needs to reveal x_B to claim his coins with the timelock of $\mathsf{MHP}_1[n-1].TL$:

$$\mathsf{HTLC}'_C \leftarrow (C \rightarrow B, h_B, TL_C - \Delta, amt_C) \tag{2}$$

where Δ is the time required to publish a transaction on the ledger. It is important to note that, unlike other HTLCs, the amount amt_C in HTLC'_C is not deducted from C, but B, which is the released amount in HTLC_C. It is better to interpret HTLC'_C as a conditional payment that uses collateral of B, and B can re-claim it by revealing x_B, otherwise, it goes to C after the timelock period.

For the channel $\gamma_{B,C}$, there are three existing HTLCs: HTLC_C has condition h for the amount of amt_C from B to C, $\mathsf{MHP}_1[n-1]$ has conditions $\{h, h_B\}$ for the amount of amt_{cxl} from C to B and $\mathsf{MHP}_2[n-1]$ has condition $\{h_B\}$ for the amount of f_{n-1} from C to B. For full cancellation where the amounts are the same, i.e., $amt_C = amt_{cxl}$, B and C update $\gamma_{B,C}$ by canceling HTLC_C and $\mathsf{MHP}_1[n-1]$, and locking HTLC'_C. Otherwise, for partial cancellation where $amt_C > amt_{cxl}$, parties additionally lock HTLC^{new}_C where

$$\mathsf{HTLC}^{new}_C := (B \rightarrow C, h, TL_C, amt_C - amt_{cxl}).$$

For the channel $\gamma_{A,B}$, there are also three ongoing HTLCs: HTLC_A has condition h for the amount of amt_A from A to B, $\mathsf{MHP}_1[0]$ has conditions $\{h, h_B\}$ for the amount of amt_C from B to A and $\mathsf{MHP}_2[0]$ has condition $\{h_B\}$ for the amount of $\sum_{j=0}^{n-1} f_j$ from B to A. For full cancellation, since atomic reveal of x_B is not necessary for A, A and B will update $\gamma_{A,B}$ by canceling HTLC_A and $\mathsf{MHP}_1[0]$. Here, the difference of cancelling HTLC_A and $\mathsf{MHP}_1[0]$, $amt_A - amt_C$, can be seen as an additional fee gain for A. For partial cancellation, parties lock HTLC_A^{new} where

$$\mathsf{HTLC}_A^{new} := (A \to B, h, TL_A, amt_A - amt_{cxl}).$$

In the honest case where both channels of B are updated, B can reveal x_B to C and update their transitory state by unlocking HTLC'_C and receiving payment $\mathsf{MHP}_2[n-1]$. Here, B can also share x_B with A and make the payment of $\mathsf{MHP}_2[0]$.

If a malicious A or C does not complete the channel update, then B publishes the previous state of both channels, which includes the pending HTLCs of MHP_0, MHP_1 and MHP_2. Then, B does not reveal x_B and waits until the end of all timelocks that require x_B. For the initial HTLCs, HTLC_A and HTLC_C, he follows the standard HTLC protocol. Hence, even if A and/or C are malicious, B doesn't lose any funds.

Pay and Reroute. In this phase, the bailout parties get paid by MHP_2 once B reveals x_B. Here, parties follow the standard MHP payment procedure. Also, the intermediaries update the locking condition of MHP_1 by eliminating h_B there. For each $i \in [1, n-2]$, $\mathsf{MHP}_1[i]$ is updated with

$$\mathsf{MHP}_1^{new}[i] = (\mathsf{payer}_i \to \mathsf{payee}_i, h, \mathsf{TL}[i], \mathsf{amt}_1[i]). \quad (3)$$

This implies that MHP_0 is re-routed. In the full cancellation case, HTLC_A and HTLC_C are replaced by $\mathsf{MHP}_1^{new}[1], \ldots, \mathsf{MHP}_1^{new}[n-2]$. In other words, the new payment path goes via $D_1, \ldots, D_{n-3}$, and B is no longer involved in the payment. In partial cancellation case, the locked amounts in channels $\gamma_{A,B}$ and $\gamma_{B,C}$ are reduced by amt_{cxl}, which is now locked in the alternative path.

3.3 Security Discussion

Here, we briefly argue the balance security of the parties. For parties A and C, they are replacing their existing HTLCs of MHP_0 with the ones in MHP_1 where the timelocks are hash conditions are the same. Thus for them, only the path is changing. For the bailout intermediaries, the balance security mainly relies on the security of MHPs since they are regular intermediaries. For B, the balance security comes from the fact that the new MHPs depend on the secret x_B chosen by him. Thus, if the HTLC updates and the cancellation phase are incomplete, then B can always ignore the new HTLCs since only he has the witness x_B of them. Because of the page limitations, we present the detailed security discussion of the HTLC updates with timelines in Appx. C. Also, in Appx. D, we provide the ideal functionality $\mathcal{F}_{BO}$ and we show that our protocol `Bailout` ($\mathit{\Pi}_{BO}$) emulates the ideal functionality $\mathcal{F}_{BO}$.

4 Evaluation

We consider the scenario that a party (Bob) wants to go offline and *bailout of all of his payments*. In Appendix F, we also treat the case of a party wanting to bailout to re-gain liquidity. While in the first scenario, the party wants to get out of all ongoing payments, for the second case he only wants to bailout of a subset of payments that allows him to freely use a certain amount of locked funds.

Metrics. Our evaluation is focused on the rate of successful bailouts. For this, we classify the result of a bailout in three categories:

1. *No Loop*: the network does not contain an alternative path that can be used for bailout for at least one of the payments the party aims to bailout from.
2. *Failed*: the party finds an alternative path for all payments but the bailout fails nevertheless, e.g., due to insufficient balance on the alternative paths.
3. *Successful*: the party managed to bailout of all payments.

During a simulation, we count the number of occurrences of each of the above, and the sum of all these three numbers (called *number of bailout events*).

The first possible cause of failure, 'No Loop', results from the topology of the network. Our algorithm does not directly impact the topology, since no new channel is created or deleted during the protocol execution. However, it stands to reason that if parties have the option to use `Bailout`, they ensure that bailout parties are present by establishing channels such that alternative paths exist. Consequently, we expect a lower amount of 'No Loop' cases when our protocol is deployed than for the current Lightning topology, which we use as a model in our evaluation. In order to focus on protocol-related rather than topology-related aspects, we compute the *failure ratio* as $(Failed)/(Successful + Failed)$.

Simulation Model. We implemented the protocol by extending a known simulator, and the code is open-source[6]. We simulate the Lightning Network by using real-world topology snapshots. As 92% of parties use the LND client [38], our simulation implements the routing behavior of LND. Other clients differ slightly in the path selection but otherwise execute the same behavior.

Payments are executed concurrently. For simplicity, we disregard the time required for local operations and only add network latency for the communication. As Lightning only requires relatively fast operations such as encryption and decryption of messages of 1300 bytes as well as hashing [13], the network latency should dominate the local computation time.

Generally, the latency of payments that are properly executed are chosen such that parties do not bailout during this time but only if additional delays happen. In order for parties to use `Bailout`, we consider the following behaviors that cause additional delays:

– *Delaying*: with a certain probability p, an intermediary or receiver delays the payment (e.g., by being offline) until the maximal timeout.

– *Not settling*: a fraction p of intermediaries does not cancel failed payments but rather waits until the timeout expires.

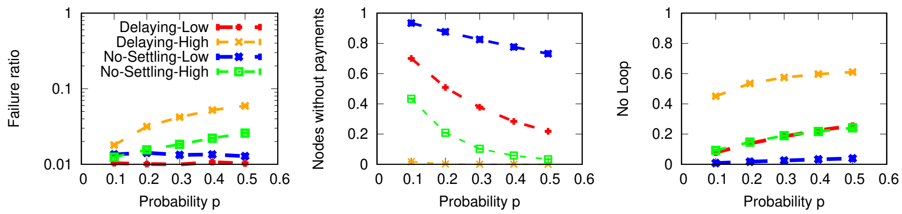
Parameters. We run our simulation on a real-world Lightning snapshot [42]. We restricted our evaluation to the largest connected component with nearly 7,000 nodes and about 65,000 channels to ensure that every node had a path to every other node. For each channel and direction, we choose the balance exponentially

[6] `https://github.com/stef-roos/PaymentRouting/tree/bailout`

with an average of 4 million satoshi, similar to the statistics of Lightning from early 2022 [1]. For the normal Lightning fees, we roughly approximated the statistics as follows: More than 75% of the parties choose a base fee of 0 or 1, so we chose each with a probability of 50%. For the fee rate, the probability to have a rate of 0.000001 was 25%, otherwise the fee rate followed an exponential distribution with parameter $\lambda = 1/0.000004$. We chose the local timelock of each party to be the widely used value of 144 blocks. We generated 100,000 transactions with random source-destination pairs, an exponentially distributed payment value of 10% of the average channel balance, and an average of 10 transactions per party and hour. There is no real-world data on transactions in Lightning as they are considered private. Thus, we took the same parameters as previous work [19]. For the additional delays, i.e., Delaying and Not Settling, p was varied between 0.1 and 0.5 in steps of 0.1. All results were averaged over 10 runs. When the last transaction is initiated, a party B decides that he wants to go offline. He waits 60s such that any ongoing payments without additional delays can terminate. 60s was chosen as Lightning payments should terminate within a minute [2]. During the 60s, he no longer accepts to forward new payments. After the 60s, he attempts to bailout of all remaining payments. For simplicity, we assume that bailout parties are not paid fees here but we consider how to choose fees in Appendix F.

The party aiming to use `Bailout` considers each ongoing payment and first determines a list of alternative paths for the payment. The discovery of alternative paths works as follows: We initialize a queue containing paths, with the first path in the queue being a path containing only the party A, i.e., the party preceding the party B that aims to go offline. We want to find loop-free path from A to B's successor C, which does not contain B. In each step of the path discovery algorithm, we remove the first path from the queue. We iterate over all neighbors I of the last node in the path. If $I = C$, we extend the path by I and add it to the list of alternative paths. Otherwise, if I is not B and appending it to the path does not create a loop, we add the path with I appended to the queue. For efficiency reasons, we limit the alternative path length to at most 4 and the maximal queue size to 1000. If no alternative paths are found, we record the result 'No Loop' to indicate that the bailout failed due to the absence of alternative paths.

After determining a list of alternative paths, the party checks whether he can bailout of the payment using one or several of the alternative paths. Concretely, we consider the first path and determine the amount of funds that can be sent via it in accordance with the balance constraints. If the balance is sufficient to take over the complete payment value, we bailout out of the payment by moving the value to this alternative paths. Note that the balance of the path is accordingly reduced. Otherwise, we split the payment value and execute `Bailout` for the amount that can be moved to the alternative path. For the remaining funds, we consider the second path found, for which we repeat the same process. We continue the algorithm until we have either moved all funds to another path or there are no alternative paths left. In the later case, the bailout fails.



(a) Failure Rate (b) No Need (c) No Path

Fig. 3: a) Failure ratio for bailing out of all ongoing payments; b)+c) Fraction of parties that do not attempt to bailout because they b) do not have ongoing payments or c) do not have an alternative path

The party executes the above process for all ongoing payments he is an intermediary for. Note that the party can only go offline if he can bailout of all these payments. Thus, we mark the bailout as 'Successful' if all separate bailouts are successful. If we experience 'No Loop' for any of them, we terminate and record 'No Loop' as the result of the overall bailout attempt. Otherwise, the bailout is 'Failed'. We count the number of 'No Loop', 'Failed', and 'Successful' by executing the above bailout protocol for every party that has at least one ongoing payment. Based on these value, we compute the success ratio of bailouts. Note that parties cannot bailout of payments that they are the source off. However, as they do not need to relay a preimage to their predecessor when they are the source, these payments do not prevent them from going offline, so that we do not consider them in the set of ongoing payments.

As concurrency has a major impact on the number of ongoing payments, we consider a low-concurrency and a high-concurrency scenario. In the low-concurrency scenario, a party on average sends 0.04 transactions per hour, or roughly 1 transactions per day. In the high concurrency scenario, parties send an average of 10 transactions per hour.

Results. Figure 3a shows the failure ratio. Note that since few payments fail, the figure uses a log scale. High concurrency indicates that at any time, there is more collateral locked and hence the probability that an alternative path has sufficient collateral is lower. Furthermore, *Delaying* can be executed during any payment and by any party whereas *Not Settling* only happens when payments fail, which is less frequent. As a consequence, there are less ongoing payments to bail out for *Not Settling*, resulting in a lower failure ratio.

The main difference between the various parameter selections lies in the number of parties that attempt to bailout. Parties may not attempt a bailout because they do not need to as they have no ongoing payments or because they cannot find an alternative path. Thus, we divide the parties in the snapshot in four classes: 'No Loop', 'Successful', and 'Failed', as defined in Section 4, as well as 'No Need', the parties without ongoing payments. Figures 3b and 3c show the fraction of parties that all fall into the 'No Need' and 'No Loop' category, respectively. As there are more concurrent payments and a higher probability of delay, more parties have ongoing payments and consequently, the fraction of parties not discovering an alternative path increases. In particular, when few parties have ongoing parties, ongoing payments mainly affect central parties with a large number of links. These parties can easily find alternative paths. As more parties are affected, parties with few connections that are not part of any loops have ongoing payments as well. Establishing channels such that alternative paths are possible is hence an important aspect when aiming to use `Bailout`. We can see that as long as alternative paths exist, `Bailout` is nearly always successful.

5 Related Work

There have been several works on the different channel constructions: Lightning channels [41], generalized channels [3,18], and virtual channels [4,6,16,17,23,27]. A network of channels can be used for atomic multi-channel updates and multi-hop payments over parties who do not have a direct channel [5,19,20,34,37,41].

An important aspect regarding multi-hop payments concerns the channel balances. The balance in each side of a channel determines the usability of that channel in a multi-hop payment in that direction. Thus, if a channel is depleted in one direction, then that direction cannot be used for multi-hop payments. There have been studies on reducing depletion by (i) *active re-balancing* with circular payments [9,26,30,40,45], and (ii) *passive re-balancing* with fees and incentive mechanisms [14,21,44]. It is also possible to change the capacity, and thereby the balance, of a channel by Loop-in and Loop-out protocols [29], which require on-chain transactions. Recently, Spider [43] has been proposed to improve channel balances and network throughput. It utilizes a packet-switched architecture that allows splitting transactions into smaller units for better load balancing. These re-balancing protocols re-locate the available (unlocked) coins in the channels, yet they do not solve the unavailability of locked coins.

The existing multi-hop payment protocols require locking coins in each channel in the path for a period of time, which can be days. The coins can be unlocked if the payment is completed (with success or honest immediate cancellation). However, the locking period can be abused by griefing and congestion attacks [31, 38, 39, 46], which lock the available balances in the channels, and limit their usability for the period of time. The attacks can be against the whole network or some specific parties/channels. The effect of the griefing attack can be reduced by changing the path selection algorithm [46], limiting the number of hops [38], or decreasing the locked time [5, 37]. Also, recently, an alternative HTLC protocol with a griefing-penalty mechanism is proposed [35], which requires the receiving parties (payees) to lock coins as well, which are paid in the case of griefing. With this mechanism, the budget of executing the griefing attack is increased by a factor of 4 for a path length of 4. Note that all these (partial) countermeasures are *preventive*, i.e., they aim to reduce the effect of the attack before the payment is locked. To the best of our knowledge, there was no *reactive* countermeasure that frees (unlocks) the locked coins of a party from an ongoing multi-hop payment.

Watchtowers [7, 8, 15, 25, 28, 36] address the issue of offline parties for single payment channels. In a single channel, one party may publish an invalid balance on the blockchain with the goal of earning more coins than their actual balance. Then, the other party has to publish a dispute including the correct balance within a certain time. In a watchtower protocol, the responsibility of raising a dispute is delegated to third party. However, watchtowers are not designed for relaying multi-hop payments as they are observing the blockchain rather than local payments. Indeed, multi-hop payments aim for *value privacy* [32, 33], meaning that no party not involved in the payment should learn the payment value, which seems to contradict the involvement of an outside party.

Acknowledgements. This work has been partially supported by Madrid regional government as part of the program S2018/TCS-4339 (BLOQUES-CM) co-funded by EIE Funds of the European Union; by grant IJC2020-043391-I/MCIN/AEI/10.13039/501100011033; by PRODIGY Project (TED2021-132464 B-I00) funded by MCIN/AEI/10.13039/501100011033/ and the European Union NextGenerationEU/PRTR; and by Ripple's University Blockchain Research Initiative. The Distributed ASCI supercomputer (`https://www.cs.vu.nl/das5/`) was used to run the experiments.

References

1. Lightning network statistics, `https://1ml.com/statistics`
2. Antonopoulos, A.M.: Mastering Bitcoin: Programming the open blockchain. " O'Reilly Media, Inc." (2017)
3. Aumayr, L., Ersoy, O., Erwig, A., Faust, S., Hostáková, K., Maffei, M., Moreno-Sanchez, P., Riahi, S.: Generalized channels from limited blockchain scripts and adaptor signatures. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 13091, pp. 635–664. Springer (2021)
4. Aumayr, L., Maffei, M., Ersoy, O., Erwig, A., Faust, S., Riahi, S., Hostáková, K., Moreno-Sanchez, P.: Bitcoin-compatible virtual channels. In: IEEE Symposium on Security and Privacy. pp. 901–918. IEEE (2021)
5. Aumayr, L., Moreno-Sanchez, P., Kate, A., Maffei, M.: Blitz: Secure multi-hop payments without two-phase commits. In: USENIX Security Symposium. pp. 4043–4060. USENIX Association (2021)
6. Aumayr, L., Moreno-Sanchez, P., Kate, A., Maffei, M.: Donner: Utxo-based virtual channels across multiple hops. IACR Cryptol. ePrint Arch. p. 855 (2021)
7. Avarikioti, G., Laufenberg, F., Sliwinski, J., Wang, Y., Wattenhofer, R.: Towards secure and efficient payment channels. FC (2018)
8. Avarikioti, Z., Litos, O.S.T., Wattenhofer, R.: Cerberus channels: Incentivizing watchtowers for bitcoin. In: FC (2020)
9. Awathare, N., Suraj, Akash, Ribeiro, V.J., Bellur, U.: REBAL: channel balancing for payment channel networks. In: MASCOTS. pp. 1–8. IEEE (2021)
10. Blockchain.com: Average confirmation time (2022), available at: `https://www.blockchain.com/charts/avg-confirmation-time`

11. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067 (2000), https://eprint.iacr.org/2000/067
12. Community, L.N.: Lighning network specification, https://github.com/lightning/bolts/blob/master/02-peer-protocol.md#rationale-7
13. Community, L.N.: Lightning network specification, https://lightning-bolts.readthedocs.io/en/latest/
14. Conoscenti, M., Vetrò, A., Martin, J.C.D.: Hubs, rebalancing and service providers in the lightning network. IEEE Access 7, 132828–132840 (2019)
15. Dryja, T., Milano, S.B.: Unlinkable outsourced channel monitoring. Scaling Bitcoin Milan (2016)
16. Dziembowski, S., Eckey, L., Faust, S., Hesse, J., Hostáková, K.: Multi-party virtual state channels. In: EUROCRYPT (1). Lecture Notes in Computer Science, vol. 11476, pp. 625–656. Springer (2019)
17. Dziembowski, S., Eckey, L., Faust, S., Malinowski, D.: Perun: Virtual payment hubs over cryptocurrencies. In: IEEE Symposium on Security and Privacy. pp. 106–123. IEEE (2019)
18. Dziembowski, S., Faust, S., Hostáková, K.: General state channel networks. In: CCS. pp. 949–966. ACM (2018)
19. Eckey, L., Faust, S., Hostáková, K., Roos, S.: Splitting payments locally while routing interdimensionally. IACR Cryptol. ePrint Arch. 2020, 555 (2020)
20. Egger, C., Moreno-Sanchez, P., Maffei, M.: Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks. In: CCS. pp. 801–815. ACM (2019)
21. van Engelshoven, Y., Roos, S.: The merchant: Avoiding payment channel depletion through incentives. In: DAPPS. pp. 59–68. IEEE (2021)
22. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: Sok: Layer-two blockchain protocols. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 12059, pp. 201–226. Springer (2020)
23. Jourenko, M., Larangeira, M., Tanaka, K.: Lightweight virtual payment channels. In: CANS. Lecture Notes in Computer Science, vol. 12579, pp. 365–384. Springer (2020)
24. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: TCC. Lecture Notes in Computer Science, vol. 7785, pp. 477–498. Springer (2013)
25. Khabbazian, M., Nadahalli, T., Wattenhofer, R.: Outpost: A responsive lightweight watchtower. In: ACM AFT (2019)
26. Khalil, R., Gervais, A.: Revive: Rebalancing off-blockchain payment networks. In: CCS. pp. 439–453. ACM (2017)
27. Kiayias, A., Litos, O.S.T.: Elmo: Recursive virtual payment channels for bitcoin. IACR Cryptol. ePrint Arch. p. 747 (2021)
28. Lab, T.M.D.C.I..M.: Watchtower - watch channels for fraudulent transactions (2018), available at: https://github.com/mit-dci
29. Labs, L.: Loop, available at: https://lightning.engineering/loop/
30. Li, P., Miyazaki, T., Zhou, W.: Secure balance planning of off-blockchain payment channel networks. In: INFOCOM. pp. 1728–1737. IEEE (2020)
31. Lu, Z., Han, R., Yu, J.: General congestion attack on HTLC-based payment channel networks. IACR Cryptol. ePrint Arch. p. 456 (2020)
32. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M.: Silentwhispers: Enforcing security and privacy in credit networks. In: Network and Distributed System Security Symposium (2017)

33. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., Ravi, S.: Concurrency and privacy with payment-channel networks. In: Conference on Computer and Communications Security. pp. 455–471. CCS '17, ACM, New York, NY, USA (2017), http://doi.acm.org/10.1145/3133956.3134096
34. Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Anonymous multi-hop locks for blockchain scalability and interoperability. In: NDSS. The Internet Society (2019)
35. Mazumdar, S., Banerjee, P., Ruj, S.: Griefing-penalty: Countermeasure for griefing attack in lightning network. arXiv preprint arXiv:2005.09327 (2020)
36. McCorry, P., Bakshi, S., Bentov, I., Meiklejohn, S., Miller, A.: Pisa: Arbitration outsourcing for state channels. In: ACM AFT (2019)
37. Miller, A., Bentov, I., Bakshi, S., Kumaresan, R., McCorry, P.: Sprites and state channels: Payment networks that go faster than lightning. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 11598, pp. 508–526. Springer (2019)
38. Mizrahi, A., Zohar, A.: Congestion attacks in payment channel networks. In: Financial Cryptography (2). Lecture Notes in Computer Science, vol. 12675, pp. 170–188. Springer (2021)
39. Pérez-Solà, C., Ranchal-Pedrosa, A., Herrera-Joancomartí, J., Navarro-Arribas, G., García-Alfaro, J.: Lockdown: Balance availability attack against lightning network channels. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 12059, pp. 245–263. Springer (2020)
40. Pickhardt, R., Nowostawski, M.: Imbalance measure and proactive channel rebalancing algorithm for the lightning network. In: IEEE ICBC. pp. 1–5. IEEE (2020)
41. Poon, J., Dryja, T.: The bitcoin lightning network: scalable off-chain instant payments (2016), available at: https://lightning.network/lightning-network-paper.pdf
42. roher: discharged-pc-data (github project), https://git.tu-berlin.de/rohrer/discharged-pc-data/
43. Sivaraman, V., Venkatakrishnan, S.B., Ruan, K., Negi, P., Yang, L., Mittal, R., Fanti, G., Alizadeh, M.: High throughput cryptocurrency routing in payment channel networks. In: NSDI. pp. 777–796. USENIX Association (2020)
44. Stasi, G.D., Avallone, S., Canonico, R., Ventre, G.: Routing payments on the lightning network. In: iThings/GreenCom/CPSCom/SmartData. pp. 1161–1170. IEEE (2018)
45. Subramanian, L.M., Eswaraiah, G., Vishwanathan, R.: Rebalancing in acyclic payment networks. In: PST. pp. 1–5. IEEE (2019)
46. Tochner, S., Zohar, A., Schmid, S.: Route hijacking and dos in off-chain networks. In: AFT. pp. 228–240. ACM (2020)

A Additional Models, Protocols and Functionalities

UTXO Transaction Model. The transactions are based on the *unspent transaction output* (UTXO) model, where the coins are represented by outputs. An output $\vec{\theta}$ is defined as a tuple (cash, θ) where cash denotes the number of coins in the output and θ is the corresponding spending condition. A transaction is a tuple of $\mathsf{tx} := (\mathsf{txid}, \mathsf{Input}, \mathsf{Output}, \mathsf{Witness})$ where txid is the id of the transaction, Input and Output are the inputs and outputs, respectively, and Witness is the witness that satisfies the spending conditions of the transaction outputs. In Bitcoin, the commonly used spending conditions are: signature verification (`Sig`), timelock check (`CheckLockTime`) and hash verification (`CheckCond`). `CheckLockTime` introduces a timing condition such as "a transaction tx is publishable after T blocks once its inputs are published", and `CheckCond` requires a preimage of a given hash value to spend the output. These two conditions are mainly used in payment channel protocols. We denote the hash function with $\mathcal{H}$ and the signature scheme with Σ.

Ledger and Channel Functionalities. Here, we present simplified versions of the ideal functionalities for the ledger and payment channel given in [3].

Ideal Functionality $\mathcal{G}_{Ledger}(\Delta)$
The functionality stores public keys of all parties in `PKI`, and maintains the transactions on the ledger $\mathcal{L}$. Register: Upon receiving (Register, pk_P) from P for the first time P, add (pk_P, P) to `PKI`. Post a transaction: Upon receiving (Post, tx) from P where $P \in$ `PKI`, check if the transaction tx is valid. If the check holds, publish tx on $\mathcal{L}$ within at most Δ rounds.

Ideal Functionality $\mathcal{F}_{chan}$
The functionality handles channel creation, update and closing procedures. Create: Upon receiving (Create, γ, txid$_P$) from P, wait for receiving (Create, γ, txid$_Q$) where P and Q are the parties in channel γ. If the funding transaction of the channel appears on $\mathcal{L}$, then send (Created, γ.id) to P and Q, and register the channel id γ.id. Else stop. Update: Upon receiving (Update, $id, \vec{\theta}$) from a party P where id is already registered for the channel γ and P is part of the channel, if both parties of the channel agree and the new state $\vec{\theta}$ is valid, then update the channel state within at most t_{upd} rounds, and send (Updated, $id, \vec{\theta}$) to the parties of the channel. Else, start forceful closure of the channel. Close: Upon receiving (Close, id), from a party P where id is already registered for the channel γ and P is part of the channel, start the closing procedure. Post the latest state of the channel on the ledger $\mathcal{L}$. Once it appears on $\mathcal{L}$, send (Closed, id) to the parties of the channel. The parties are assumed to spend all the outputs belonging to them accordingly.

MHP and HTLC Protocols. Here, we present the MHP setup and check protocols. Also, we provide the HTLC subprotocols with transaction output scripts. Enc$_{pk}$ denotes the encryption algorithm used in Lightning for onion routing. We denote the realization of the signature algorithm Σ with the script $\texttt{Sig}_{pk}$, that requires the signature wrt. public key pk. Hashing with $\mathcal{H}$ is denoted with the script $\texttt{CheckCond}_{\mathsf{cond}}$, which requires the corresponding preimage wrt. hash value cond. $\texttt{CheckLockTime}_{TL}$ is used for timelock condition where the transaction is publishable after the timelock TL.

Additional Protocols

Setup Multi-Hop Payment

$\texttt{SetupMHP}(\mathsf{mid}, \mathsf{amt}, \mathsf{TL}, \mathsf{cond}, \mathsf{path})$:

Let $n = |\mathsf{path}|$. For each channel in the path $\gamma_i := \mathsf{path}[i]$, assign an id cpid and create the conditional payment:
$\mathsf{HTLC}_i := (\mathsf{mid}, \mathsf{cpid}, \gamma_i, \gamma_i.\mathsf{payer} \to \gamma_i.\mathsf{payee}, \mathsf{cond}, \mathsf{TL}[i], \mathsf{amt}[i])$.
Then, $\mathsf{MHP} := (\mathsf{HTLC}_0, \ldots, \mathsf{HTLC}_{n-1})$.
Let pk_i is the public key of $\mathsf{HTLC}_i.\mathsf{payer}$, create the onion structure: $\mathsf{oMHP}[0] := \mathsf{Enc}_{pk_0}(\mathsf{HTLC}_0, \mathsf{Enc}_{pk_1}(\mathsf{HTLC}_1, \ldots, \mathsf{Enc}_{pk_{n-1}}(\mathsf{HTLC}_{n-1}]) \ldots))$.
Return $\mathsf{oMHP}[0]$.

Check SubProtocols

$\texttt{CheckMHP}(\mathsf{MHP}[i], \mathsf{MHP}[i+1])$:

Check the followings:

- $\mathsf{MHP}[i].\mathsf{mid} = \mathsf{MHP}[i+1].\mathsf{mid}$
- $\mathsf{MHP}[i].\mathsf{payee} = \mathsf{MHP}[i+1].\mathsf{payer}$
- $\mathsf{MHP}[i].\mathsf{cond} = \mathsf{MHP}[i+1].\mathsf{cond}$
- $\mathsf{MHP}[i].TL \geq \mathsf{MHP}[i+1].TL + tl_{\texttt{min}}$
- $\mathsf{MHP}[i].amt \geq \mathsf{MHP}[i+1].amt + \mathsf{f}_{min}$

where $tl_{\texttt{min}}$ is the minimum time difference accepted between two conditional payments, and f_{min} is the minimum accepted fee.
If any of them fails, return Fail. Otherwise, return $\mathsf{Success}$.

$\texttt{CheckCond}(\mathsf{witness}, \mathsf{cond})$:

Check if $\mathsf{witness}$ satisfies the condition cond for a given hard relation R. If $(\mathsf{witness}, \mathsf{cond}) \in R$, return $\mathsf{Success}$, otherwise return Fail. For a hash-based hard relation where $\mathcal{H}$ is the hash function, the check can be done via $\mathcal{H}(\mathsf{witness}) = \mathsf{cond}$.

Conditional Payment SubProtocols

For each of the following subprotocols, we use the following notations:

Parse HTLC as $(\mathsf{mid}, \mathsf{cpid}, \gamma, \mathsf{payer} \to \mathsf{payee}, \mathsf{cond}, TL, amt)$. Let $P := \mathsf{HTLC}.\mathsf{payer}$ be the payer and $Q := \mathsf{HTLC}.\mathsf{payee}$ be the payee of the payment in the channel $\mathsf{HTLC}.\gamma$. Let $\vec{\theta} := (\theta_P, \theta_Q, \vec{\theta}_{ocp})$ be the latest state of the channel $\mathsf{HTLC}.\gamma$ where $\theta_P := (c_P, \texttt{Sig}_{pk_P})$ is the output for the payer's coins, $\theta_Q := (c_Q, \texttt{Sig}_{pk_Q})$ is the output for the payee's coins and $\vec{\theta}_{ocp}$ is the set of the outputs of the ongoing conditional payments.

$\texttt{LockHTLC}(\mathsf{HTLC})$:

1. First, check $c_P \geq \mathsf{HTLC}.amt$, if it fails return Fail. Otherwise, continue.
2. Create the new output object st. $\theta_{new} :=$ $(\mathsf{HTLC}.amt, (\texttt{Sig}_P \wedge \texttt{CheckLockTime}_{\mathsf{HTLC}.TL}) \vee (\texttt{Sig}_Q \wedge \texttt{CheckCond}_{\mathsf{HTLC.cond}}))$.
Compute the new state as $\vec{\theta}' := (\theta'_P, \theta_Q, \vec{\theta}'_{ocp})$ where $\theta'_P = (c_P - \mathsf{HTLC}.amt, \texttt{Sig}_{pk_P})$ and $\vec{\theta}'_{ocp} = \vec{\theta}_{ocp} \cup \{\theta_{new}\}$.

3. Send $(\mathsf{Update}, \mathsf{HTLC}.\gamma.id, \vec{\theta}')$ to $\mathcal{F}_{chan}$.
4. Upon receiving $(\mathsf{Updated}, \mathsf{HTLC}.\gamma.id, \vec{\theta}')$, return Success. Otherwise, return Fail.

$\underline{\texttt{PayHTLC}(\mathsf{HTLC}, \mathsf{witness})}$:

1. Let θ_{cur} be the output in $\vec{\theta}_{cp}$ with the condition HTLC.cond. First, check $\texttt{CheckCond}(\mathsf{witness}, \mathsf{HTLC.cond})$, if it fails return Fail. Otherwise, continue.
2. Compute the new state as $\vec{\theta}' := (\theta_P, \theta'_Q, \vec{\theta}'_{cp})$ where $\theta'_Q = (c_Q + \mathsf{HTLC}.amt, \texttt{Sig}_{pk_Q})$ and $\vec{\theta}'_{cp} = \vec{\theta}_{cp} - \{\theta_{cur}\}$.
3. Send $(\mathsf{Update}, \mathsf{HTLC}.\gamma.id, \vec{\theta}')$ to $\mathcal{F}_{chan}$.
4. Upon receiving $(\mathsf{Updated}, \mathsf{HTLC}.\gamma.id, \vec{\theta}')$, return Success. Otherwise, return Fail.

$\underline{\texttt{RevokeHTLC}(\mathsf{HTLC})}$:

1. Compute the new state as $\vec{\theta}' := (\theta'_P, \theta_Q, \vec{\theta}'_{cp})$ where $\theta'_P = (c_P + \mathsf{HTLC}.amt, \texttt{Sig}_{pk_P})$ and $\vec{\theta}'_{cp} = \vec{\theta}_{cp} - \{\theta_{cur}\}$.
2. Send $(\mathsf{Update}, \mathsf{HTLC}.\gamma.id, \vec{\theta}')$ to $\mathcal{F}_{chan}$.
3. Upon receiving $(\mathsf{Updated}, \mathsf{HTLC}.\gamma.id, \vec{\theta}')$, return Success. Otherwise, return Fail.

$\underline{\texttt{LockHTLC2}(\mathsf{HTLC})}$:

This protocol is only used for HTLC'_C. The difference to the LockHTLC protocol is that the locked amount is taken from the receiver. Note that the pay and revoke functions do not change.

1. First, check $c_Q \geq \mathsf{HTLC}.amt$, if it fails return Fail. Otherwise, continue.
2. Create the new output object st. $\theta_{new} :=$ $(\mathsf{HTLC}.amt, (\texttt{Sig}_P \wedge \texttt{CheckLockTime}_{\mathsf{HTLC}.TL}) \vee (\texttt{Sig}_Q \wedge \texttt{CheckCond}_{\mathsf{HTLC.cond}}))$. Compute the new state as $\vec{\theta}' := (\theta_P, \theta'_Q, \vec{\theta}'_{ocp})$ where $\theta'_Q = (c_Q - \mathsf{HTLC}.amt, \texttt{Sig}_{pk_Q})$ and $\vec{\theta}'_{ocp} = \vec{\theta}_{ocp} \cup \{\theta_{new}\}$.
3. Send $(\mathsf{Update}, \mathsf{HTLC}.\gamma.id, \vec{\theta}')$ to $\mathcal{F}_{chan}$.
4. Upon receiving $(\mathsf{Updated}, \mathsf{HTLC}.\gamma.id, \vec{\theta}')$, return Success. Otherwise, return Fail.

B Our Protocol Π_{BO}

In this section, we present the interactive steps of protocol Π_{BO}, which is explained in Section 3. Also, we provide ideal functionalities for ledger and payment channels, and the additional subprotocols for constructing MHP and HTLCs. The ideal functionality of MHP $\mathcal{F}_{MHP}$ and a simplified version of Lightning Network multi-hop payment Π_{MHP} are given in Appendix E.

Protocol Π_{BO}

Setup and Lock

Party B

Upon receiving $(\mathsf{SETUP}, \mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mhpInfo}_1, \mathsf{mid}_2, \mathsf{mhpInfo}_2)$ from $\mathcal{E}$

1. Parse $\mathsf{mhpInfo}_1 := (\mathsf{amt}_1, \mathsf{TL}_1, \mathsf{path}_1)$ and $\mathsf{mhpInfo}_2 := (\mathsf{amt}_2, \mathsf{TL}_2, \mathsf{path}_2)$.
2. Check the following conditions:
   - Check the paths are the same, i.e., $\mathsf{path}_1 = \mathsf{path}_2 := \mathsf{path}$. Let $A := \mathsf{path}[1].\mathsf{payer}$, $C := \mathsf{path}[n-1].\mathsf{payer}$, and $\mathsf{path}[i].\mathsf{payee} = D_i = \mathsf{path}[i+1].\mathsf{payer}$ for $i \in [1, n-3]$.
   - Check the ongoing HTLCs with id mid_0 with A and C. Let $path_A$ and $path_B$ be part of the path of mid_0 with direction from A to B to C. Check if there are ongoing and locked HTLCs with id mid_0 in these paths. If not, stop. Otherwise, let HTLC_A and HTLC_C be the corresponding HTLCs with $(h, \mathsf{amt}_A, TL_A)$ and $(h, \mathsf{amt}_C, TL_C)$ being the hash condition, the locked amount and the timelock, respectively.
   - Check whether the MHPs are properly generated:
     - Amount: For $i \in [0, n-1]$, check $\mathsf{amt}_1[i] := amt_{cxl} \leq amt_C$ and $\mathsf{amt}_2[i] = \sum_{j=i}^{n-1} f_j$ where f_j is the fee of jth channel and amt_{cxl} is the amount that party B aims to re-route via the new path.
     - Timelocks: Check $\mathsf{TL}_1 = \mathsf{TL}_2 := \mathsf{TL}$, and $\mathsf{TL}[0] = TL_A + \Delta$, $\mathsf{TL}[n-1] = TL_C - \Delta$, and for $i \in [1, n-2]$, $\mathsf{TL}[i] = \frac{(n-2-i)}{n-3} \times (TL_A - TL_C) + TL_C$.

   If any of the checks fails, do not continue.
3. Send $(\mathsf{SETUP\text{–}OK}, \mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2)$ to $\mathcal{E}$.
4. Upon receiving both messages $(\mathsf{INIT\text{–}MHP}, \mathsf{mid}_1, \mathsf{mhpInfo}_1)$ and $(\mathsf{INIT\text{–}MHP}, \mathsf{mid}_2, \mathsf{mhpInfo}_2)$ from $\mathcal{E}$, continue to the next step. If only one is received, or none, then do nothing.
5. Let h be the condition of the HTLCs with id mid_0. Choose a random value x_B, and compute $h_B = \mathcal{H}(x_B)$. Assign $\mathsf{cond}_1 = \{h, h_B\}$ and $\mathsf{cond}_2 = \{h_B\}$
6. Setup the MHPs $\mathsf{oMHP}_1[0] \leftarrow \mathtt{SetupMHP}(\mathsf{mid}_1, \mathsf{amt}_1, \mathsf{TL}_1, \mathsf{cond}_1, \mathsf{path}_1)$ and $\mathsf{oMHP}_2[0] \leftarrow \mathtt{SetupMHP}(\mathsf{mid}_2, \mathsf{amt}_2, \mathsf{TL}_2, \mathsf{cond}_2, \mathsf{path}_2)$. Obtain $(\mathsf{MHP}_1[0], \mathsf{oMHP}_1[1])$ and $(\mathsf{MHP}_2[0], \mathsf{oMHP}_2[1])$ by decrypting $\mathsf{oMHP}_1[0]$ and $\mathsf{oMHP}_2[0]$.
7. Send both $(\mathsf{LockMHP}, \mathsf{MHP}_1[0], \mathsf{oMHP}_1[1])$ and $(\mathsf{LockMHP}, \mathsf{MHP}_2[0], \mathsf{oMHP}_2[1])$ to $\mathsf{path}[0].\mathsf{payee}$, and follow the MHP protocol, Π_{MHP}, steps for locking both MHPs.

Party $\mathsf{path}[i].\mathsf{payee} = \mathsf{path}[i+1].\mathsf{payer}$ (for $i = 0, \ldots, n-2$)

Upon receiving $(\mathsf{LockMHP}, \mathsf{MHP}_1[i], \mathsf{oMHP}_1[i+1])$ and $(\mathsf{LockMHP}, \mathsf{MHP}_2[i], \mathsf{oMHP}_2[i+1])$ from $\mathsf{path}[i].\mathsf{payer}$, follow Π_{MHP} protocol steps for locking both MHPs.

1. If both $\mathsf{MHP}_1[i]$ and $\mathsf{MHP}_2[i]$ are locked, then send $(\mathsf{LockMHP}, \mathsf{MHP}_1[i+1], \mathsf{oMHP}_1[i+2])$ and $(\mathsf{LockMHP}, \mathsf{MHP}_2[i+1], \mathsf{oMHP}_2[i+2])$ to $\mathsf{path}[i+1].\mathsf{payee}$, and follow Π_{MHP} protocol steps for locking both MHPs. Otherwise, stop.

Party B

Upon receiving ($\mathsf{LockMHP}, \mathsf{MHP}_1[n-1], \mathsf{oMHP}_1[n]$) and ($\mathsf{LockMHP}, \mathsf{MHP}_2[n-1], \mathsf{oMHP}_2[n]$) from $\mathsf{path}[n-1].\mathsf{payer}$, follow Π_{MHP} protocol steps for locking both MHPs.

1. If both $\mathsf{MHP}_1[n-1]$ and $\mathsf{MHP}_2[n-1]$ are locked, then the Setup and Lock phase is successful, send ($\mathsf{LOCK–OK}, \mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2$) to $\mathcal{E}$ and continue to the next phase.

Cancellation

Party B

Upon receiving ($\mathsf{CANCEL}, \mathsf{CxlInfo}_1, \mathsf{CxlInfo}_2$) from $\mathcal{E}$ at round τ_0,

1. Parse both $\mathsf{CxlInfo}_1 := (\mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2, \mathsf{path}[0])$ and $\mathsf{CxlInfo}_2 := (\mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2, \mathsf{path}[n-1])$.
2. Let $\mathsf{HTLC}_{A,1} \leftarrow \mathsf{MHP}_1[0]$, $\mathsf{HTLC}_{A,2} \leftarrow \mathsf{MHP}_2[0]$, $\mathsf{HTLC}_{C,1} \leftarrow \mathsf{MHP}_1[n-1]$ and $\mathsf{HTLC}_{C,2} \leftarrow \mathsf{MHP}_2[n-1]$. Create the new HTLCs for the channels $\gamma_{A,B}$ and $\gamma_{B,C}$:
   – Use the same MHP id mid_0, and assign unique ids cpid_A^{new} and cpid_C^{new}.
   – $\mathsf{HTLC}_A^{new} \leftarrow (\mathsf{mid}_0, \mathsf{cpid}_A^{new}, \gamma_{A,B}, A \rightarrow B, h, TL_A, \mathsf{amt}_A - amt_{cxl})$
   – $\mathsf{HTLC}_C^{new} \leftarrow (\mathsf{mid}_0, \mathsf{cpid}_A^{new}, \gamma_{B,C}, B \rightarrow C, h, TL_C, \mathsf{amt}_C - amt_{cxl})$
   Create the temporary HTLC for channel between B and C, $\gamma_{B,C}$:
   – Assign unique ids mid' and cpid'.
   – $\mathsf{HTLC}'_C \leftarrow (\mathsf{mid}', \mathsf{cpid}', \gamma_{B,C}, C \rightarrow B, h_B, TL_C - \Delta, \mathsf{amt}_C)$
3. Let $\mathcal{HTLC}_C \leftarrow \{\mathsf{HTLC}_C, \mathsf{HTLC}_{C,1}, \mathsf{HTLC}_{C,2}, \mathsf{HTLC}_C^{new}, \mathsf{HTLC}'_C\}$ and $\mathcal{HTLC}_A \leftarrow \{\mathsf{HTLC}_A, \mathsf{HTLC}_{A,1}, \mathsf{HTLC}_A^{new}, \mathsf{HTLC}_{A,2}\}$. At time τ_1, send ($\mathsf{Cancel}, \gamma_{B,C}, \mathcal{HTLC}_C$) to C, and ($\mathsf{Cancel}, \gamma_{A,B}, \mathcal{HTLC}_A$) to A.

Party C

Upon receiving ($\mathsf{CANCEL}, \mathsf{CxlInfo}_2$) from $\mathcal{E}$,

1. Wait until receiving ($\mathsf{Cancel}, \gamma_{B,C}, \mathcal{HTLC}_C$) from B. If no such a message is received, then do not continue.
2. Parse $\mathsf{mhpInfo}_2 := (\mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2, \mathsf{path}[n-1])$ and $\mathcal{HTLC}_C := \{\mathsf{HTLC}_C, \mathsf{HTLC}_{C,1}, \mathsf{HTLC}_{C,2}, \mathsf{HTLC}_C^{new}, \mathsf{HTLC}'_C\}$, check correctness of the HTLCs:
   – Timelock: $\mathsf{HTLC}_C^{new}.TL \stackrel{?}{=} \mathsf{HTLC}_C.TL \stackrel{?}{=} \mathsf{HTLC}_{C,1}.TL + \Delta \stackrel{?}{=} \mathsf{HTLC}_{C,2}.TL + \Delta \stackrel{?}{=} \mathsf{HTLC}'_C.TL + \Delta$.
   – Hash: $\mathsf{HTLC}_C.\mathsf{cond} = \mathsf{HTLC}_C^{new}.\mathsf{cond} = \{h\}$, $\mathsf{HTLC}_{C,1}.\mathsf{cond} = \{h, h_B\}$, $\mathsf{HTLC}_{C,2}.\mathsf{cond} = \{h_B\}$, and $\mathsf{HTLC}'_C.\mathsf{cond} = \{h_B\}$.
   – Amount: $\mathsf{HTLC}_{C,1}.amt + \mathsf{HTLC}_C^{new}.amt \stackrel{?}{=} \mathsf{HTLC}_C.amt \stackrel{?}{=} \mathsf{HTLC}'_C.amt$ and $\mathsf{HTLC}_{C,2}.amt \geq f_C$ where f_C is the channel fee.
   – If any of them fails, do not continue and initiate channel closing of $\gamma_{B,C}$.
3. If all the checks are successful, send ($\mathsf{UpdateOk}, \gamma_{B,C}$) to B, and continue.

Party A

Upon receiving $(\mathsf{CANCEL}, \mathsf{CxlInfo}_1)$ from $\mathcal{E}$,

1. Wait until receiving $(\mathsf{Cancel}, \gamma_{A,B}, \mathcal{HTLC}_A)$ from B. If not such a message is received, then do not continue.
2. Parse $\mathsf{mhpInfo}_1 := (\mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2, \mathsf{path}[0])$ and $\mathcal{HTLC}_A := \{\mathsf{HTLC}_A, \mathsf{HTLC}_{A,1}, \mathsf{HTLC}_A^{new}, \mathsf{HTLC}_{A,2}\}$, check correctness of the HTLCs:
   - Timelock: $\mathsf{HTLC}_A.TL \stackrel{?}{=} \mathsf{HTLC}_A^{new}.TL \stackrel{?}{=} \mathsf{HTLC}_{A,1}.TL - \Delta \stackrel{?}{=} \mathsf{HTLC}_{A,2}.TL - \Delta$.
   - Hash: $\mathsf{HTLC}_A.\mathsf{cond} = \mathsf{HTLC}_A^{new}.\mathsf{cond} = \{h\}$, $\mathsf{HTLC}_{A,1}.\mathsf{cond} = \{h, h_B\}$ and $\mathsf{HTLC}_{A,2}.\mathsf{cond} = \{h_B\}$.
   - Amount: $\mathsf{HTLC}_{A,1}.amt + \mathsf{HTLC}_A^{new}.amt \stackrel{?}{=} \mathsf{HTLC}_A.amt$ and $\mathsf{HTLC}_{A,2}.amt \geq f_A$ where f_A is the channel fee.
   - If any of them fails, do not continue and initiate channel closing of $\gamma_{A,B}$.
3. If all the checks are successful, send $(\mathsf{UpdateOk}, \gamma_{A,B})$ to B, and continue.

Party B

Wait to receive messages $(\mathsf{UpdateOk}, \gamma_{B,C})$ from C and $(\mathsf{UpdateOk}, \gamma_{A,B})$ from A,

1. Initiate channel updates for both $\gamma_{B,C}$ and $\gamma_{A,B}$ at round τ_1:
   - Execute $\mathtt{RevokeHTLC}(\mathsf{HTLC}_C), \mathtt{LockHTLC}(\mathsf{HTLC}_C^{new}), \mathtt{RevokeHTLC}(\mathsf{HTLC}_{C,1})$ and $\mathtt{LockHTLC2}(\mathsf{HTLC}'_C)$ simultaneously in the same channel update of $\gamma_{B,C}$.
   - Execute $\mathtt{RevokeHTLC}(\mathsf{HTLC}_A), \mathtt{LockHTLC}(\mathsf{HTLC}_A^{new}), \mathtt{RevokeHTLC}(\mathsf{HTLC}_{A,1})$ simultaneously in the same channel update of $\gamma_{A,B}$.

   Here before revoking the previous states of the channels, wait for both parties A and C to revoke. If any of them is not revoked within $\tau_1 + t_{UPD}$, then execute channel closing of $\gamma_{A,B}$ and $\gamma_{B,C}$ with both parties. Otherwise continue.

Party C

If initiated by B, follow the channel update protocol for execution of $\mathtt{RevokeHTLC}(\mathsf{HTLC}_C)$, $\mathtt{LockHTLC}(\mathsf{HTLC}_C^{new})$, $\mathtt{RevokeHTLC}(\mathsf{HTLC}_{C,1})$, and $\mathtt{LockHTLC2}(\mathsf{HTLC}'_C)$.

Party A

If initiated by B, follow the channel update protocol for execution of $\mathtt{RevokeHTLC}(\mathsf{HTLC}_A)$, $\mathtt{LockHTLC}(\mathsf{HTLC}_A^{new})$, and $\mathtt{RevokeHTLC}(\mathsf{HTLC}_{A,1})$.

Party B

If both channels $\gamma_{A,B}$ and $\gamma_{B,C}$ are updated with the corresponding revoking and locking payments,

1. Initiate channel updates for $\gamma_{A,B}$ and $\gamma_{B,C}$ at round τ_2:
   - Execute $\mathtt{PayHTLC}(\mathsf{HTLC}'_C, x_B)$ and $\mathtt{PayHTLC}(\mathsf{MHP}_2[n-1], x_B)$ in channel $\gamma_{B,C}$.
   - Execute $\mathtt{PayHTLC}(\mathsf{MHP}_2[0], x_B)$ in channel $\gamma_{A,B}$.

Party C

Once B initiates $\texttt{PayHTLC}(\mathsf{HTLC}'_C, x_B)$ and $\texttt{PayHTLC}(\mathsf{MHP}_2[n-1], x_B)$,

1. Check $\mathcal{H}(x_B) = \mathsf{HTLC}'_C.\mathsf{cond}$, if it fails, then do not continue. Otherwise, follow the channel update protocol.

Party A

Once B initiates $\texttt{PayHTLC}(\mathsf{MHP}_2[0], x_B)$,

1. Check $\mathcal{H}(x_B) = \mathsf{MHP}_2[0].\mathsf{cond}$, if it fails, then do not continue. Otherwise, follow the channel update protocol.

Party B

1. If the updates fail, then execute channel closing of $\gamma_{A,B}$ and $\gamma_{B,C}$ with A and C.
2. If the updates are completed, then the Nullify phase is successful and send $(\mathsf{CANCEL\text{–}OK}, \mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2)$ to $\mathcal{E}$ and continue to the next phase.

Pay and Reroute

Party $\mathsf{path}[i].\mathsf{payee}$ (for i $= n-2$ to 1)

Upon receiving $(\mathsf{REDUCE\text{–}CP}, \mathsf{mid}_1, \mathsf{mid}_2, \mathsf{path}[i])$ from $\mathcal{E}$,

1. Distinguish the following cases:
   - For $i = n-2$, i.e., $\mathsf{path}[i].\mathsf{payee} = C$, check if the nullify phase is completed by canceling HTLC with id mid_1. Otherwise, do not continue.
   - For $i < n-2$, check if HTLC in $\mathsf{path}[i+1]$ is paid with id mid_2. Otherwise, do not continue.
2. Let x_B the corresponding value revealed in the previous step. Execute $\texttt{PayHTLC}(\mathsf{MHP}_2[i], x_B)$ by revealing x_B to C.
3. Send $(\mathsf{ReduceCP}, \mathsf{MHP}_1[i])$ to $\mathsf{MHP}_1[i].\mathsf{payer}$.

Party $\mathsf{path}[i].\mathsf{payer}$ (for i $= n-2$ to 1)

Once $\mathsf{path}[i].\mathsf{payee}$ initiates $\texttt{PayHTLC}(\mathsf{MHP}_2[i], x_B)$,

1. Check $\mathcal{H}(x_B) = \mathsf{MHP}_2[i].\mathsf{cond}$, if it fails, then do not continue. Otherwise, follow the channel update protocol.

Upon receiving $(\mathsf{ReduceCP}, \mathsf{MHP}_1[i])$ from $\mathsf{MHP}_1[i].\mathsf{payee}$,

1. If $\mathsf{MHP}_2[i]$ is paid, then initiate the channel update with $\mathsf{MHP}_1[i].\mathsf{payee}$:

– Execute $\texttt{RevokeHTLC}(\mathsf{MHP}_1[i])$ and $\texttt{LockHTLC}(\mathsf{MHP}'_1[i])$ simultaneously in the same channel update where the only difference between the HTLCs is the condition and $\mathsf{MHP}'_1[i].\mathsf{cond} := h$.

Party $\mathsf{path}[i].\mathsf{payee}$ (for i = $n-2$ to 1)

Once $\mathsf{path}[i].\mathsf{payer}$ initiates $\texttt{RevokeHTLC}(\mathsf{MHP}_1[i])$ and $\texttt{LockHTLC}(\mathsf{MHP}'_1[i])$,

1. Follow the channel update protocol. If the update is completed, then the Pay and Reduce Condition phase is successful, send $(\mathsf{REDUCE–OK}, \mathsf{path}[i])$ to $\mathcal{E}$.

C Security Discussion of Our Protocol with a Timeline

In this section, we argue the balance security of the parties with the timeline of the ongoing HTLCs. Here, we will investigate the full cancellation case where B would like to cancel both of the existing HTLCs, HTLC_A and HTLC_C. The analysis of partial cancellation case (where the payment amounts are partially moved to another path) can be shown similarly.

The ideal functionality for multi-hop payments $\mathcal{F}_{MHP}$ ensures the balance security of an intermediary party under the assumption that the timelock difference between two consecutive conditional payments (CP) is adequate for an honest party to react. Also, each intermediary party locks his coins once he is ensured to be paid for the same locking condition by the previous party. Thus, in our security analysis, we can rely on the security guarantees of the MHP functionality. In this manner, since the bailout parties are only involved as intermediaries for the new MHPs, MHP_0 and MHP_1, we omit their analysis.

We discuss the balance security of B and the neighbors A and C. For each case, we show that an honest party does not lose their coins regardless of the actions of others. First, we analyze the case of B. B is aiming to cancel the existing HTLCs in his channels with A and C. For that, he first constructs new HTLCs from A to C via $D_1, \ldots, D_{n-3}$. Then, he cancels all HTLCs with his neighbors.

Balance Security of B: As shown in Figure 4[7], at the beginning of the protocol there are two HTLCs, HTLC_A and HTLC_C where he is guaranteed that if he pays HTLC_C to C in exchange for the corresponding preimage x, he can claim the same amount (plus fee) via HTLC_A from A by sharing the same preimage.

In the Setup and Lock phase, B creates two MHPs, MHP_1 and MHP_2, which are both conditioned with a hash value h_B of his choice. Thus, he is the only party who can start the unlocking of these new MHPs, which ensures that no one can claim the new HTLC payments of MHP_1 and MHP_2. This is crucial if the preimage of the ongoing HTLCs HTLC_A and HTLC_C is revealed during this phase. In that case, B can finalize both HTLC_A and HTLC_C payments and

[7] For simplicity, we did not include MHP_2 in the figures.

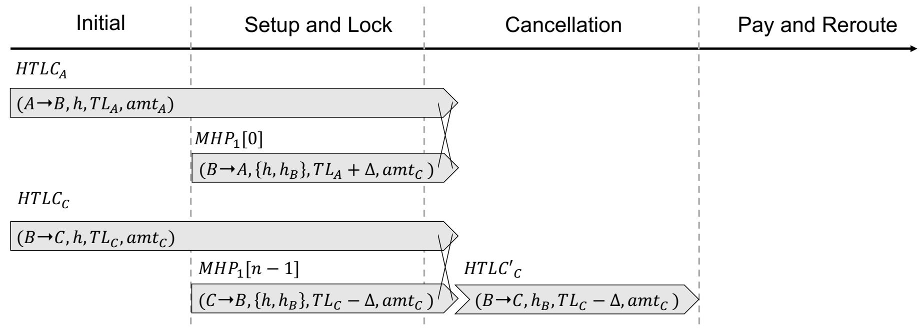


Fig. 4: Conditional Payments of B wrt. protocol phases

cancel the new MHPs. If the preimage is not received by B and both MHPs are successfully locked, then Setup and Lock phase is completed and B can rely on the standard HTLC guarantees. If at least one of the new MHPs is not successfully locked, then B assumes the Setup and Lock phase failed, and does not continue. For releasing of the locked coins earlier, B may start revocation of them as well.

In the Cancellation phase, B first updates his channels with both parties $P \in \{A, C\}$ by canceling the existing HTLCs. Both channels are updated simultaneously.

For the channel $\gamma_{A,B}$, there are three existing HTLCs: HTLC_A is conditioned with h of amount amt_A from A to B, $\mathsf{MHP}_1[0]$ is conditioned with $\{h, h_B\}$ of amount amt_C from B to A, $\mathsf{MHP}_2[0]$ is conditioned with $\{h_B\}$ of fee amount $\mathsf{MHP}_2[0].\mathsf{amt}$ from B to A. Since x_B is known to B, it can be seen that HTLC_A and $\mathsf{MHP}_1[0]$ HTLCs are conditioned with h. Thus, parties can update their channel balances accordingly by canceling both HTLCs. For the channel $\gamma_{B,C}$, B and C can update their channels by cancelling the HTLCs HTLC_C, $\mathsf{MHP}_1[n-1]$ and $\mathsf{MHP}_2[n-1]$, and locking HTLC'_C (See Figure 4). The interim step HTLC'_C has the updated channel balances where C can receive her coins immediately, but B needs to provide x_B to claim his coins. Note that since B knows x_B, he can claim his coins whenever he wants.

If one of the channel updates of $\gamma_{A,B}$ or $\gamma_{B,C}$ fails, meaning that it is not revoked, then B publishes the non-updated version of both channels on the blockchain. Here, since the updates of both channels have additional timelock to publish, the other parties are not able to publish the new state of the channels. This ensures the atomicity of both channel updates. In this case, party B does not reveal x_B and waits for timelocks of the HTLCs of MHP_1 and MHP_2 to reclaim his coins. For HTLC_A and HTLC_C, he will either reclaim HTLC_C after timelock or receive the coins from HTLC_A if x is revealed. Overall, it can be said that B is not losing his coins if the updates fail.

If both channel updates are successful, then B can share x_B with C to update their channel to unlock the coins in HTLC'_C in $\gamma_{B,C}$. If C does not collaborate, then B can publish the new channel state on the blockchain and claim his coins by revealing x_B. For the channel $\gamma_{A,B}$, coins of both parties are already unlocked and can be used for other payments. Thus, if the channel updates are successful, then B is again not losing his coins. This concludes the balance security for B.

Case of A: As shown in Figure 5, for party A, initially, there is an HTLC HTLC_A where A pays to B for the preimage x. In the Setup and Lock phases, two new MHPs are created, and Lightning's MHP construction ensures the balance security of them. In the Cancellation phase, A updates her channel with B by canceling HTLC_A and $\mathsf{MHP}_1[0]$. Here, both of the HTLCs are conditioned with h, and $\mathsf{MHP}_1[0]$ has an additional condition of h_B. Since, A is the payer of $\mathsf{MHP}_1[0]$, it is convenient for A to cancel both HTLCs. This is because if the preimage of h is revealed, then A has to pay HTLC_A to B, yet cannot claim $\mathsf{MHP}_1[0]$ without knowing x_B.

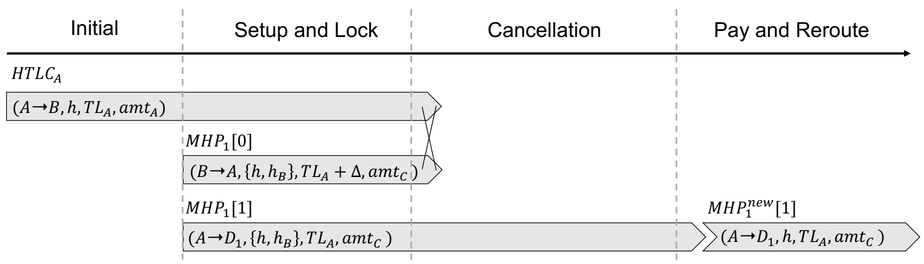


Fig. 5: Conditional Payments of A wrt. protocol phases

If the cancellation of HTLCs fails, i.e., the channel update of $\gamma_{A,B}$ is failed, then A waits for all MHPs to be finalized by either fulfilling the condition or timelocks. In both cases, the balance security of A is guaranteed by the MHP constructions.

If the cancellation succeeds, then A has the HTLCs of MHP_2 and $\mathsf{MHP}_1[1]$. For MHP_2, her balance security is guaranteed by standard MHP construction. MHP_2 will either succeed if x_B is revealed or not; in both cases, A will not lose her coins.

For $\mathsf{MHP}_1[1]$, we need to show that it is equivalent to the initial state HTLC_A regarding the balance security of A. For that, it can be seen that the differences between the two HTLCs: preimage conditions and timelocks. If there is one bailout intermediary, then the timelocks of both HTLCs are the same, meaning that A can reclaim her coins at the same time. However, if there are multiple intermediaries, then the timelock of $\mathsf{MHP}_1[1]$ will be lower than that of HTLC_A. Therefore, A can claim her coins earlier. For the hash conditions, the preimage conditions in $\mathsf{MHP}_1[1]$ covers the one in HTLC_A, thus if A pays the payment $\mathsf{MHP}_1[1]$, she obtains x. Overall, her balance security is not affected.

Case of C: The case of C is similar to A. If the new HTLCs are locked but the channel update fails, then balance security is guaranteed by Lightning's HTLC construction. The main difference between C and A happens if the update is successful. In that case, C is replacing HTLC_C with $\mathsf{MHP}_1[n-2]$, and $\mathsf{MHP}_1[n-2]$ has an additional preimage condition h_B. Unlike A, since C is the payee of these HTLCs, it is important to make sure that C obtains x_B before the corresponding timelock is expired. For this reason, as shown in Figure 6, the update of $\gamma_{B,C}$ has an interim state HTLC'_C where B is required to publish the preimage x_B to claim his coins. This condition ensures that C either obtains x_B within time $\mathsf{MHP}_1[n-1].TL$ or C can get the coins of B in their channel. Since the amount of locked coins of B in HTLC'_C is amt_C, C is adequately compensated if x_B is not revealed on time.

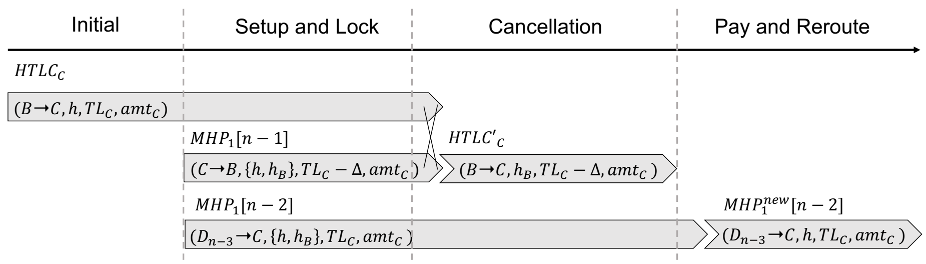


Fig. 6: Conditional Payments of C wrt. protocol phases

In the case where x_B is revealed, HTLC_C and $\mathsf{MHP}_1[n-2]$ are equivalent regarding the hash condition. The only difference of the HTLCs is the timelocks, in $\mathsf{MHP}_1[n-2]$, the timelock condition is $1/(n-3)$ of the timelock of HTLC_C. Thus, for only one bailout party the timelock is the same, but otherwise, it is smaller. Under the assumption that the new timelock is adequate for claiming the coins in the case of a dispute, both HTLCs are equivalent. Overall, the balance security of C is preserved in the protocol.

D Security Analysis

We model the security of `Bailout` in the Universal Composability framework [11]. We follow a similar security model as other off-chain protocols [3–5, 16, 17]. We assume the set of parties involved in the protocol is fixed and the public keys of all parties are known in `PKI`. A PPT (probabilistic polynomial time) adversary $\mathcal{A}$ can corrupt any party at the beginning of the protocol, which is called a *static adversary*. Once a party is corrupted, $\mathcal{A}$ can read the internal state, as well as all of the incoming and outgoing messages, of that party. The communication channels between parties and functionalities are secure and authenticated. Also, we assume a synchronous communication network, and all parties know the current round by utilizing an ideal functionality $\mathcal{F}_{clock}$ [24].

We use $\mathcal{G}_{Ledger}(\Delta)$ to model global ledger functionality where Δ denotes the upper bound on the delay of publishing transactions on the ledger. Let $\mathcal{H}$ be the hash function and Σ be the signature scheme used in the ledger. Moreover, we use $\mathcal{F}_{chan}$ to model a simplified ideal functionality for payment channels. The ideal functionalities are given in Appendix A. To model two-phase multi-hop payments, we present a simplified ideal functionality $\mathcal{F}_{MHP}$ and its realizations for Lightning Network Π_{MHP} in Appendix E.

We present an hybrid ideal functionality $\mathcal{F}_{BO}$ that achieves the behavior of the bailout operation. $\mathcal{F}_{BO}$ also stipulates any behavior of the ledger, payment channel and multi-hop payment functionalities as well. We show that $\mathcal{F}_{BO}$ satisfies balance security, i.e., honest parties do not lose their coins. Then, we show that our protocol `Bailout` (Π_{BO}), explained in Section 3 and defined in UC framework in Appendix B, emulates the ideal functionality $\mathcal{F}_{BO}$.

Ideal Functionality $\mathcal{F}_{BO}$

The hybrid ideal functionality $\mathcal{F}_{BO}$ maintains the set of (to be) nullified conditional payments $\mathcal{HTLC}_{BO}$ and acts as the multi-hop payment functionality $\mathcal{F}_{MHP}$ when necessary. For the MHPs, it maintains the set of ongoing multi-hop payments $\mathcal{MHP}$ and conditional payments $\mathcal{HTLC}$.

Setup and Lock

Upon receiving $(\mathsf{SETUP}, \mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mhpInfo}_1, \mathsf{mid}_2, \mathsf{mhpInfo}_2)$ from B where $B := \mathsf{path}_1[0].\mathsf{payer}$ in $\mathsf{mhpInfo}_1 := (\mathsf{amt}_1, \mathsf{TL}_1, \mathsf{path}_1)$ and $\mathsf{mhpInfo}_2 := (\mathsf{amt}_2, \mathsf{TL}_2, \mathsf{path}_2)$,

1. Check the following conditions:
   - Check the paths are the same, i.e., $\mathsf{path}_1 = \mathsf{path}_2 := \mathsf{path}$. Let $A := \mathsf{path}[1].\mathsf{payer}$, $C := \mathsf{path}[n-1].\mathsf{payer}$, and $\mathsf{path}[i].\mathsf{payee} = D_i = \mathsf{path}[i+1].\mathsf{payer}$ for $i \in [1, n-3]$.
   - Let $path_A$ in channel $\gamma_{A,B}$ and $path_C$ in channel $\gamma_{B,C}$ be part of the path of mid_0 with direction from A to B to C. Check if there are two CPs $(\mathsf{mid}_0, \mathsf{amt}_A, TL_A, path_A, \texttt{status} = \textit{locked})$ and $(\mathsf{mid}_0, \mathsf{amt}_C, TL_C, path_C, \texttt{status} = \textit{locked})$ stored in $\mathcal{HTLC}$.
   - Check whether the MHPs are properly generated:
     - Amount: For $i \in [0, n-1]$, check $\mathsf{amt}_1[i] := amt_{cxl} \leq amt_C$ and $\mathsf{amt}_2[i] = \sum_{j=i}^{n-1} f_j$ where f_j is the fee of jth channel.
     - Timelocks: Check $\mathsf{TL}_1 = \mathsf{TL}_2 := \mathsf{TL}$, and $\mathsf{TL}[0] = TL_A + \Delta$, $\mathsf{TL}[n-1] = TL_C - \Delta$, and for $i \in [1, n-2]$, $\mathsf{TL}[i] = \frac{(n-2-i)}{n-3} \times (TL_A - TL_C) + TL_C$.

   If any of the checks fails, do not continue.
2. Send $(\mathsf{SETUP\text{–}OK}, \mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2)$ to B.
3. Upon receiving both $(\mathsf{INIT\text{–}MHP}, \mathsf{mid}_1, \mathsf{mhpInfo}_1)$ and $(\mathsf{INIT\text{–}MHP}, \mathsf{mid}_2, \mathsf{mhpInfo}_2)$ from B, execute the multi-hop payment functionality and follow the functionality steps. If only one of the messages is received, or none, then do nothing.
4. If all the channels in the path are locked, then the Setup and Lock phase is successful, store $(\mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2, \mathsf{path})$ in $\mathcal{HTLC}_{BO}$, and send $(\mathsf{LOCK\text{–}OK}, \mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2)$ to B. Otherwise stop.

Cancellation

Upon receiving ($\mathsf{CANCEL}, \mathsf{CxlInfo}_1, \mathsf{CxlInfo}_2$) from $B := \mathsf{path}[0].\mathsf{payer}$ at round τ_0,

1. Parse $\mathsf{CxlInfo}_1 := (\mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2, \mathsf{path}[0])$ and $\mathsf{CxlInfo}_2 := (\mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2, \mathsf{path}[n-1])$. Check $(\mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2, \mathsf{path})$ is stored in $\mathcal{HTLC}_{BO}$ with the corresponding $\mathsf{path}[0]$ and $\mathsf{path}[n-1]$; if not, go idle.
2. Wait until round $\tau_0 + 1$ for receiving both messages ($\mathsf{CANCEL}, \mathsf{CxlInfo}_1$) from $A := \mathsf{path}[0].\mathsf{payee}$, and also ($\mathsf{CANCEL}, \mathsf{CxlInfo}_2$) from $C := \mathsf{path}[n-1].\mathsf{payer}$, then continue. Otherwise, go idle.
3. Within $2t_{\mathsf{upd}} + 2$ rounds, update both channels in $\mathsf{path}[0]$ and $\mathsf{path}[n-1]$ by cancelling out the corresponding amounts in each conditional payments with ids mid_0, mid_1 and paying mid_2. More specifically, for cancelling out, if $amt_{cxl} = amt_C$, then cancel the corresponding payments in both channels, otherwise,
   - In channel $\gamma_{A,B}$, remove the payment with id mid_1, and update the payment with id mid_0 by replacing the locked amount with $\mathsf{amt}_A - amt_{cxl}$.
   - In channel $\gamma_{B,C}$, remove the payment with id mid_1, and update the payment with id mid_0 by replacing the locked amount with $\mathsf{amt}_C - amt_{cxl}$.

   If any of the update fails, initiate channel closing for both channels in $\mathsf{path}[0]$ and $\mathsf{path}[n-1]$ (wrt. states before the update). If the updates are successful, send ($\mathsf{CANCEL{-}OK}, \mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2$) to B.

Pay and Reroute

Upon receiving ($\mathsf{REDUCE{-}CP}, \mathsf{mid}_1, \mathsf{mid}_2, \mathsf{path}[i]$) from $\mathsf{path}[i].\mathsf{payee}$ at round t_i (for $i = n-2$ to 1),

1. Check that mid_1 and mid_2 are stored in $\mathcal{HTLC}_{BO}$ with the corresponding path, if not go idle.
2. For $i = n-2$, check if the nullify phase is completed. Otherwise, do not continue. For $i < n-2$, check if $\mathsf{path}[i+1]$ is paid for mid_2. Otherwise, do not continue.
3. Update the channel of $\mathsf{path}[i]$ by paying the corresponding amount in the conditional payment with id mid_2.
4. Update the path of mid_0: replace the two conditional payments in mid_0, path_A and path_C, with the ones in mid_1, $\mathsf{path}_2[1], \ldots, \mathsf{path}_1[n-2]$.
5. Send ($\mathsf{REDUCE{-}OK}, \mathsf{path}[i]$) to $\mathsf{path}[i].\mathsf{payee}$.

To evaluate expected behavior of nullify operation, we utilize the balance security definition given in [5, 19, 33]:

Definition 1 (Balance Security). *No honest party involved in the nullify protocol loses her coins except the paid fees.*

Balance security for B implies that the total balance of B is preserved except the fees paid to the other parties. For the other parties, it refers that the total balance of an honest party is not decreased. Here, if A is the sender of MHP_0, and if the payment is successfully finalized, then A would pay the payment amount. However, this is not considered as losing coins as it is an intentional payment. It would have been seen as losing coins if the receiver would not receive the coins.

Note that for the multi-hop payment operations, we utilize the same functionality of the Lightning Network. The ideal functionality for multi-hop payments $\mathcal{F}_{MHP}$ and its realization in the Lightning Network ensure the balance security of an intermediary party under the assumption that the timelock difference between two consecutive HTLCs is adequate for an honest party to react. Also, each intermediary party locks their coins once they are ensured to be paid for the same locking condition by the previous party. In our security analysis, we can rely on the security guarantees of the MHP functionality, which we refer to as MHP balance security assumption (MHP-BSA). Note that this assumption does not take into account the wormhole attack, which targets the fee of honest parties [34].

Theorem 1. *The ideal functionality $\mathcal{F}_{BO}$ satisfies balance security of honest parties under the MHP balance security assumption.*

Proof. We discuss the balance security in four cases: B, the neighbors A and C, and the bailout parties D_i's. For each case, we show that the honest parties do not lose their coins regardless of the actions of others, and the status of other ongoing MHPs including MHP_0.

Case of B: At the beginning of the protocol there are two conditional payments in HTLC_A and HTLC_C (of MHP_0) where he is guaranteed that if he pays HTLC_C to C, he can claim the same amount (plus fee) via HTLC_A from A via MHP-BSA.

In the Setup and Lock phase, B creates two MHPs, MHP_1 and MHP_2. Before locking these, the status of the ongoing HTLC_A and HTLC_C are checked. If they are finalized, then there is no need to operate new MHPs. Once both MHPs are successfully locked, then Setup and Lock phase is completed and B can rely on the standard MHP guarantees. Also, since B is the sender/receiver of MHP_1 and MHP_2, he is the party who can start the unlocking of these new MHPs, which ensures that no one can claim the new HTLC payments of MHP_1 and MHP_2 otherwise.

In the Cancellation phase, B first updates his channels with both parties $P \in \{A, C\}$ by (partial) canceling the existing HTLCs. Both channels are updated simultaneously. For the channel $\gamma_{A,B}$, there are three existing HTLCs: HTLC_A, $\mathsf{MHP}_1[0]$, and $\mathsf{MHP}_2[0]$. In this phase, HTLC_A and $\mathsf{MHP}_1[0]$ are canceled, HTLC_A^{new} is locked and $\mathsf{MHP}_2[0]$ is paid. Note that HTLC_A and $\mathsf{MHP}_1[0]$ are in the opposite directions and have a difference in the amount of HTLC_A^{new}. If $amt_{cxl} = amt_C$ and both channels are fully cancelled, then, the difference between HTLC_A and $\mathsf{MHP}_1[0]$ is equal to the fee of B from MHP_0 (see Step (1) of Setup and Lock phase of $\mathcal{F}_{BO}$). The difference can be considered part of the fee paid to A for the cancellation operation. Thus, B does not lose his coins with the (partial) cancellation of HTLC_A and $\mathsf{MHP}_1[0]$, and locking HTLC_A^{new}. Similarly, the channel $\gamma_{B,C}$ is updated by cancelling HTLC_C, $\mathsf{MHP}_1[n-1]$, locking HTLC_C^{new} and paying $\mathsf{MHP}_2[n-1]$ where the amount in HTLC_C is equal to the summation of the amounts in $\mathsf{MHP}_1[n-1]$ and HTLC_C^{new}. Here, again, B does not lose coins with the cancellations in $\gamma_{B,C}$. If any of the channel updates of $\gamma_{A,B}$ or $\gamma_{B,C}$ fails, then both updates are cancelled and the channel closing procedure is started (see Step (3) of Cancellation phase of $\mathcal{F}_{BO}$)). In this case, balance of B is preserved via MHP-BSA.

Case of A: For party A, initially, there is a conditional payment HTLC_A where A pays to B. Depending on A being the sender of MHP_0, there could be another conditional payment before. In both cases, the balance security regarding only MHP_0 is guaranteed by MHP-BSA.

In the Setup and Lock phases, two new MHPs are created. Both MHPs are locked via the MHP functionality where A is ensured to be paid first before locking for a payment. In the Cancellation phase, A updates her channel with B by canceling HTLC_A and $\mathsf{MHP}_1[0]$ and locking HTLC_A^{new}. Since $\mathsf{HTLC}_A.amt = \mathsf{MHP}_1[0].amt + \mathsf{HTLC}_A^{new}.amt$, A does not lose any coins with the cancellation. If both channels are fully cancelled, the fee gain of A is increased by f_B. Moreover, A is paid by $\mathsf{MHP}_2[0]$ if the updates are successful. If the updates fail, the channel is closed. Then, for all three existing MHPs (as HTLC_A^{new} is not locked), the balance security of A is guaranteed by MHP-BSA.

In the Pay and Reroute phase, A pays to D_1 in $\mathsf{MHP}_2[1]$ only if the Cancellation phase is completed (see Step (1) of Pay and Reroute phase of $\mathcal{F}_{BO}$), which includes the payment of $\mathsf{MHP}_2[0]$ where $\mathsf{MHP}_2[0].amt - \mathsf{MHP}_2[1].amt = f_1$ (see Step (1) of Setup and Lock phase of $\mathcal{F}_{BO}$). Thus, in this case, A gains f_1 fee from MHP_2.

Overall, it can be seen that HTLC_A is replaced with HTLC_A^{new} and $\mathsf{MHP}_1[1]$ (see Step (4) of Pay and Reroute phase of $\mathcal{F}_{BO}$) where the timelocks are the same. Thus, A does not lose any coins in the protocol, but might earn the fee of B in MHP_0 (in the case of full cancellation), in addition to $\mathsf{MHP}_2[0]$.

Case of C: The case of C is similar to A in the sense that for both parties the protocol replaces their connection in the path from B to bailout party. The only difference is that the amounts in HTLC_A and HTLC_C are not the same. Yet, since HTLC_C and $\mathsf{MHP}_1[n-1]$ have the same amount, C does not lose any coins by canceling both of them. Thus, we omit the detailed discussion of this case because of the similarities with the previous case.

Case of D_i: At the beginning there are no ongoing HTLCs. In the Setup phase, D_i is involved in two MHP payments, MHP_1 and MHP_2. As in previous cases, the balance security at this stage is secured by MHP-BSA. Note that if there are multiple D_i's, then the timelock value is divided among the bailout parties. It is responsibility of an honest D_i to assess if the offered timelock value is adequate or not.

In the Pay and Reroute phase, payments of MHP_2 are paid in which first D_i receives the payment. Here, D_i earns a fee. Finally, the conditional payments in MHP_1 are moved to MHP_0. From D_i's perspective, there is no difference since MHP-BSA applies here as well. Thus, D_i does not lose any coins as well.

Now, we can show that our protocol `Bailout` emulates the ideal functionality $\mathcal{F}_{BO}$. We prove this by showing that any attack applied on $\mathit{\Pi}_{BO}$ can be simulated on $\mathcal{F}_{BO}$ as well. More specifically, we design a simulator $\mathcal{S}$ that simulates any attack of an adversary $\mathcal{A}$ on the protocol $\mathit{\Pi}_{BO}$ into the ideal functionality $\mathcal{F}_{BO}$.

This way, we show that an environment $\mathcal{E}$ cannot distinguish the real world with Π_{BO} from the ideal world with $\mathcal{F}_{BO}$.

Theorem 2. *Let $\mathcal{H}$ be a cryptographic hash function and Σ be a* $\mathsf{EUF\text{–}CMA}$ *secure signature scheme. Then, the protocol Π_{BO} UC-*realizes *the ideal functionality $\mathcal{F}_{BO}$.*

In general, the challenge in UC-realization is that a simulator is required to provide an indistinguishable transcript in the ideal world wrt. the real execution of the protocol without knowing the secret inputs of the parties. In our case, parties do not obtain secret values, but receive commands from environment $\mathcal{E}$. Thus, the only challenge is handling the behavior of the adversary, i.e., ensuring the same messages/transactions are seen at the same rounds by the environment who observes both real and ideal worlds.

Note that the indistinguishability of real and ideal worlds relies on the security of the $\mathcal{H}$ and Σ primitives. More specifically, the cryptographically secure hash function $\mathcal{H}$ ensures that the preimage conditions in the HTLCs are only satisfied with a unique "correct" secret value, which can only be known by the receiver. The $\mathsf{EUF\text{–}CMA}$ secure signature scheme Σ ensures that the signature of a party cannot be forged without knowing the corresponding private key. These properties ensure that the adversary cannot (i) obtain the preimage of the hash condition unless it is revealed (ii) sign a message/transaction on behalf of other parties. In other words, it prevents an adversary to execute an unauthorized operation, which would lead to the distinguishability of real and ideal worlds. We present the simulator code below.

D.1 Simulator for Our Protocol

In order to show that our protocol Π_{BO} emulates the ideal functionality $\mathcal{F}_{BO}$, we prove that any attack applied on Π_{BO} can be simulated on $\mathcal{F}_{BO}$ as well. More specifically, we design a simulator $\mathcal{S}$ that simulates any attack of an adversary $\mathcal{A}$ on the protocol Π_{BO} into the ideal functionality $\mathcal{F}_{BO}$. This way, we show that an environment $\mathcal{E}$ cannot distinguish the real world with Π_{BO} from the ideal world with $\mathcal{F}_{BO}$.

Here, we present the simulator for each phase of the protocol. Note that we do not provide the simulator for the case where all parties are honest since it is a straightforward consequence of the protocol steps. We investigate the cases with one honest party (for each role) and the rest is malicious. These cases already cover the rest of the cases where there are multiple honest parties.

Simulator for Setup and Lock

Honest B, Dishonest $\mathsf{path}[i].\mathsf{payee}$ (for $i = 0, \ldots, n-2$)

1. If B sends $(\mathsf{SETUP}, \mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mhpInfo}_1, \mathsf{mid}_2, \mathsf{mhpInfo}_2)$ to $\mathcal{F}_{BO}$,
   (a) Parse $\mathsf{mhpInfo}_1 := (\mathsf{amt}_1, \mathsf{TL}_1, \mathsf{path}_1)$ and $\mathsf{mhpInfo}_2 := (\mathsf{amt}_2, \mathsf{TL}_2, \mathsf{path}_2)$.
   (b) Check the following conditions:

– Check the paths are the same, i.e., $\mathsf{path}_1 = \mathsf{path}_2 := \mathsf{path}$. Let $A := \mathsf{path}[1].\mathsf{payer}$, $C := \mathsf{path}[n-1].\mathsf{payer}$, and $\mathsf{path}[i].\mathsf{payee} = D_i = \mathsf{path}[i+1].\mathsf{payer}$ for $i \in [1, n-3]$.
– Check the ongoing HTLCs with id mid_0 with A and C. Let $path_A$ and $path_B$ be part of the path of mid_0 with direction from A to B to C. Check if there are ongoing and locked HTLCs with id mid_0 in these paths. If not, stop. Otherwise, let HTLC_A and HTLC_C be the corresponding HTLCs with $(h, \mathsf{amt}_A, TL_A)$ and $(h, \mathsf{amt}_C, TL_C)$ hash condition, the locked amount and the timelock, respectively.
– Check whether the MHPs are properly generated:
  - Amount: For $i \in [0, n-1]$, check $\mathsf{amt}_1[i] := amt_{cxl} \leq amt_C$ and $\mathsf{amt}_2[i] = \sum_{j=i}^{n-1} f_j$ where f_j is the fee of jth channel.
  - Timelocks: Check $\mathsf{TL}_1 = \mathsf{TL}_2 := \mathsf{TL}$, and $\mathsf{TL}[0] = TL_A + \Delta$, $\mathsf{TL}[n-1] = TL_C - \Delta$, and for $i \in [1, n-2]$, $\mathsf{TL}[i] = \frac{(n-2-i)}{n-3} \times (TL_A - TL_C) + TL_C$.

If any of the checks fails, do not continue.

2. If B sends both messages $(\mathsf{INIT\text{–}MHP}, \mathsf{mid}_1, \mathsf{mhpInfo}_1)$ and $(\mathsf{INIT\text{–}MHP}, \mathsf{mid}_2, \mathsf{mhpInfo}_2)$ to $\mathcal{F}_{BO}$, continue to the next step. If only one is received, or none, then do nothing.
   (a) Let h be the condition of the HTLCs with id mid_0. Choose a random value x_B, and compute $h_B = \mathcal{H}(x_B)$. Assign $\mathsf{cond}_1 = \{h, h_B\}$ and $\mathsf{cond}_2 = \{h_B\}$
   (b) Setup the MHPs $\mathsf{oMHP}_1[0] \leftarrow \mathtt{SetupMHP}(\mathsf{mid}_1, \mathsf{amt}_1, \mathsf{TL}_1, \mathsf{cond}_1, \mathsf{path}_1)$ and $\mathsf{oMHP}_2[0] \leftarrow \mathtt{SetupMHP}(\mathsf{mid}_2, \mathsf{amt}_2, \mathsf{TL}_2, \mathsf{cond}_2, \mathsf{path}_2)$. Obtain $(\mathsf{MHP}_1[0], \mathsf{oMHP}_1[1])$ and $(\mathsf{MHP}_2[0], \mathsf{oMHP}_2[1])$ by decrypting $\mathsf{oMHP}_1[0]$ and $\mathsf{oMHP}_2[0]$.
   (c) Send $(\mathsf{LockMHP}, \mathsf{MHP}_1[0], \mathsf{oMHP}_1[1])$ and $(\mathsf{LockMHP}, \mathsf{MHP}_2[0], \mathsf{oMHP}_2[1])$ to $\mathsf{path}[0].\mathsf{payee}$, and follow the MHP protocol, Π_{MHP}, execute the simulator code for locking both MHPs.
3. Upon receiving $(\mathsf{LockMHP}, \mathsf{MHP}_1[n-1], \mathsf{oMHP}_1[n])$ and $(\mathsf{LockMHP}, \mathsf{MHP}_2[n-1], \mathsf{oMHP}_2[n])$ from $\mathsf{path}[n-1].\mathsf{payer}$, follow Π_{MHP} protocol, execute the simulator code for locking both MHPs.
   (a) If both $\mathsf{MHP}_1[n-1]$ and $\mathsf{MHP}_2[n-1]$ are locked, then assume the Setup and Lock phase is successful, and continue to the next phase.

Honest $\mathsf{path}[i].\mathsf{payee}$, Dishonest B, $\mathsf{path}[j].\mathsf{payee}$

(for $j \neq i$, $i \leq n-2$)

1. Upon receiving both $(\mathsf{LockMHP}, \mathsf{MHP}_1[i], \mathsf{oMHP}_1[i+1])$ and $(\mathsf{LockMHP}, \mathsf{MHP}_2[i], \mathsf{oMHP}_2[i+1])$ from $\mathsf{path}[i].\mathsf{payer}$, follow Π_{MHP} protocol, execute the simulator code for locking both MHPs.
   (a) If $\mathsf{MHP}_1[i]$ and $\mathsf{MHP}_2[i]$ are locked, then send $(\mathsf{LockMHP}, \mathsf{MHP}_1[i+1], \mathsf{oMHP}_1[i+2])$ and $(\mathsf{LockMHP}, \mathsf{MHP}_2[i+1], \mathsf{oMHP}_2[i+2])$ to $\mathsf{path}[i+1].\mathsf{payee}$, and follow Π_{MHP} protocol, execute the simulator code for locking both MHPs. Otherwise, stop.

Simulator for Cancellation

Honest B, Dishonest A and C

1. If B sends $(\mathsf{CANCEL}, \mathsf{CxlInfo}_1, \mathsf{CxlInfo}_2)$ to $\mathcal{F}_{BO}$ at round τ_0,
   (a) Parse both $\mathsf{CxlInfo}_1 := (\mathsf{mid}_0,$ $\mathsf{mid}_1, \mathsf{mid}_2, \mathsf{path}[0])$ and $\mathsf{CxlInfo}_2 := (\mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2, \mathsf{path}[n-1])$.
   (b) Let $\mathsf{HTLC}_{A,1} \leftarrow \mathsf{MHP}_1[0]$, $\mathsf{HTLC}_{A,2} \leftarrow \mathsf{MHP}_2[0]$, $\mathsf{HTLC}_{C,1} \leftarrow \mathsf{MHP}_1[n-1]$ and $\mathsf{HTLC}_{C,2} \leftarrow \mathsf{MHP}_2[n-1]$. Create the new HTLCs for the channels $\gamma_{A,B}$ and $\gamma_{B,C}$:
      – Use the same MHP id mid_0, and assign unique ids cpid_A^{new} and cpid_C^{new}.
      – $\mathsf{HTLC}_A^{new} \leftarrow (\mathsf{mid}_0, \mathsf{cpid}_A^{new}, \gamma_{A,B}, A \rightarrow B, h, TL_A, \mathsf{amt}_A - amt_{cxl})$
      – $\mathsf{HTLC}_C^{new} \leftarrow (\mathsf{mid}_0, \mathsf{cpid}_A^{new}, \gamma_{B,C}, B \rightarrow C, h, TL_C, \mathsf{amt}_C - amt_{cxl})$
      Create the temprorary HTLC for channel between B and C, $\gamma_{B,C}$:
      – Assign unique ids mid' and cpid'.
      – $\mathsf{HTLC}'_C \leftarrow (\mathsf{mid}', \mathsf{cpid}', \gamma_{B,C}, C \rightarrow B, h_B, TL_C - \Delta, \mathsf{amt}_C)$
   (c) Let $\mathcal{HTLC}_C \leftarrow \{\mathsf{HTLC}_C, \mathsf{HTLC}_{C,1}, \mathsf{HTLC}_{C,2}, \mathsf{HTLC}_C^{new}, \mathsf{HTLC}'_C\}$ and $\mathcal{HTLC}_A \leftarrow \{\mathsf{HTLC}_A, \mathsf{HTLC}_{A,1}, \mathsf{HTLC}_A^{new}, \mathsf{HTLC}_{A,2}\}$. At time τ_1, send $(\mathsf{Cancel}, \gamma_{B,C}, \mathcal{HTLC}_C)$ to C, and $(\mathsf{Cancel}, \gamma_{A,B}, \mathcal{HTLC}_A)$ to A.
2. Upon receiving messages $(\mathsf{UpdateOk}, \gamma_{B,C})$ from C and $(\mathsf{UpdateOk}, \gamma_{A,B})$ from A, send $(\mathsf{CANCEL}, \mathsf{CxlInfo}_1)$ to $\mathcal{F}_{BO}$ on behalf of A, and send $(\mathsf{CANCEL},$ $\mathsf{CxlInfo}_2)$ to $\mathcal{F}_{BO}$ on behalf of C if they have not sent the messages.
   (a) Initiate channel updates for both $\gamma_{B,C}$ and $\gamma_{A,B}$ at round τ_1:
      – Execute simulator code for `RevokeHTLC`(HTLC_C), `LockHTLC`(HTLC_C^{new}), `RevokeHTLC`$(\mathsf{HTLC}_{C,1})$, and `LockHTLC2`(HTLC'_C) simultaneously in the same channel update of $\gamma_{B,C}$.
      – Execute simulator code for `RevokeHTLC`(HTLC_A), `LockHTLC`(HTLC_A^{new}), `RevokeHTLC`$(\mathsf{HTLC}_{A,1})$ simultaneously in the same channel update of $\gamma_{A,B}$.
      Here before revoking the previous states of the channels, wait for both parties A and C to revoke. If any of them is not revoked within $\tau_1 + t_{\mathtt{upd}}$, then execute channel closing of $\gamma_{A,B}$ and $\gamma_{B,C}$ with both parties. Otherwise continue.
3. If both channels $\gamma_{A,B}$ and $\gamma_{B,C}$ are updated with the corresponding revoking and locking payments,
   (a) Initiate channel updates for $\gamma_{A,B}$ and $\gamma_{B,C}$ at round τ_2:
      – Execute `PayHTLC`(HTLC'_C, x_B) and `PayHTLC`$(\mathsf{MHP}_2[n-1], x_B)$ in channel $\gamma_{B,C}$.
      – Execute `PayHTLC`$(\mathsf{MHP}_2[0], x_B)$ in channel $\gamma_{A,B}$.
4. If the updates fail, then execute the simulator code for channel closing of $\gamma_{A,B}$ and $\gamma_{B,C}$ with A and C.
5. If the updates are completed, then assume the Nullify phase is successful, and continue to the next phase.

Honest C, Dishonest B and A

1. If C sends $(\mathsf{CANCEL}, \mathsf{CxlInfo}_2)$ to $\mathcal{F}_{BO}$,

(a) Wait until receiving ($\mathsf{Cancel}, \gamma_{B,C}, \mathcal{HTLC}_C$) from B. If not such a message is received, then do not continue.

(b) Parse $\mathsf{mhpInfo}_2 := (\mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2, \mathsf{path}[n-1])$ and $\mathcal{HTLC}_C := \{\mathsf{HTLC}_C, \mathsf{HTLC}_{C,1}, \mathsf{HTLC}_{C,2}, \mathsf{HTLC}_C^{new}, \mathsf{HTLC}'_C\}$, check correctness of the HTLCs:

  – Timelock: $\mathsf{HTLC}_C^{new}.TL \stackrel{?}{=} \mathsf{HTLC}_C.TL \stackrel{?}{=} \mathsf{HTLC}_{C,1}.TL + \Delta \stackrel{?}{=} \mathsf{HTLC}_{C,2}.TL + \Delta \stackrel{?}{=} \mathsf{HTLC}'_C.TL + \Delta$.

  – Hash: $\mathsf{HTLC}_C.\mathsf{cond} = \mathsf{HTLC}_C^{new}.\mathsf{cond} = \{h\}$, $\mathsf{HTLC}_{C,1}.\mathsf{cond} = \{h, h_B\}$, $\mathsf{HTLC}_{C,2}.\mathsf{cond} = \{h_B\}$, and $\mathsf{HTLC}'_C.\mathsf{cond} = \{h_B\}$.

  – Amount: $\mathsf{HTLC}_{C,1}.amt + \mathsf{HTLC}_C^{new}.amt \stackrel{?}{=} \mathsf{HTLC}_C.amt \stackrel{?}{=} \mathsf{HTLC}'_C.amt$ and $\mathsf{HTLC}_{C,2}.amt \geq f_C$ where f_C is the channel fee.

  – If any of them fails, do not continue and initiate channel closing of $\gamma_{B,C}$, and execute the simulator code.

(c) If all the checks are successful, send ($\mathsf{UpdateOk}, \gamma_{B,C}$) to B, and continue.

2. If initiated by B, follow the simulator code for the channel update protocol for execution of `RevokeHTLC`(HTLC_C), `LockHTLC`(HTLC_C^{new}), `RevokeHTLC`($\mathsf{HTLC}_{C,1}$), and `LockHTLC2`(HTLC'_C).
3. Once B initiates `PayHTLC`(HTLC'_C, x_B) and `PayHTLC`($\mathsf{MHP}_2[n-1], x_B$),

(a) Check $\mathcal{H}(x_B) = \mathsf{HTLC}'_C.\mathsf{cond}$, if it fails, then do not continue. Otherwise, follow the simulator code for the update of the channel $\gamma_{B,C}$.

Honest A, Dishonest B and C

1. If A sends ($\mathsf{CANCEL}, \mathsf{CxlInfo}_1$) to $\mathcal{F}_{BO}$,

(a) Wait until receiving ($\mathsf{Cancel}, \gamma_{A,B}, \mathcal{HTLC}_A$) from B. If not such a message is received, then do not continue.

(b) Parse $\mathsf{mhpInfo}_1 := (\mathsf{mid}_0, \mathsf{mid}_1, \mathsf{mid}_2, \mathsf{path}[0])$ and $\mathcal{HTLC}_A := \{\mathsf{HTLC}_A, \mathsf{HTLC}_{A,1}, \mathsf{HTLC}_A^{new}, \mathsf{HTLC}_{A,2}\}$, check correctness of the HTLCs:

  – Timelock: $\mathsf{HTLC}_A.TL \stackrel{?}{=} \mathsf{HTLC}_A^{new}.TL \stackrel{?}{=} \mathsf{HTLC}_{A,1}.TL - \Delta \stackrel{?}{=} \mathsf{HTLC}_{A,2}.TL - \Delta$.

  – Hash: $\mathsf{HTLC}_A.\mathsf{cond} = \mathsf{HTLC}_A^{new}.\mathsf{cond} = \{h\}$, $\mathsf{HTLC}_{A,1}.\mathsf{cond} = \{h, h_B\}$ and $\mathsf{HTLC}_{A,2}.\mathsf{cond} = \{h_B\}$.

  – Amount: $\mathsf{HTLC}_{A,1}.amt + \mathsf{HTLC}_A^{new}.amt \stackrel{?}{=} \mathsf{HTLC}_A.amt$ and $\mathsf{HTLC}_{A,2}.amt \geq f_A$ where f_A is the channel fee.

  – If any of them fails, do not continue and initiate channel closing of $\gamma_{A,B}$, and execute the simulator code.

(c) If all the checks are successful, send ($\mathsf{UpdateOk}, \gamma_{A,B}$) to B, and continue.

2. If initiated by B, follow the simulator code for the channel update protocol for execution of `RevokeHTLC`(HTLC_A), `LockHTLC`(HTLC_A^{new}), and `RevokeHTLC`($\mathsf{HTLC}_{A,1}$).
3. Once B initiates `PayHTLC`($\mathsf{MHP}_2[0], x_B$),

(a) Check $\mathcal{H}(x_B) = \mathsf{MHP}_2[0].\mathsf{cond}$, if it fails, then do not continue. Otherwise, follow the simulator code for the update of the channel $\gamma_{A,B}$.

Simulator for Pay and Reroute

Honest $\mathsf{path}[i].\mathsf{payee}$**, Dishonest** B**,** $\mathsf{path}[j].\mathsf{payee}$
(for $j \neq i$, $1 \leq i \leq n-2$)

1. If $\mathsf{path}[i].\mathsf{payee}$ sends $(\mathsf{REDUCE{-}CP}, \mathsf{mid}_1, \mathsf{mid}_2, \mathsf{path}[i])$ to $\mathcal{F}_{BO}$,
   (a) Distinguish the following cases:
   - For $i = n-2$, i.e., $\mathsf{path}[i].\mathsf{payee} = C$, check if the nullify phase is completed by canceling HTLC with id mid_1. Otherwise, do not continue.
   - For $i < n-2$, check if HTLC in $\mathsf{path}[i+1]$ is paid with id mid_2. Otherwise, do not continue.

   (b) Let x_B the corresponding value revealed in the previous step. Execute the simulator code for $\mathtt{PayHTLC}(\mathsf{MHP}_2[i], x_B)$ by revealing x_B to C.
   (c) Send $(\mathsf{ReduceCP}, \mathsf{MHP}_1[i])$ to $\mathsf{MHP}_1[i].\mathsf{payer}$.
2. Once $\mathsf{path}[i].\mathsf{payer}$ initiates $\mathtt{RevokeHTLC}(\mathsf{MHP}_1[i])$ and $\mathtt{LockHTLC}(\mathsf{MHP}'_1[i])$,
   (a) Follow the simulator code for channel update protocol. If the update is completed, then assume Pay and Reduce Condition phase is successful.

Honest $\mathsf{path}[i].\mathsf{payer}$**, Dishonest** B**,** $\mathsf{path}[j].\mathsf{payer}$

(for $j \neq i$, $1 \leq i \leq n-2$)

1. Once $\mathsf{path}[i].\mathsf{payee}$ initiates $\mathtt{PayHTLC}(\mathsf{MHP}_2[i], x_B)$,
   (a) Check $\mathcal{H}(x_B) = \mathsf{MHP}_2[i].\mathsf{cond}$, if it fails, then do not continue. Otherwise, follow the simulator code for channel update protocol.

   Upon receiving $(\mathsf{ReduceCP}, \mathsf{MHP}_1[i])$ from $\mathsf{MHP}_1[i].\mathsf{payee}$,
   (a) If $\mathsf{MHP}_2[i]$ is paid, then initiate the simulator code for channel update with $\mathsf{MHP}_1[i].\mathsf{payee}$:
   - Execute $\mathtt{RevokeHTLC}(\mathsf{MHP}_1[i])$ and $\mathtt{LockHTLC}(\mathsf{MHP}'_1[i])$ simultaneously in the same channel update where the only difference between the HTLCs is the condition and $\mathsf{MHP}'_1[i].\mathsf{cond} := h$.

E Multi-hop Payment Functionality

In this section, we first present the ideal functionality for two-round multi-hop payments. Then, we provide a simplified version of Lightning's MHP protocol Π_{MHP}. Finally, we present a simulator that shows that Π_{MHP} UC-realizes $\mathcal{F}_{MHP}$.

E.1 Ideal Functionality for MHP

$\mathcal{F}_{MHP}$ Ideal Functionality for MHP

The ideal functionality $\mathcal{F}_{MHP}$ utilizes the channel functionality $\mathcal{F}_{chan}$. It maintains the set of ongoing multi-hop payments $\mathcal{MHP}$ and conditional payments $\mathcal{HTLC}$.

Setup and Lock

1. Upon receiving ($\mathsf{INIT–MHP}, \mathsf{mid}, \mathsf{mhpInfo}$) from $\mathsf{Sender} := \mathsf{path}[0].\mathsf{payer}$ where $\mathsf{mhpInfo} := (\mathsf{amt}, \mathsf{TL}, \mathsf{path})$
   - Wait until to receive ($\mathsf{INIT–MHP}, \mathsf{mid}, \mathsf{mhpInfo}$) from $\mathsf{Receiver} := \mathsf{path}[n-1].\mathsf{payee}$ where $n = |\mathsf{path}|$, register ($\mathsf{mid}, \mathsf{mhpInfo}, \mathtt{status} = \mathit{initiated}$) into $\mathcal{MHP}$, and send ($\mathsf{INITIATED}, \mathsf{mid}$) to $\mathsf{Receiver}$. Otherwise, stop.
2. Upon receiving ($\mathsf{LOCK–MHP}, \mathsf{mid}, \mathsf{path}[i]$) from party $\mathsf{path}[i].\mathsf{payer}$ at round t_i; for $i = 0$, i.e., $\mathsf{path}[0].\mathsf{payer} = \mathsf{Sender}$ where there is a corresponding MHP registered as initiated with id mid, update the status with *setup* in $\mathcal{MHP}$. For $i \geq 1$ where there is a MHP registered with $\mathtt{status} = \mathit{setup}$ for id mid, check if $\mathsf{path}[i-1]$ is registered with $\mathtt{status} = \mathit{locked}$ in $\mathcal{HTLC}$. If not, then stop.
   - Until $t_i + t_{\mathsf{upd}}$, if the channel balance permits, lock the amount $\mathsf{amt}[i]$ in channel $\mathsf{path}[i]$ from the balance of $\mathsf{path}[i].\mathsf{payer}$, i.e., remove them from $\mathsf{path}[i].\mathsf{payer}$'s available balance, add them to locked coin outputs, and keep track of them.
   - If the channel is successfully updated with the corresponding locking, send ($\mathsf{LOCKED}, \mathsf{mid}, \mathsf{path}[i]$) to $\mathsf{path}[i].\mathsf{payer}$ and $\mathsf{path}[i].\mathsf{payee}$ and register ($\mathsf{mid}, \mathsf{amt}[i], \mathsf{TL}[i], \mathsf{path}[i], \mathtt{status} = \mathit{locked}$) in $\mathcal{HTLC}$, and continue. Otherwise, send ($\mathsf{FAILED}, \mathsf{mid}, \mathsf{path}[i]$) to $\mathsf{path}[i].\mathsf{payer}$ and $\mathsf{path}[i].\mathsf{payee}$ and register ($\mathsf{mid}, \mathsf{amt}[i], \mathsf{TL}[i], \mathsf{path}[i], \mathtt{status} = failed$) in $\mathcal{HTLC}$. Change status of mid in $\mathcal{MHP}$ to $failed$, and do not continue.
3. Once all the channels in path are locked including $\mathsf{path}[n-1]$, update status of mid in $\mathcal{MHP}$ to *locked*, and continue.

<u>Pay or Revoke</u>

1. Upon receiving ($\mathsf{PAY–MHP}, \mathsf{mid}, \mathsf{path}[i]$) from $\mathsf{path}[i].\mathsf{payee}$ at round τ_i, where mid is registered with status *locked*; for $i = n-1$, i.e., $\mathsf{path}[n-1].\mathsf{payee} = \mathsf{Receiver}$, continue. For $i < n-1$, check if $\mathsf{path}[i+1]$ is registered with $\mathtt{status} = \mathit{paid}$. If not, then stop.
   - Until $\tau_i + t_{\mathsf{upd}}$, pay the corresponding locked amount $\mathsf{amt}[i]$ in channel $\mathsf{path}[i]$ to the balance of $\mathsf{path}[i].\mathsf{payee}$. Otherwise, initiate channel closing procedure.
   - If the payment is successful, send ($\mathsf{PAYED}, \mathsf{mid}, \mathsf{path}[i]$) to $\mathsf{path}[i].\mathsf{payer}$ and $\mathsf{path}[i].\mathsf{payee}$ and update the corresponding status in $\mathcal{HTLC}$ with *paid*, and continue. Otherwise, send ($\mathsf{FAILED}, \mathsf{mid}, \mathsf{path}[i]$) to $\mathsf{path}[i].\mathsf{payer}$ and $\mathsf{path}[i].\mathsf{payee}$, update the status with $failed$ and initiate channel closing procedure.
2. Upon receiving ($\mathsf{REVOKE–MHP}, \mathsf{mid}, \mathsf{path}[i]$) from $\mathsf{path}[i].\mathsf{payer}$ at round τ'_i where $\mathsf{path}[i]$ is registered with status *locked* and $\tau'_i \geq \mathsf{TL}[i]$,
   - Until $\tau'_i + t_{\mathsf{upd}}$, revoke the payment by re-paying corresponding locked amount $\mathsf{amt}[i]$ in channel $\mathsf{path}[i]$ to the balance of $\mathsf{path}[i].\mathsf{payer}$. Otherwise, initiate channel closing procedure.
   - If the revoke is successful, send ($\mathsf{REVOKED}, \mathsf{mid}, \mathsf{path}[i]$) to $\mathsf{path}[i].\mathsf{payer}$ and $\mathsf{path}[i].\mathsf{payee}$ and and update the corresponding status in $\mathcal{HTLC}$ with *revoked*, and continue. Otherwise, send ($\mathsf{FAILED}, \mathsf{mid}, \mathsf{path}[i]$) to $\mathsf{path}[i]$ $.\mathsf{payer}$ and $\mathsf{path}[i].\mathsf{payee}$, update the status with $failed$ and initiate channel closing procedure.

E.2 Simplified version of Lightning Multi-Hop Payment Protocol

In the Setup and Lock phase, the payment route is created, and each channel in the path locks wrt. the conditional payments. A MHP setup is executed by calling $\texttt{SetupMHP}(\mathsf{mid}, \mathsf{amt}, \mathsf{TL}, \mathsf{cond}, \mathsf{path})$ function. Here, mid is the unique id of the payment, and amt denotes the payment amounts in each channel (including the fees). cond is randomly chosen by $\mathsf{Receiver}$ for which he possesses a preimage ($\mathsf{witness}$) satisfying the condition. TL are the timelocks determined for each channel in the path.

Note that for the privacy concerns, each intermediary party is allowed to know the previous and the following party in the path. The MHP payment MHP is transmitted via onion routing. More specifically, each party $\mathsf{MHP}[i].\mathsf{payer}$ receives $\mathsf{oMHP}[i] := \mathsf{Enc}_{pk_i}(\mathsf{MHP}[i], \mathsf{Enc}_{pk_{i+1}}(\mathsf{MHP}[i{+}1], \ldots, \mathsf{Enc}_{pk_{n-1}}(\mathsf{MHP}[n-1]$)) where Enc is the public key encryption scheme. Using the private key sk_i, $\mathsf{MHP}[i].\mathsf{payer}$ can extract $\mathsf{MHP}[i]$ and pass the rest $\mathsf{oMHP}[i{+}1]$ to $\mathsf{MHP}[i].\mathsf{payee} = \mathsf{MHP}[i+1].\mathsf{payer}$. The origin of the onion structure, $\mathsf{oMHP}[0]$ is created by Sender using the public keys of each parties in the payment route.

In the Pay or Revoke phase, the payment is completed with success or revocation. First, the receiver provides the preimage ($\mathsf{witness}$) to $\mathsf{path}[n-1].\mathsf{payer}$. If the preimage is valid, i.e.,
$\mathcal{H}(\mathsf{witness}) = \mathsf{cond}$, then the corresponding channel of $\mathsf{MHP}[n-1]$ is updated by completing the payment. In the same manner, party $\mathsf{path}[n-1].\mathsf{payer}$ reveals the preimage to $\mathsf{path}[n-2].\mathsf{payer}$ to get paid in $\mathsf{MHP}[n-2]$. This goes until the first channel in the path is also updated with payment. If the receiver does not reveal the preimage on time, or the process stops at a malicious intermediary, each party $\mathsf{path}[i].\mathsf{payer}$ can re-claim the payment at the timelock time $\mathsf{MHP}[i].TL$. If one of the parties in a channel does not accept to update the channel (in payment or revocation), the other party initiates the closure of the channel to obtain her coins on the blockchain.

Π_{MHP} Multi-hop Payments with HTLCs

The hybrid protocol Π_{MHP} utilizes the channel functionality $\mathcal{F}_{chan}$.

Setup and Lock

Let Sender and $\mathsf{Receiver}$ be the sender and receiver of the payment, respectively. The conditional payments are generated wrt. the hard relation R.

$\boxed{\mathsf{Sender}}$

Upon receiving $(\mathsf{INIT{-}MHP}, \mathsf{mid}, \mathsf{mhpInfo})$ from $\mathcal{E}$,

1. Parse $\mathsf{mhpInfo} := (\mathsf{amt}, \mathsf{TL}, \mathsf{path})$.
2. Check $\mathsf{path}[0].\mathsf{payer} = \mathsf{Sender}$, and continue. Otherwise, stop.
3. Store $(\mathsf{mid}, \mathsf{amt}, \mathsf{TL}, \mathsf{path})$ as an initiated MHP.

$\boxed{\mathsf{Receiver}}$

Upon receiving ($\mathsf{INIT\text{–}MHP}, \mathsf{mid}, \mathsf{mhpInfo}$) from $\mathcal{E}$,

1. Parse $\mathsf{mhpInfo} := (\mathsf{amt}, \mathsf{TL}, \mathsf{path})$.
2. Check $\mathsf{Receiver} = \mathsf{path}[n-1].\mathsf{payee}$ where $n = |\mathsf{path}|$.
3. Check $\mathsf{amt} > 0$. If not, stop. Otherwise, continue.
4. Choose a random secret $\mathsf{witness}$ and compute the condition cond such that $(\mathsf{witness}, \mathsf{cond}) \in R$. Store $(\mathsf{witness}, \mathsf{mid})$.
5. Store $(\mathsf{mid}, \mathsf{amt}, \mathsf{TL}, \mathsf{path}, \mathsf{cond})$ as an initiated MHP.
6. Send $(\mathsf{InitOk}, \mathsf{mid}, \mathsf{cond})$ to Sender and $(\mathsf{INITIATED}, \mathsf{mid})$ to $\mathcal{E}$.

$\boxed{\mathsf{Sender}}$

Upon receiving $(\mathsf{InitOk}, \mathsf{mid}, \mathsf{cond})$ from $\mathsf{Receiver}$ and $(\mathsf{LOCK\text{–}MHP}, \mathsf{mid}, \mathsf{path}[0])$ from $\mathcal{E}$ at round t_0,

1. Execute $\mathsf{oMHP}[0] \leftarrow \mathtt{SetupMHP}(\mathsf{mid}, \mathsf{amt}, \mathsf{TL}, \mathsf{cond}, \mathsf{path})$.
2. Obtain $\mathsf{MHP}[0]$ and $\mathsf{oMHP}[1]$ by decrypting $\mathsf{oMHP}[0]$.
3. Send $(\mathsf{LockMHP}, \mathsf{MHP}[0], \mathsf{oMHP}[1])$ to party $\mathsf{MHP}[0].\mathsf{payee}$.
4. Upon receiving $(\mathsf{LockOk}, \mathsf{MHP}[0])$ from party $\mathsf{MHP}[0].\mathsf{payee}$ until $t_0 + 1$, execute $\mathtt{LockHTLC}(\mathsf{MHP}[0])$. Otherwise, stop.
5. Distinguish the following cases:
   - Upon receiving $\mathsf{Success}$, assume $\mathsf{MHP}[0].\mathsf{mid}$ as locked, send $(\mathsf{LOCKED}, \mathsf{mid}, \mathsf{path}[0])$ to $\mathcal{E}$, and continue.
   - Upon receiving Fail, assume $\mathsf{MHP}[0].\mathsf{mid}$ as failed, send $(\mathsf{FAILED}, \mathsf{mid}, \mathsf{path}[0])$ to $\mathcal{E}$, and do not continue.

$\boxed{\mathsf{MHP}[i].\mathsf{payee} = \mathsf{MHP}[i+1].\mathsf{payer} \text{ (for } i = 0, \ldots, n-2)}$

Upon receiving $(\mathsf{LockMHP}, \mathsf{MHP}[i], \mathsf{oMHP}[i+1])$ from $\mathsf{MHP}[i].\mathsf{payer}$,

1. Within a round, send $(\mathsf{LockOk}, \mathsf{MHP}[i])$ to party $\mathsf{MHP}[i].\mathsf{payer}$.
2. Distinguish the following cases:
   - Upon receiving $(\mathsf{Updated}, \mathsf{MHP}[i].\gamma.id, \vec{\theta})$ from $\mathcal{F}_{chan}$ where $\vec{\theta}$ includes the conditional payment $\mathsf{MHP}[i]$, assume $\mathsf{MHP}[i]$ as locked, send $(\mathsf{LOCKED}, \mathsf{mid}, \mathsf{path}[0])$ to $\mathcal{E}$, and continue.
   - If no such message is received within at most t_{upd} rounds, assume $\mathsf{MHP}[i]$ as failed, send $(\mathsf{FAILED}, \mathsf{mid}, \mathsf{path}[0])$ to $\mathcal{E}$, and do not continue.

Upon receiving $(\mathsf{LOCK\text{–}MHP}, \mathsf{mid}, \mathsf{path}[i+1])$ from $\mathcal{E}$ at round t_{i+1},

1. Obtain $\mathsf{MHP}[i+1]$ and $\mathsf{oMHP}[i+2]$ by decrypting $\mathsf{oMHP}[i+1]$.
2. Execute $\mathtt{CheckMHP}(\mathsf{MHP}[i], \mathsf{MHP}[i+1])$, if it returns Fail, stop.
3. Send $(\mathsf{LockMHP}, \mathsf{MHP}[i+1], \mathsf{oMHP}[i+2])$ to party $\mathsf{MHP}[i+1].\mathsf{payee}$.
4. Upon receiving $(\mathsf{LockOk}, \mathsf{MHP}[i+1])$ from party $\mathsf{MHP}[i+1].\mathsf{payee}$ until $t_{i+1}+1$, execute $\mathtt{LockHTLC}(\mathsf{MHP}[i+1])$. Otherwise, stop.
5. Distinguish the following cases:
   - Upon receiving $\mathsf{Success}$, assume $\mathsf{MHP}[i+1].\mathsf{mid}$ as locked, send $(\mathsf{LOCKED}, \mathsf{mid}, \mathsf{path}[i+1])$ to $\mathcal{E}$, and continue.
   - Upon receiving Fail, assume $\mathsf{MHP}[i+1].\mathsf{mid}$ as failed, send $(\mathsf{FAILED}, \mathsf{mid}, \mathsf{path}[i+1])$ to $\mathcal{E}$, and do not continue.

$\boxed{\text{Receiver}}$

Upon receiving $(\mathsf{LockMHP}, \mathsf{MHP}[n-1], \mathsf{oMHP}[n])$ from $\mathsf{MHP}[n-1].\mathsf{payer}$,

1. Within a round, send $(\mathsf{LockOk}, \mathsf{MHP}[n-1])$ to party $\mathsf{MHP}[n-1].\mathsf{payer}$.
2. Distinguish the following cases:
   – Upon receiving $(\mathsf{Updated}, \mathsf{MHP}[n-1].\gamma.id, \vec{\theta})$ from $\mathcal{F}_{chan}$ where $\vec{\theta}$ includes the conditional payment $\mathsf{MHP}[n-1]$, assume $\mathsf{MHP}[n-1].\mathsf{mid}$ as locked, send $(\mathsf{LOCKED}, \mathsf{mid}, \mathsf{path}[i+1])$ to $\mathcal{E}$, and continue to the Pay and Revoke phase.
   – If no such message is received within at most t_{upd} rounds, assume $\mathsf{MHP}[n-1].\mathsf{mid}$ as failed, send $(\mathsf{FAILED}, \mathsf{mid}, \mathsf{path}[i+1])$ to $\mathcal{E}$, and do not continue.

<u>Pay or Revoke</u>

$\boxed{\text{Receiver}}$

Upon receiving $(\mathsf{PAY{-}MHP}, \mathsf{mid}, \mathsf{path}[n-1])$ from $\mathcal{E}$ at round τ_{n-1},

1. Let $(\mathsf{witness}, \mathsf{MHP}[n-1].\mathsf{mid})$ be the stored pair of witness and id of the conditional payment $\mathsf{MHP}[n-1].\mathsf{mid}$ which is locked. Send $(\mathsf{PayMHP}, \mathsf{MHP}[n-1], \mathsf{witness})$ to $\mathsf{MHP}[n-1].\mathsf{payer}$.
2. Upon receiving $(\mathsf{PayOk}, \mathsf{MHP}[n-1])$ from party $\mathsf{MHP}[n-1].\mathsf{payer}$ until $\tau_{n-1}+1$, execute $\mathtt{PayHTLC}(\mathsf{MHP}[n-1], \mathsf{witness})$. Otherwise, send $(\mathsf{Close}, \mathsf{MHP}[n-1].\gamma.id)$ to $\mathcal{F}_{chan}$.
3. Distinguish the following cases:
   – Upon receiving $\mathsf{Success}$, assume $\mathsf{MHP}[n-1].\mathsf{mid}$ as paid, and send $(\mathsf{PAYED}, \mathsf{mid}, \mathsf{path}[n-1])$ to $\mathcal{E}$.
   – Upon receiving Fail, assume $\mathsf{MHP}[n-1].\mathsf{mid}$ as failed, send $(\mathsf{FAILED}, \mathsf{mid}, \mathsf{path}[n-1])$ to $\mathcal{E}$, and send $(\mathsf{Close}, \mathsf{MHP}[n-1].\gamma.id)$ to $\mathcal{F}_{chan}$.

$\boxed{\mathsf{MHP}[i].\mathsf{payer} = \mathsf{MHP}[i-1].\mathsf{payee} \text{ (for } i = n-1, \ldots, 1)}$

Upon receiving $(\mathsf{PayMHP}, \mathsf{MHP}[i], \mathsf{witness})$ from $\mathsf{MHP}[i].\mathsf{payee}$,

1. Execute $\mathtt{CheckCond}(\mathsf{witness}, \mathsf{MHP}[i].\mathsf{cond})$, if it returns Fail, then send $(\mathsf{Close}, \mathsf{MHP}[i].\gamma.id)$ to $\mathcal{F}_{chan}$. Otherwise, within a round, send $(\mathsf{PayOk}, \mathsf{MHP}[i])$ from party $\mathsf{MHP}[i].\mathsf{payee}$.
2. Distinguish the following cases:
   – Upon receiving $(\mathsf{Updated}, \mathsf{MHP}[i].\gamma.id, \vec{\theta})$ from $\mathcal{F}_{chan}$ where $\vec{\theta}$ excludes the conditional payment $\mathsf{MHP}[i]$, assume $\mathsf{MHP}[i].\mathsf{mid}$ as paid, and send $(\mathsf{PAYED}, \mathsf{mid}, \mathsf{path}[n-1])$ to $\mathcal{E}$.
   – If no such message is received within at most t_{upd} rounds, assume $\mathsf{MHP}[i].\mathsf{mid}$ as failed, send $(\mathsf{FAILED}, \mathsf{mid}, \mathsf{path}[n-1])$ to $\mathcal{E}$.

Upon receiving $(\mathsf{PAY{-}MHP}, \mathsf{mid}, \mathsf{path}[i-1])$ from $\mathcal{E}$ at round τ_{i-1}

1. Let $\mathsf{witness}$ be the witness obtained by paying $\mathsf{MHP}[i]$, send $(\mathsf{PayMHP}, \mathsf{MHP}[i-1], \mathsf{witness})$ to $\mathsf{MHP}[i-1].\mathsf{payer}$. Otherwise, stop.

2. Upon receiving $(\mathsf{PayOk}, \mathsf{MHP}[i-1])$ from party $\mathsf{MHP}[i-1].\mathsf{payer}$ until $\tau_i + 1$, execute $\texttt{PayHTLC}(\mathsf{MHP}[i-1], \mathsf{witness})$. Otherwise, send $(\mathsf{Close}, \mathsf{MHP}[i-1].\gamma.id)$ to $\mathcal{F}_{chan}$.
3. Distinguish the following cases:
   - Upon receiving $\mathsf{Success}$, assume $\mathsf{MHP}[i-1].\mathsf{mid}$ as paid, and send $(\mathsf{PAYED}, \mathsf{mid}, \mathsf{path}[i-1])$ to $\mathcal{E}$.
   - Upon receiving Fail, assume $\mathsf{MHP}[i-1].\mathsf{mid}$ as failed, and send $(\mathsf{FAILED}, \mathsf{mid}, \mathsf{path}[i-1])$ to $\mathcal{E}$, send $(\mathsf{Close}, \mathsf{MHP}[i-1].\gamma.id)$ to $\mathcal{F}_{chan}$.

Upon receiving $(\mathsf{REVOKE{-}MHP}, \mathsf{mid}, \mathsf{path}[i])$ from $\mathcal{E}$ at round τ_i' where $\tau_i' \geq \mathsf{MHP}[i].\mathsf{TL}$,

1. Send $(\mathsf{RevokeMHP}, \mathsf{MHP}[i])$ to $\mathsf{MHP}[i].\mathsf{payee}$.
2. Upon receiving $(\mathsf{RevokeOk}, \mathsf{MHP}[i])$ from party $\mathsf{MHP}[i].\mathsf{payee}$ until $\tau_i' + 1$, execute $\texttt{RevokeHTLC}(\mathsf{MHP}[i])$. Otherwise, send $(\mathsf{Close}, \mathsf{MHP}[i].\gamma.id)$ to $\mathcal{F}_{chan}$.
3. Distinguish the following cases:
   - Upon receiving $\mathsf{Success}$, assume $\mathsf{MHP}[i].\mathsf{mid}$ as revoked, send $(\mathsf{REVOKED}, \mathsf{mid}, \mathsf{path}[i])$ to $\mathcal{E}$.
   - Upon receiving Fail, assume $\mathsf{MHP}[i].\mathsf{mid}$ as failed, send $(\mathsf{FAILED}, \mathsf{mid}, \mathsf{path}[i])$ to $\mathcal{E}$, send $(\mathsf{Close}, \mathsf{MHP}[i].\gamma.id)$ to $\mathcal{F}_{chan}$.

Sender

Upon receiving $(\mathsf{PayMHP}, \mathsf{MHP}[0], \mathsf{witness})$ from $\mathsf{MHP}[0].\mathsf{payee}$,

1. Execute $\texttt{CheckCond}(\mathsf{witness}, \mathsf{MHP}[0].\mathsf{cond})$, if it returns Fail, then send $(\mathsf{Close}, \mathsf{MHP}[0].\gamma.id)$ to $\mathcal{F}_{chan}$. Otherwise, within a round, send $(\mathsf{PayOk}, \mathsf{MHP}[0])$ from party $\mathsf{MHP}[0].\mathsf{payee}$.
2. Distinguish the following cases:
   - Upon receiving $(\mathsf{Updated}, \mathsf{MHP}[0].\gamma.id, \vec{\theta})$ from $\mathcal{F}_{chan}$ where $\vec{\theta}$ excludes the conditional payment $\mathsf{MHP}[0]$, assume $\mathsf{MHP}[0].\mathsf{mid}$ as paid, and send $(\mathsf{PAYED}, \mathsf{mid}, \mathsf{path}[i-1])$ to $\mathcal{E}$.
   - If no such message is received within at most t_{upd} rounds, assume $\mathsf{MHP}[0].\mathsf{mid}$ as failed, and send $(\mathsf{FAILED}, \mathsf{mid}, \mathsf{path}[i-1])$ to $\mathcal{E}$.

Upon receiving $(\mathsf{REVOKE{-}MHP}, \mathsf{mid}, \mathsf{path}[0])$ from $\mathcal{E}$ at round τ_0' where $\tau_0' \geq \mathsf{MHP}[0].\mathsf{TL}$,

1. Send $(\mathsf{RevokeMHP}, \mathsf{MHP}[0])$ to $\mathsf{MHP}[0].\mathsf{payee}$.
2. Upon receiving $(\mathsf{RevokeOk}, \mathsf{MHP}[0])$ from party $\mathsf{MHP}[0].\mathsf{payee}$ until $\tau_0' + 1$, execute $\texttt{RevokeHTLC}(\mathsf{MHP}[0])$. Otherwise, send $(\mathsf{Close}, \mathsf{MHP}[0].\gamma.id)$ to $\mathcal{F}_{chan}$.
3. Distinguish the following cases:
   - Upon receiving $\mathsf{Success}$, assume $\mathsf{MHP}[0].\mathsf{mid}$ as revoked, send $(\mathsf{REVOKED}, \mathsf{mid}, \mathsf{path}[0])$ to $\mathcal{E}$.
   - Upon receiving Fail, assume $\mathsf{MHP}[0].\mathsf{mid}$ as failed, send $(\mathsf{FAILED}, \mathsf{mid}, \mathsf{path}[0])$ to $\mathcal{E}$, send $(\mathsf{Close}, \mathsf{MHP}[0].\gamma.id)$ to $\mathcal{F}_{chan}$.

E.3 Simulator for Multi-hop payment

In order to show that simplified Lightning MHP protocol Π_{MHP} emulates the ideal functionality $\mathcal{F}_{MHP}$, we prove that any attack applied on Π_{MHP} can be simulated on $\mathcal{F}_{MHP}$ as well. More specifically, we design a simulator $\mathcal{S}$ that simulates any attack of an adversary $\mathcal{A}$ on the protocol Π_{MHP} into the ideal functionality $\mathcal{F}_{MHP}$. This way, we show that an environment $\mathcal{E}$ cannot distinguish, the real world with Π_{MHP} from the ideal world with $\mathcal{F}_{MHP}$.

Here, we present the simulator for each phase of the protocol. Like in Section D.1, here, we do not explain the simulator for the case where all parties are honest since it is straightforward follow of the protocol steps.

Simulator for Setup and Lock

Honest Sender**, Dishonest** $\mathsf{MHP}[i].\mathsf{payee}$ **(for** $i \in [0, n-1]$**)**

1. If Sender sends $(\mathsf{INIT{-}MHP}, \mathsf{mid}, \mathsf{mhpInfo})$ to $\mathcal{F}_{MHP}$,
   (a) Parse $\mathsf{mhpInfo} := (\mathsf{amt}, \mathsf{TL}, \mathsf{path})$.
   (b) Check $\mathsf{path}[0].\mathsf{payer} = \mathsf{Sender}$, and continue. Otherwise, stop.
   (c) Store $(\mathsf{mid}, \mathsf{mhpInfo})$ as an initiated MHP.
2. Upon receiving $(\mathsf{InitOk}, \mathsf{mid}, \mathsf{cond})$ from $\mathsf{Receiver}$, send $(\mathsf{INIT{-}MHP}, \mathsf{mid}, \mathsf{mhpInfo})$ to $\mathcal{F}_{MHP}$ on behalf of $\mathsf{Receiver}$, if $\mathsf{Receiver}$ has not send this message.
3. If Sender sends $(\mathsf{LOCK{-}MHP}, \mathsf{mid}, \mathsf{path}[0])$ to $\mathcal{F}_{MHP}$ at round t_0,
   (a) Execute $\mathsf{oMHP}[0] \leftarrow \mathtt{SetupMHP}(\mathsf{mid}, \mathsf{amt}, \mathsf{TL}, \mathsf{cond}, \mathsf{path})$.
   (b) Obtain $\mathsf{MHP}[0]$ and $\mathsf{oMHP}[1]$ by decrypting $\mathsf{oMHP}[0]$.
   (c) Send $(\mathsf{LockMHP}, \mathsf{MHP}[0], \mathsf{oMHP}[1])$ to $\mathsf{MHP}[0].\mathsf{payee}$.
   (d) Upon receiving $(\mathsf{LockOk}, \mathsf{MHP}[0])$ from $\mathsf{MHP}[0].\mathsf{payee}$ until t_0+1, execute simulator code for $\mathtt{LockHTLC}(\mathsf{MHP}[0])$. Otherwise, stop.
   (e) Distinguish the following cases:
      – Upon receiving Fail, assume $\mathsf{MHP}[0].\mathsf{mid}$ as failed, and do not continue.
      – Upon receiving $\mathsf{Success}$, assume $\mathsf{MHP}[0].\mathsf{mid}$ as locked, and continue.

Honest $\mathsf{Receiver}$**, Dishonest** $\mathsf{MHP}[i].\mathsf{payer}$ **(for** $i \in [0, n-1]$**)**

1. If $\mathsf{Receiver}$ sends $(\mathsf{INIT{-}MHP}, \mathsf{mid}, \mathsf{mhpInfo})$ to $\mathcal{F}_{MHP}$,
   (a) Parse $\mathsf{mhpInfo} := (\mathsf{amt}, \mathsf{TL}, \mathsf{path})$.
   (b) Check $\mathsf{Receiver} = \mathsf{path}[n-1].\mathsf{payee}$ where $n = |\mathsf{path}|$.
   (c) Check $\mathsf{amt} > 0$. If not, stop. Otherwise, continue.
   (d) Choose a random witness $\mathsf{witness}$ and compute the condition cond such that $(\mathsf{witness}, \mathsf{cond}) \in R$. Store $(\mathsf{witness}, \mathsf{mid})$.
   (e) Store $(\mathsf{mid}, \mathsf{mhpInfo}, \mathsf{cond})$ as an initiated MHP.
   (f) Send $(\mathsf{InitOk}, \mathsf{mid}, \mathsf{cond})$ to Sender.
2. Send $(\mathsf{INIT{-}MHP}, \mathsf{mid}, \mathsf{mhpInfo})$ to $\mathcal{F}_{MHP}$ on behalf of Sender, if Sender has not send this message.
3. Upon receiving $(\mathsf{LockMHP}, \mathsf{MHP}[n-1], \mathsf{oMHP}[n])$ from $\mathsf{MHP}[n-1].\mathsf{payer}$,
   (a) Within a round, send $(\mathsf{LockOk}, \mathsf{MHP}[n-1])$ to party $\mathsf{MHP}[n-1].\mathsf{payer}$, and execute the simulator code for locking.
   (b) Distinguish the following cases:
      – Upon receiving $(\mathsf{Updated}, \mathsf{MHP}[n-1].\gamma.id, \vec{\theta})$ from $\mathcal{F}_{chan}$ where $\vec{\theta}$ includes the conditional payment $\mathsf{MHP}[n-1]$, assume $\mathsf{MHP}[n-1].\mathsf{mid}$ as locked, and continue to the Pay and Revoke phase.

– If no such message is received within at most t_{upd} rounds, assume $\mathsf{MHP}[n-1].\mathsf{mid}$ as failed and do not continue.

Honest $\mathsf{MHP}[i].\mathsf{payee}$**, Dishonest** Sender**,** $\mathsf{MHP}[j].\mathsf{payee}$

(for $j \neq i$, $i \neq n-1$)

1. Upon receiving ($\mathsf{LockMHP}, \mathsf{MHP}[i], \mathsf{oMHP}[i+1]$) from party $\mathsf{MHP}[i].\mathsf{payer}$,
   (a) Within a round, send ($\mathsf{LockOk}, \mathsf{MHP}[i]$) to $\mathsf{MHP}[i].\mathsf{payer}$, and execute the simulator code for locking.
   (b) Distinguish the following cases:
      – Upon receiving ($\mathsf{Updated}, \mathsf{MHP}[i].\gamma.id, \vec{\theta}$) from $\mathcal{F}_{chan}$ where $\vec{\theta}$ includes the conditional payment $\mathsf{MHP}[i]$, assume $\mathsf{MHP}[i]$ as locked, and continue.
      – If no such message is received within at most t_{upd} rounds, assume $\mathsf{MHP}[i]$ as failed and do not continue.
2. Send ($\mathsf{INIT{-}MHP}, \mathsf{mid}, \mathsf{mhpInfo}$) to $\mathcal{F}_{MHP}$ on behalf of Sender and $\mathsf{Receiver}$, if they have not send the message.
3. If $\mathsf{MHP}[i].\mathsf{payee}$ sends ($\mathsf{LOCK{-}MHP}, \mathsf{mid}, \mathsf{path}[i+1]$) to $\mathcal{F}_{MHP}$ at round t_{i+1},
   (a) Obtain $\mathsf{MHP}[i+1]$ and $\mathsf{oMHP}[i+2]$ by decrypting $\mathsf{oMHP}[i+1]$.
   (b) Execute the simulator code for `CheckMHP`($\mathsf{MHP}[i], \mathsf{MHP}[i{+}1]$), if it returns Fail, stop.
   (c) Send ($\mathsf{LockMHP}, \mathsf{MHP}[i+1], \mathsf{oMHP}[i+2]$) to party $\mathsf{MHP}[i+1].\mathsf{payee}$.
   (d) Upon receiving ($\mathsf{LockOk}, \mathsf{MHP}[i+1]$) from party $\mathsf{MHP}[i+1].\mathsf{payee}$ until $t_{i+1}+1$, execute simulator code for `LockHTLC`($\mathsf{MHP}[i+1]$). Otherwise, stop.
   (e) Distinguish the following cases:
      – Upon receiving $\mathsf{Success}$, assume $\mathsf{MHP}[i+1].\mathsf{mid}$ as locked, and continue.
      – Upon receiving Fail, assume $\mathsf{MHP}[i+1].\mathsf{mid}$ as failed, and do not continue.

Simulator for Pay or Revoke

Honest Sender**, Dishonest** $\mathsf{MHP}[i].\mathsf{payee}$ **(for** $i \in [0, n-1]$**)**

1. Upon receiving ($\mathsf{PayMHP}, \mathsf{MHP}[0], \mathsf{witness}$) from the party $\mathsf{MHP}[0].\mathsf{payee}$,
   (a) Execute `CheckCond`($\mathsf{witness}, \mathsf{MHP}[0].\mathsf{cond}$), if it returns Fail, then initiate $\mathcal{F}_{MHP}$ for sending ($\mathsf{Close}, \mathsf{MHP}[0].\gamma.id$) to $\mathcal{F}_{chan}$ and execute the simulator code for closing procedure. Otherwise, within a round, send ($\mathsf{PayOk}, \mathsf{MHP}[0]$) from party $\mathsf{MHP}[0].\mathsf{payee}$.
   (b) Distinguish the following cases:
      – Upon receiving ($\mathsf{Updated}, \mathsf{MHP}[0].\gamma.id, \vec{\theta}$) from $\mathcal{F}_{chan}$ where $\vec{\theta}$ excludes the conditional payment $\mathsf{MHP}[0]$, assume $\mathsf{MHP}[0].\mathsf{mid}$ as paid, and continue.
      – If no such message is received within at most t_{upd} rounds, assume $\mathsf{MHP}[0].\mathsf{mid}$ as failed.

2. If Sender sends $(\mathsf{REVOKE\text{–}MHP}, \mathsf{mid}, \mathsf{path}[0])$ to $\mathcal{F}_{MHP}$ at round τ'_0 where $\tau'_0 \geq \mathsf{MHP}[0].\mathsf{TL}$,
   (a) Send $(\mathsf{RevokeMHP}, \mathsf{MHP}[0])$ to $\mathsf{MHP}[0].\mathsf{payee}$.
   (b) Upon receiving $(\mathsf{RevokeOk}, \mathsf{MHP}[0])$ from $\mathsf{MHP}[0].\mathsf{payee}$ until $\tau'_0 + 1$, execute the simulator code for $\mathtt{RevokeHTLC}(\mathsf{MHP}[0])$. If Sender sends $(\mathsf{Close}, \mathsf{MHP}[0].\gamma.id)$ to $\mathcal{F}_{chan}$, execute the simulator code for closing procedure.
   (c) Distinguish the following cases:
      – Upon receiving $\mathsf{Success}$, assume $\mathsf{MHP}[0].\mathsf{mid}$ as revoked, then send $(\mathsf{REVOKED}, \mathsf{mid}, \mathsf{path}[0])$ to $\mathcal{E}$.
      – Upon receiving Fail, assume $\mathsf{MHP}[0].\mathsf{mid}$ as failed. If Sender sends $(\mathsf{Close}, \mathsf{MHP}[0].\gamma.id)$ to $\mathcal{F}_{chan}$, execute the simulator code for closing procedure.

Honest Receiver**, Dishonest** $\mathsf{MHP}[i].\mathsf{payer}$ **(for** $i \in [0, n-1]$**)**

1. If $\mathsf{Receiver}$ sends $(\mathsf{PAY\text{–}MHP}, \mathsf{mid}, \mathsf{path}[n-1])$ to $\mathcal{F}_{MHP}$ at round τ_{n-1},
   (a) Let $(\mathsf{witness}, \mathsf{MHP}[n-1].\mathsf{mid})$ be the stored pair of witness and id of the conditional payment $\mathsf{MHP}[n-1].\mathsf{mid}$ which is locked. Send $(\mathsf{PayMHP}, \mathsf{MHP}[n-1], \mathsf{witness})$ to $\mathsf{MHP}[n-1].\mathsf{payer}$.
   (b) Upon receiving $(\mathsf{PayOk}, \mathsf{MHP}[n-1])$ from party $\mathsf{MHP}[n-1].\mathsf{payer}$ until $\tau_{n-1}+1$, execute the simulator code for $\mathtt{PayHTLC}(\mathsf{MHP}[n-1], \mathsf{witness})$. If $\mathsf{Receiver}$ sends $(\mathsf{Close}, \mathsf{MHP}[n-1].\gamma.id)$ to $\mathcal{F}_{chan}$, execute the simulator code for closing procedure.
   (c) Distinguish the following cases:
      – Upon receiving $\mathsf{Success}$, assume $\mathsf{MHP}[n-1].\mathsf{mid}$ as paid.
      – Upon receiving Fail, assume $\mathsf{MHP}[n-1].\mathsf{mid}$ as failed. If $\mathsf{Receiver}$ sends $(\mathsf{Close}, \mathsf{MHP}[n-1].\gamma.id)$ to $\mathcal{F}_{chan}$, execute the simulator code for closing procedure.

Honest $\mathsf{MHP}[i].\mathsf{payer}$**, Dishonest** Receiver**,** $\mathsf{MHP}[j].\mathsf{payer}$

(for $j \neq i$, $i \neq 0$)

1. Upon receiving $(\mathsf{PayMHP}, \mathsf{MHP}[i], \mathsf{witness})$ from the party $\mathsf{MHP}[i].\mathsf{payee}$,
   (a) Execute $\mathtt{CheckCond}(\mathsf{witness}, \mathsf{MHP}[i].\mathsf{cond})$, if it returns Fail, then initiate $\mathcal{F}_{MHP}$ for sending $(\mathsf{Close}, \mathsf{MHP}[i].\gamma.id)$ to $\mathcal{F}_{chan}$ and execute the simulator code for closing procedure. Otherwise, within a round, send $(\mathsf{PayOk}, \mathsf{MHP}[i])$ from party $\mathsf{MHP}[i].\mathsf{payee}$.
   (b) Distinguish the following cases:
      – Upon receiving $(\mathsf{Updated}, \mathsf{MHP}[i].\gamma.id, \vec{\theta})$ from $\mathcal{F}_{chan}$ where $\vec{\theta}$ excludes the conditional payment $\mathsf{MHP}[i]$, assume $\mathsf{MHP}[i].\mathsf{mid}$ as paid, and continue.
      – If no such message is received within at most t_{upd} rounds, assume $\mathsf{MHP}[i].\mathsf{mid}$ as failed.
2. If $\mathsf{MHP}[i].\mathsf{payer} = \mathsf{MHP}[i-1].\mathsf{payee}$ sends $(\mathsf{PAY\text{–}MHP}, \mathsf{mid}, \mathsf{path}[i-1])$ to $\mathcal{F}_{MHP}$ at round τ_{i-1},

(a) Let $(\mathsf{witness}, \mathsf{MHP}[i-1].\mathsf{mid})$ be the stored pair of witness and id of the conditional payment $\mathsf{MHP}[i-1].\mathsf{mid}$ which is locked. Send $(\mathsf{PayMHP}, \mathsf{MHP}[i-1], \mathsf{witness})$ to $\mathsf{MHP}[i-1].\mathsf{payer}$.

(b) Upon receiving $(\mathsf{PayOk}, \mathsf{MHP}[i-1])$ from party $\mathsf{MHP}[i-1].\mathsf{payer}$ until $\tau_{i-1}+1$, execute the simulator code for $\mathtt{PayHTLC}(\mathsf{MHP}[i-1], \mathsf{witness})$. If $\mathsf{MHP}[i].\mathsf{payer}$ sends $(\mathsf{Close}, \mathsf{MHP}[i-1].\gamma.id)$ to $\mathcal{F}_{chan}$, execute the simulator code for closing procedure.

(c) Distinguish the following cases:
   - Upon receiving $\mathsf{Success}$, assume $\mathsf{MHP}[i-1].\mathsf{mid}$ as paid.
   - Upon receiving Fail, assume $\mathsf{MHP}[i-1].\mathsf{mid}$ as failed. If $\mathsf{MHP}[i].\mathsf{payer}$ sends $(\mathsf{Close}, \mathsf{MHP}[i-1].\gamma.id)$ to $\mathcal{F}_{chan}$, execute the simulator code for closing procedure.

3. If $\mathsf{MHP}[i].\mathsf{payer}$ sends $(\mathsf{REVOKE\text{–}MHP}, \mathsf{mid}, \mathsf{path}[i])$ to $\mathcal{F}_{MHP}$ at round τ_i' where $\tau_i' \geq \mathsf{MHP}[i].\mathsf{TL}$,

(a) Send $(\mathsf{RevokeMHP}, \mathsf{MHP}[i])$ to $\mathsf{MHP}[i].\mathsf{payee}$.

(b) Upon receiving $(\mathsf{RevokeOk}, \mathsf{MHP}[i])$ from $\mathsf{MHP}[i].\mathsf{payee}$ until $\tau_i'+1$, execute the simulator code for $\mathtt{RevokeHTLC}(\mathsf{MHP}[i])$. If $\mathsf{MHP}[i].\mathsf{payer}$ sends $(\mathsf{Close}, \mathsf{MHP}[i].\gamma.id)$ to $\mathcal{F}_{chan}$, execute the simulator code for closing procedure.

(c) Distinguish the following cases:
   - Upon receiving $\mathsf{Success}$, assume $\mathsf{MHP}[i].\mathsf{mid}$ as revoked.
   - Upon receiving Fail, assume $\mathsf{MHP}[i].\mathsf{mid}$ as failed. If $\mathsf{MHP}[i].\mathsf{payer}$ sends $(\mathsf{Close}, \mathsf{MHP}[i].\gamma.id)$ to $\mathcal{F}_{chan}$, execute the simulator code for closing procedure.

F Experiments: Requiring Liquidity

In this second evaluation scenario, parties can bail out of payments at any time during the simulation. They bailout because they require the funds for a concurrent payment.

Metrics. For this scenario, bailing out of ongoing payments to realize concurrent payments should have the side effect of increasing the overall success probability of payments, as bailouts allow forwarding payments that otherwise fail. As a consequence, we also consider the *payment success ratio*, i.e., the overall fraction of payments that succeed, as a metric, in addition to the metrics listed in Section 4.

Simulation Setup. During the simulation, a party B can be prevented from initiating or forwarding a payment due to collateral locked in ongoing payments. If that happens, the party considers all ongoing payments that have been in progress for at least 60s. He checks whether bailing out of these payments enables him to complete the new payment. If yes, he checks if he can bailout. We consider only paths with one bailout party.

Once the party B has identified one or more potential bailout parties, B sends any potential bailout party D information about the payment, namely the value that needs to be locked on the two channels. If D's channels do not have sufficient balance, D denies participation. If there is sufficient balance, D names the fee he charges. B then chooses the bailout party. We consider two

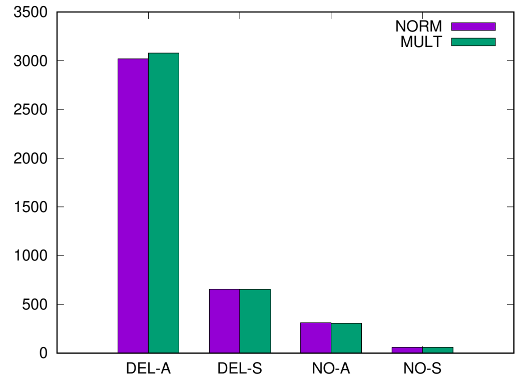


Fig. 7: Bailouts over time for fee strategies NORM and MULT considering both Delaying (DEL) and Not Settling (NO); A denotes the number of overall bailouts while S denotes the number of successful bailouts, $p = 0.1$

configurations for the fee charged by D: i) NORM: D charges the same fee as for a normal payment, and i) $MULT$: D multiplies its normal fee with a factor of 10. Given that payments that have been ongoing for a while are likely to only be resolved when the timeout expires rather than within seconds as most payments, a higher fee seems appropriate to make up for the longer than average time. The problem of how to exactly select this fee is out of scope for this paper and we just use a factor 10 as an example. Note that to include the aspect of selecting bailout parties based in the cheapest fee offered, we only choose one alternative path rather than splitting the amount over multiple paths as in our first scenario. We only consider the high concurrency scenario here, which has a higher chance to require bailouts.

Results. There are relatively few bailout events during the simulation and many of them are not successful. The failures are almost always due to the lack of a bailout party rather than a lack of funds. Figure 7 displays the number of overall bailout events and the number of successful bailouts over 100,000 transactions for $p = 0.1$. Note that the number of overall events includes all three options listed in Section 4. Results for other values of p show slightly higher values but a similar ratio between overall bailouts and successful ones.

We see that the choice of fee strategy — NORM or MULT — has no statistically significant impact on the number of bailouts, which justifies disregarding the aspect of fees in our first scenario. The behavior with regard to delays is a major factor: For *No Settling*, the number of bailouts are much lower. The reason lies in the fact that *No Settling* is only applied by parties who cannot forward a payment while delaying is an action that can be executed by any party on the path. Thus, there are more delayed payments for *Delaying* and thus, more parties are affected and need to bailout. Given the low number of successful bailouts, the impact on successful payment ratio is minimal, increasing the fraction from 0.66 to 0.67.