TU Delft

Delft University of Technology

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

Ersoy, Oğuzhan; Moreno-Sanchez, Pedro; Roos, Stefanie

DOI
10.1007/978-3-031-47751-5_6

Publication date
2024

Document Version
Final published version

Published in
Financial Cryptography and Data Security - 27th International Conference, FC 2023, Revised Selected Papers

Citation (APA)
Ersoy, O., Moreno-Sanchez, P., & Roos, S. (2024). Get Me Out of This Payment! Bailout: An HTLC Re-routing Protocol. In F. Baldimtsi, & C. Cachin (Eds.), *Financial Cryptography and Data Security - 27th International Conference, FC 2023, Revised Selected Papers* (pp. 92-109). (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Vol. 13951). Springer. https://doi.org/10.1007/978-3-031-47751-5_6

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

Green Open Access added to TU Delft Institutional Repository

'You share, we take care!' - Taverne project

https://www.openaccess.nl/en/you-share-we-take-care

Otherwise as indicated in the copyright section: the publisher
is the copyright holder of this work and the author uses the
Dutch legislation to make this work public.

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

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

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

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

© International Financial Cryptography Association 2024
F. Baldimtsi and C. Cachin (Eds.): FC 2023, LNCS 13951, pp. 92–109, 2024.
https://doi.org/10.1007/978-3-031-47751-5_6

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) [3–5,15,16,31,38] 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 [5,18,31], 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 [5,18,32,38], *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 Sect. 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 [36].

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 [36,40]. 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 [11]. 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 [7,8,14,24,26,34].

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. Details of transactions and ledger functionality are given in [21].

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 $P \in$

γ.users, γ.cash denotes the capacity of the channel and $\gamma.\mathsf{st}:=(\vec{\theta}_1,\ldots,\vec{\theta}_n)$ 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-chain. The payment channel functionality is given in [21].

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 $\mathsf{path}[i]$.payer (and also $\mathsf{MHP}[i]$.payer) denotes the ith party in the path who pays to the $(i+1)$th party, $\mathsf{path}[i]$.payee.

We present the ideal functionality of MHP $\mathcal{F}_{MHP}$ in [21], 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 $\mathsf{path}[i]$.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 $\mathsf{path}[i]$.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 $\mathsf{path}[i]$.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 HTLC.cpid is the id of the HTLC in channel HTLC.γ between the payer HTLC.payer and the payee HTLC.payee. If the HTLC is part of a multi-hop payment, then HTLC.mid stores the corresponding id, otherwise it is $\perp$. The payment amount of the HTLC is HTLC.amt that is locked for the condition HTLC.cond. If the HTLC is part of a MHP, the amount is deducted from the available coins of HTLC.payer. If a witness witness is provided s.t. $\mathcal{H}(\mathsf{witness})=\mathsf{cond}$ until time HTLC.TL, then the payment amount is given to HTLC.payee. Otherwise, at time HTLC.TL, the amount is returned to 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 [21].

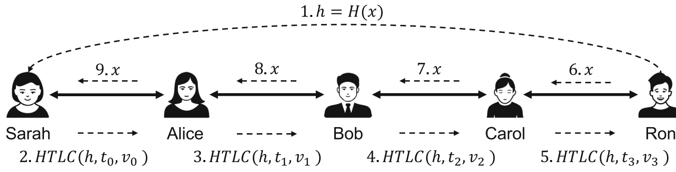


Fig. 1. A multi-hop payment with HTLCs. h denotes $\mathsf{MHP}[i].\mathsf{cond}$ where x is the corresponding preimage, and t_i and v_i represents $\mathsf{MHP}[i].TL$ and $\mathsf{MHP}[i].amt$.

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 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 Fig. 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 Fig. 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 $\gamma_{B,C}$, defined as: $\mathsf{HTLC}_A{:=}(A \rightarrow B, h, TL_A, amt_A)$ and $\mathsf{HTLC}_C{:=}(B \rightarrow C, h, TL_C, amt_C)$, 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 [26]. 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 Fig. 2).[1] 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 Fig. 2. The

[1] 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.

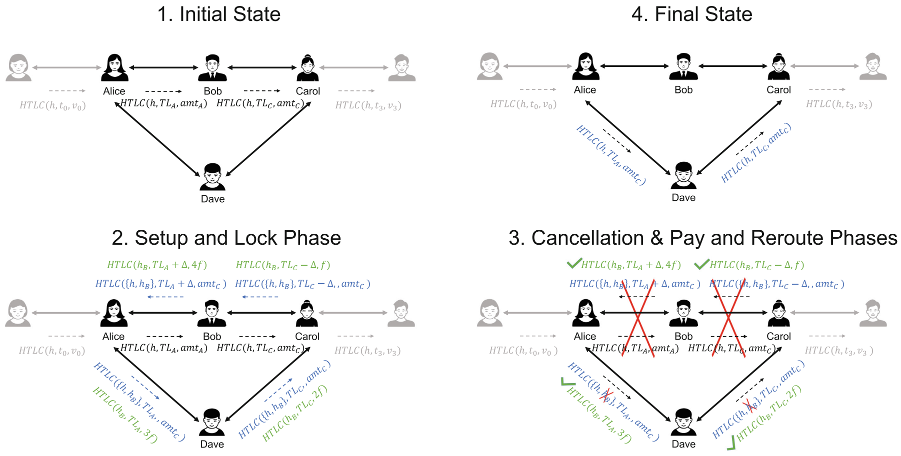


Fig. 2. Simplified protocol phases for the full cancellation/re-routing. In Setup and Lock Phase, the new multi-hop payments (MHP_1 and MHP_2) are locked. In Cancellation Phase, the HTLCs of B are cancelled in the channels with A and C. In Pay and Reroute Phase, MHP_2 is payed by sharing the preimage 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. (Color figure online)

re-routing of the original payment can be seen in the path difference between the Initial and Final State given in Fig. 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 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[2], 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

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

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 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 Fig. 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 min [36]. 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`

In [21], we give the protocol, Π_{BO} in the UC framework. Here, we explain the 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. 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 $i \in [0, n-1]$):

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

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 $\mathsf{MHP}_1[n-1].TL$:

$$\mathsf{HTLC}'_C \leftarrow (C \rightarrow B, h_B, TL_C - \Delta, amt_C) \tag{1}$$

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

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 HTLC_A^{new}:=$(A \rightarrow B, h, TL_A, amt_A - amt_{cxl})$.

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 execute $\mathsf{MHP}_2[0]$.

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

$$\mathsf{MHP}_1^{new}[i] = (\mathsf{payer}_i \rightarrow \mathsf{payee}_i, h, \mathsf{TL}[i], \mathsf{amt}_1[i]). \tag{2}$$

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 [21]. Also, in [21], 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 [21], 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[3]. We simulate the Lightning Network by using real-world topology snapshots. As 92% of parties use the LND client [36], 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 [12], 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:

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

– *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.

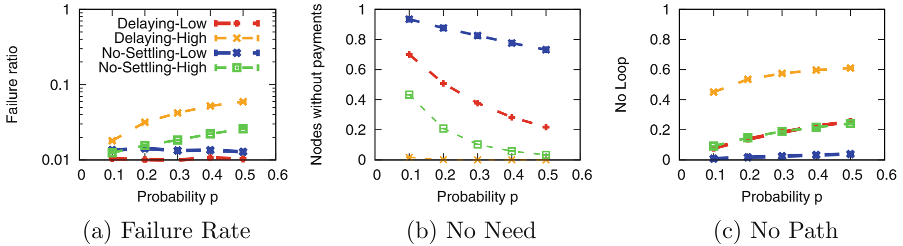
Parameters. We run our simulation on a real-world Lightning snapshot [39]. We restricted our evaluation to the largest connected component with nearly 7,000 nodes and about 65,000 channels to ensure that every node had a path to every other node. For each channel and direction, we choose the balance exponentially 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 [18]. 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 60 s such that any ongoing payments without additional delays can terminate. 60 s was chosen as Lightning payments should terminate within a minute [2]. During the 60 s, he no longer forwards new payments. After the 60 s, he attempts to bailout of all remaining payments. For simplicity, we assume that bailout parties are not paid fees here, but we consider them in [21].

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 'No Loop' to note 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.

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.



(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.

As concurrency has a major impact on the number of ongoing payments, we consider a low-concurrency and a high-concurrency scenario. In the former, a party on average sends 0.04 transactions per hour, or roughly 1 transaction per day. In the latter, 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 Sect. 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 [38], generalized channels [3,17], and virtual channels [4,6,15,16,23,25]. 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 [5,18,19,32,35,38].

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 [9,28,37,42], and (ii) *passive re-balancing* with fees and incentive mechanisms [13,20,41]. It is also possible to change the capacity, and thereby the balance, of a channel by Loop-in and Loop-out protocols [27], which require on-chain transactions. Recently, Spider [40] 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 [29,36,43], 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 [43], limiting the number of hops [36], or decreasing the locked time [5,35]. Also, recently, an alternative HTLC protocol with a griefing-penalty mechanism is proposed [33], 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 [7,8,14,24,26,34] 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* [30,31], 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., Boston (2017)
3. Aumayr, L.: Generalized channels from limited blockchain scripts and adaptor signatures. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13091, pp. 635–664. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92075-3_22
4. Aumayr, L., et al.: Bitcoin-compatible virtual channels. In: IEEE SP (2021)
5. Aumayr, L., Moreno-Sanchez, P., Kate, A., Maffei, M.: Blitz: secure multi-hop payments without two-phase commits. In: USENIX Security Symposium (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. In: 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). https://www.blockchain.com/charts/avg-confirmation-time
11. Community, L.N.: Lighning network specification. https://github.com/lightning/bolts/blob/master/02-peer-protocol.md#rationale-7
12. Community, L.N.: Lightning network specification. https://lightning-bolts.readthedocs.io/en/latest/
13. Conoscenti, M., Vetrò, A., Martin, J.C.D.: Hubs, rebalancing and service providers in the lightning network. IEEE Access **7**, 132828–132840 (2019)
14. Dryja, T., Milano, S.B.: Unlinkable outsourced channel monitoring. Scaling Bitcoin Milan (2016)
15. Dziembowski, S., Eckey, L., Faust, S., Hesse, J., Hostáková, K.: Multi-party virtual state channels. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 625–656. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_21
16. Dziembowski, S., Eckey, L., Faust, S., Malinowski, D.: Perun: virtual payment hubs over cryptocurrencies. In: IEEE SP (2019)
17. Dziembowski, S., Faust, S., Hostáková, K.: General state channel networks. In: CCS, pp. 949–966. ACM (2018)
18. Eckey, L., Faust, S., Hostáková, K., Roos, S.: Splitting payments locally while routing interdimensionally. IACR Cryptol. ePrint Arch. **2020**, 555 (2020)
19. Egger, C., Moreno-Sanchez, P., Maffei, M.: Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks. In: CCS (2019)
20. van Engelshoven, Y., Roos, S.: The merchant: avoiding payment channel depletion through incentives. In: DAPPS, pp. 59–68. IEEE (2021)
21. Ersoy, O., Moreno-Sanchez, P., Roos, S.: Get me out of this payment! bailout: an htlc re-routing protocol (full version). Cryptology ePrint Archive, Paper 2022/958 (2022). https://eprint.iacr.org/2022/958
22. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: SoK: layer-two blockchain protocols. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 201–226. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_12
23. Jourenko, M., Larangeira, M., Tanaka, K.: Lightweight virtual payment channels. In: CANS (2020)
24. Khabbazian, M., Nadahalli, T., Wattenhofer, R.: Outpost: a responsive lightweight watchtower. In: ACM AFT (2019)
25. Kiayias, A., Litos, O.S.T.: Elmo: recursive virtual payment channels for bitcoin. IACR Cryptol. ePrint Arch., p. 747 (2021)
26. Lab, T.M.D.C.I..M.: Watchtower - watch channels for fraudulent transactions (2018). https://github.com/mit-dci
27. Labs, L.: Loop. https://lightning.engineering/loop/
28. Li, P., Miyazaki, T., Zhou, W.: Secure balance planning of off-blockchain payment channel networks. In: INFOCOM, pp. 1728–1737. IEEE (2020)
29. Lu, Z., Han, R., Yu, J.: General congestion attack on HTLC-based payment channel networks. IACR Cryptol. ePrint Arch., p. 456 (2020)
30. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M.: Silentwhispers: enforcing security and privacy in credit networks. In: NDSS (2017)
31. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., Ravi, S.: Concurrency and privacy with payment-channel networks. In: ACM CCS (2017)
32. Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Anonymous multi-hop locks for blockchain scalability and interoperability. In: NDSS (2019)

33. Mazumdar, S., Banerjee, P., Ruj, S.: Griefing-penalty: countermeasure for griefing attack in lightning network. arXiv preprint arXiv:2005.09327 (2020)
34. McCorry, P., Bakshi, S., Bentov, I., Meiklejohn, S., Miller, A.: Pisa: arbitration outsourcing for state channels. In: ACM AFT (2019)
35. Miller, A., Bentov, I., Bakshi, S., Kumaresan, R., McCorry, P.: Sprites and state channels: payment networks that go faster than lightning. In: FC (2019)
36. Mizrahi, A., Zohar, A.: Congestion attacks in payment channel networks. In: Borisov, N., Diaz, C. (eds.) FC 2021. LNCS, vol. 12675, pp. 170–188. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-662-64331-0_9
37. Pickhardt, R., Nowostawski, M.: Imbalance measure and proactive channel rebalancing algorithm for the lightning network. In: IEEE ICBC, pp. 1–5. IEEE (2020)
38. Poon, J., Dryja, T.: The bitcoin lightning network: scalable off-chain instant payments (2016). https://lightning.network/lightning-network-paper.pdf
39. roher: discharged-pc-data (github project). https://git.tu-berlin.de/rohrer/discharged-pc-data/
40. Sivaraman, V., et al.: High throughput cryptocurrency routing in payment channel networks. In: NSDI, pp. 777–796. USENIX Association (2020)
41. Stasi, G.D., Avallone, S., Canonico, R., Ventre, G.: Routing payments on the lightning network. In: iThings/GreenCom/CPSCom/SmartData. IEEE (2018)
42. Subramanian, L.M., Eswaraiah, G., Vishwanathan, R.: Rebalancing in acyclic payment networks. In: PST, pp. 1–5. IEEE (2019)
43. Tochner, S., Zohar, A., Schmid, S.: Route hijacking and dos in off-chain networks. In: AFT, pp. 228–240. ACM (2020)