Griefing-Penalty: Countermeasure for Griefing Attack in Lightning Network

Subhra Mazumdar\textsuperscript{a,}\textsuperscript{*}, Prabal Banerjee\textsuperscript{a,c}, Sushmita Ruj\textsuperscript{a,b}

\textsuperscript{a}Cryptology and Security Research Unit, Indian Statistical Institute, Kolkata, India
\textsuperscript{b}CSIRO Data61, Australia
\textsuperscript{c}Polygon (previously Matic Network)

Abstract

Lightning Network can execute unlimited number of off-chain payments, without incurring the cost of recording each of them in the blockchain. However, conditional payments in such networks is susceptible to Griefing Attack. In this attack, an adversary doesn’t resolve the payment with the intention of blocking channel capacity of the network. We propose an efficient countermeasure for the attack, known as Griefing-Penalty. If any party in the network mounts a griefing attack, it needs to pay a penalty proportional to the collateral cost of executing a payment. The penalty is used for compensating affected parties in the network. We propose a new payment protocol \textit{HTLC-GP} or \textit{Hashed Timelock Contract with Griefing-Penalty} to demonstrate the utility of the countermeasure. Upon comparing our protocol with existing payment protocol \textit{Hashed Timelock Contract}, we observe that the average revenue earned by the attacker decreases substantially for \textit{HTLC-GP} as compared to \textit{HTLC}. We also study the impact of path length for routing a transaction and rate of griefing-penalty on the budget invested by an adversary for mounting the attack. The budget needed for mounting griefing attack in \textit{HTLC-GP} is 12 times more than the budget needed by attacker in \textit{HTLC}, given that each payment instance being routed via path length of hop count 20.

Keywords: Lightning Network, Griefing Attack, Griefing-Penalty, Reverse-Griefing, Hashed Timelock Contract with Griefing-Penalty.

1. Introduction

Since the inception of Bitcoin\textsuperscript{2} in 2009, Blockchain technology has seen widespread adoption in the payment space. In spite of many desired features like decentralization and pseudonymity, a constant criticism faced by Bitcoin and Ethereum\textsuperscript{3} is that of scalability.

Layer-two protocols\textsuperscript{4} enables users to perform transactions \textit{off-chain}, massively cutting down data processing on the blockchain. \textit{Payment Channel}\textsuperscript{5}, \textsuperscript{6} stood out as a practically deployable solution. It is modular in nature, without requiring any fundamental changes in the protocol layer. Except for the opening and closing of the payment channel, several transactions can be executed off-chain without recording it in Blockchain. In case of dispute, any party can unilaterally broadcast the latest valid transaction in the blockchain and terminate the channel. A malicious party will lose all the funds locked in the channel if it tries to broadcast any older transaction. Since opening a payment channel has its overhead in terms of time and funds locked, parties that are not connected directly leverage on the set of existing payment channels for transfer of funds. The set of payment channels form the \textit{Payment Channel Network} or PCN\textsuperscript{6}. In
practice, Lightning Network for Bitcoin [6] and Raiden Network for Ethereum [7] are the widely deployed PCNs.

Figure 1: Bob mounts Griefing Attack

Payment in Lightning Network. If a sender and a recipient of payment do not share a channel, such payments make use of Hashed Timelock Contracts or HTLCs [6]. Since the payment gets routed via several intermediaries, a specific condition is imposed via off-chain contracts in order to prevent cheating. Payments are contingent to fulfillment of this condition. We describe with an example how conditional payments get executed between parties not directly connected by payment channel in Lightning Network. Suppose Alice wants to transfer $p$ coins to Bob via path comprising payment channels Alice-Dave, Dave-Charlie and Charlie-Bob, as shown in Fig. 1. Each intermediate node charge a processing fee of $p'$. Alice forwards a conditional payment to Dave, forming an off-chain contract, denoted as $\text{Contract}(p + 2p', t + 2\Delta)$, locking $p + 2p'$ coins for a time period $t + 2\Delta$. Here $\Delta$ is the worst-case confirmation time for settling a transaction on-chain. Dave deducts $p'$ coins from the amount and forwards the payment to Charlie by forming an off-chain contract, locking $p + p'$ coins for $t + \Delta$. Finally, Charlie deducts $p'$ coins from the payment amount and locks $p$ coins with Bob for a time period $t$. In order to claim $p$ coins from Charlie, Bob must resolve the payment within time period $t$. If the time period elapses, Charlie goes on-chain to claim refund, closes the channel Charlie-Bob and unlocks the money from the contract. Using the information released by Bob, rest of the intermediaries resolve the payment as well, each claiming a processing fee of $p'$.

Payment susceptible to Griefing Attack. Griefing Attack in Lightning Network was first mentioned in [8]. Paralyzing the network for multiple days by overloading each channel with maximum unresolved HTLCs has been studied in [9], [10]. In [11], sybil nodes initiate several payments via multiple paths and grieves them simultaneously.

In the example described above, Bob mounts griefing attack by not responding, as shown in Fig. 1. Charlie can go on-chain and withdraw the coins locked in the contract only after the elapse of the contract’s timeperiod. Thus Bob manages to lock $O(p)$ coins in each of the preceding payment channels for a timeperiod of $t$ units, without investing any money. Note that $t$ could be of the order of $24$ hours. Hence for an entire day, none of the parties can utilize the amount locked in their respective off-chain contracts.

Motive behind Griefing Attack. By mounting griefing attack, an adversary may try to achieve either of the objectives:

- Stalling network using self payment: The adversary controls the sender and receiver of several payment requests, blocking multiple intermediaries from accepting any other payments to be routed through it [11], [12]. In order to decrease the network throughput, an adversary may setup several Sybil nodes at strategic positions across the PCN and amplify the damage by submitting several payment requests.

- Eliminating a competitor from the network: The adversary tries to eliminate a competitor and block all its existing channel’s outgoing capacity [13], [12]. The adversary sets the victim as an intermediate node in the path carrying out the self-payment. The transaction value of self payment is equivalent to the victim’s outgoing channel capacity, jamming all the channels of the victim node. The victim cannot utilize the
fund until the adversary decides to claim the payment. As a consequence, several future payment request which could have been routed through the victim node now gets routed through the adversary. It reaps indirect economic benefit by claiming the processing fee for routing such transactions.

In Fig. 2, Node B has outgoing channel with A and C, each of capacity 0.1 BTC (each party having a balance of 0.05 BTC). Node D has channel with A and C, each of capacity 0.2 BTC. It conducts self-payment of 0.05 BTC, in each direction. Upon grieving for 24 hrs, D denies B from accepting any transaction request. A and C, having residual outgoing capacity of 0.1 BTC each in channel AD and CD, is now forced to route all the payments via D.

• Stalling network using intermediary: The adversary controls a node with a high degree centrality and broadcasts its processing fee to be extremely low in order to ensure multiple payments get routed through such nodes [14]. It later ignores all the payment by not forwarding the message to outgoing neighbours, locking funds across multiple paths thereby affecting a large portion of the network.

1.1. Our Goal

Griefing attack in Lightning Network cannot be prevented as long as a malicious node has nothing to lose or, in other words, it has nothing at stake. The problem cannot be solved until and unless the attacker has the fear of losing money upon mounting the attack. Thus, before accepting the payment parties must make a commitment to pay a compensation, in case they stop responding intentionally. The amount deducted from adversary’s balance must be able to compensate all the parties which got affected by the attack. A high level idea of the countermeasure has been pictorially depicted in Fig. 3. Alice forwards the payment to Bob via some intermediaries. Each party accepting the off-chain contract is supposed to lock an amount, which gets deducted if the party fails to resolve the payment before the contract timeout period. Here Bob doesn’t respond intentionally, allowing the timeout period of the contract to elapse. As per the terms of the contract, he gets penalized and the funds slashed from his account is used to compensate Alice, Dave and Charlie.

Figure 2: Eliminating a competitor

Figure 3: Bob is penalized
1.2. Our Contributions

In this paper, we have made the following contributions:

- We propose a countermeasure for mitigating griefing attack in Lightning Network, known as Griefing-Penalty. It punishes the griefer by forcing it to pay compensation to all the parties whose funds got locked for a certain time-period as a result of the attack.

- To illustrate the benefit of the proposed countermeasure, we propose a new payment protocol, called HTLC-GP or Hashed Timelock Contract with Griefing-Penalty. The penalty deducted is a fraction of the amount of funds locked by the attacker per unit time. This fraction is termed as rate of griefing penalty.

- We propose a construction for secure multihop payment using HTLC-GP and provide security analysis for the same. It proves that our protocol is privacy preserving and mitigates loss due to griefing attack by compensating the honest nodes.

- We study two attacking strategies for eliminating competitor node from network. Upon mounting the griefing attack following either of the strategy, we compare the profit made by the attacker in HTLC and HTLC-GP, by executing the protocols on several snapshots of Lightning Network. The profit here is termed as Return on Investment (RoI). It is observed that RoI is negative for HTLC-GP compared to a positive RoI in HTLC, hence disincentivizing a node from mounting griefing-attack due to substantial loss incurred.

- We compare the investment made by the adversary for mounting the attack in both HTLC-GP and HTLC upon varying path length and rate of griefing-penalty. The budget needed for mounting griefing attack in HTLC-GP is 4 times more than the budget needed for HTLC, when the path length is set to 4. The ratio increases to 12 for path length of 20, which is the maximum hop count allowed in Lightning Network. For a fixed path length, the ratio increases up to 500 for rate of griefing penalty exceeding 10^{-3}.

1.3. Organization of the Paper

We provide a high level overview of our proposed countermeasure, Griefing-Penalty, in Section 3. Based on this idea, we have proposed a new payment protocol, HTLC-GP or Hashed Timelock Contract with Griefing-Penalty in Section 5. We provide a detailed construction of Multi hop payment using HTLC-GP in Section 6. Security analysis of the proposed multihop payment protocol has been provided in Section 7. We divide the Performance Evaluation in Section 8 into two parts. Firstly, we analyze the profit earned by eliminating competitor in Section 8.1, demonstrating the efficiency of HTLC-GP over HTLC in countering griefing attack. Next, we discuss in Section 8.2 the impact of certain parameters on the investment made by attacker in HTLC-GP. Related Works has been stated in Section 9 and finally, we conclude the paper stating the scope for future work in Section 10. Notations used in the paper is given in Table 1.

2. Background

2.1. Payment Channel Network

A Payment Channel Network or PCN is defined as a bidirected graph $G := (V, E)$, where $V$ is the set of accounts dealing with cryptocurrency and $E$ is the set of payment channels opened between a pair of accounts. A PCN is defined with respect to a blockchain. Apart from the opening and closing of the payment channel, none of the transaction gets recorded on the blockchain. Upon closing the channel, cryptocurrency gets deposited into each user’s wallet according to the most recent balance in the payment channel. Every node $v \in V$ charges a processing fee $fee(v)$ for relaying funds across the network. Correctness of payment across each channel is enforced cryptographically by hash-based scripts [6] or scriptless locking [15]. Each payment channel $(v_i, v_j)$ has an associated capacity $locked(v_i, v_j)$, denoting the amount locked by $v_i$ and $locked(v_j, v_i)$ denoting the amount locked by $v_j$. $remain(v_i, v_j)$ signifies the residual amount of coins $v_i$ can transfer to $v_j$. Suppose sender $S$, which is node $v_0$, wants to transfer amount $\alpha$ to $R$, which is node $v_n$ through a path $v_0 \rightarrow v_1 \rightarrow v_2 \ldots \rightarrow v_n$, with each node $v_i$ charging a processing fee $fee(v_i)$. If
### Table 1: Notations used in the paper

| Notation | Description |
|----------|-------------|
| $G(V,E)$ | Graph representing the Lightning Network |
| $V$      | Set of nodes in Lightning Network |
| $E$      | Set of payment channels in Lightning Network, $E \subseteq V \times V$ |
| $U_0$    | Payer/Sender, $U_0 \in V$ |
| $U_n$    | Payee/Receiver, $U_n \in V$ |
| $\alpha$| Amount to be transferred from $U_0$ to $U_n$ |
| $P$      | Path connecting $U_0$ to $U_n$ |
| $n$      | Length of the path $P$ |
| $U_i \in V, i \in [0,n]$ | Nodes in $P, (U_i, U_{i+1}) \in E$ |
| locked($U_i, U_j$) | Amount of funds locked by $U_i$ in the payment channel $(U_i, U_j)$ |
| remain($U_i, U_j$) | Net balance of $U_i$ that can be transferred to $U_j$ via off-chain transaction |
| fee($U_i$) | Processing fee charged by $U_i$ for forwarding the payment |
| $\lambda$ | Security Parameter |
| $\mathcal{H}(0,1)^{\lambda} \rightarrow \{0,1\}^k$ | Standard Cryptographic Hash function |
| $\Delta$ | Worst-case confirmation time when a transaction is settled on-chain |
| $\gamma$ | Rate of gifting penalty (per minute) |

### 2.2. Hashed Time-lock Contract

Multihop Payment in Lightning Network is enabled by the use of Hashed Time-lock Contract (HTLC) [6]. A payer $S$ wants to transfer funds to a payee $R$, using a network of channels across an $n$-hop route $(v_0, v_1, v_2, \ldots, v_n), S = v_0, R = v_n$. The payee $R$ creates a condition $y$ defined by $y = \mathcal{H}(\tilde{x})$ where $\tilde{x}$ is a random string and $\mathcal{H}$ is a random oracle [17]. The condition $y$ is shared with $S$. The condition is shared across the whole payment path. Between any pair of adjacent nodes $(v_i, v_{i+1})$, the hashed time-lock contract is defined by $HTLC(v_i, v_{i+1}, y, b, t)$. It implies that $v_i$ locks $b$ units of fund in this contract. The amount locked can be claimed by party $v_{i+1}$ only if it releases the correct preimage $\tilde{x} : y = \mathcal{H}(\tilde{x})$ within timeout $t$. If $v_{i+1}$ doesn’t release $\tilde{x}$ within time $t$, then $v_i$ settles the dispute on-chain by broadcasting the transaction. The channel between $v_i$ and $v_{i+1}$ is closed and $v_i$ unlocks the money from the contract. If $v_{i+1}$ releases the preimage $x$ then it can either broadcast the transaction on-chain or settle the contract off-chain. Upon off-chain settlement, the contract is invalidated by creating a new commitment transaction, with $b$ units being added to $v_{i+1}$’s account. $HTLC$ acts like a conditional payment which is forwarded by each of the intermediate parties until it reaches the payee. If the payee or any other intermediate node ignores to resolve incoming contract request and waits for the expiration of the off-chain contract, the funds remain locked in all the channels starting connecting payer to the attacker. Note that after the timeout period, all the parties
withdraw the fund locked in the contract. The attacker manages to mount griefing attack without losing any money in the process.

3. **Key Idea of Griefing-Penalty**

Designing fair protocols on Bitcoin, where the adversary is forced to pay a mutually predefined monetary penalty to compensate for the loss of honest parties was first introduced by Bentov et al. [18]. Inspired by this idea, we propose a countermeasure for griefing attack, *Griefing-Penalty*, to solve the problem of griefing in the Lightning Network. The griefing-penalty imposed on an adversary for mounting griefing attack on a path of length, \( n \) is proportional to the summation of collateral cost of each payment channel involved in routing. *Collateral cost per payment channel* is defined as the product of the amount locked in the off-chain contract and the expiration time of the contract. The amount deducted per unit time from adversary’s balance is fraction of the collateral locked. This fraction is termed as *rate of griefing penalty* or \( \gamma \). The reason behind considering the expiration time of the contract for accounting griefing-penalty is to punish griefer for denying service to other participants in the path.

In the next section, we discuss how to incorporate griefing-penalty into the existing payment protocol, Hashed Timelock Contract or *HTLC*. Note that use of *Griefing-Penalty* is independent of the cryptographic primitive used for the underlying payment protocol. It can be incorporated in *Scriptless Scripts* as well [15].

4. **A Simple Protocol for countering Griefing Attack: HTLC1.0**

We incorporate *Griefing-Penalty* into HTLC [6]. Let us rename the modified payment protocol as *HTLC1.0*. It is assumed that for a given node, individual griefing-penalty earned upon elapse of locktime of the off-chain contract is less than the expected revenue earned by processing several transaction request within the given locktime. We assume that all the nodes in the network are rational, whose intention is to maximize the earning by remaining active in the network. Hence any such rational player would prefer to utilize their funds rather than earn penalty by reverse-griefing and keep their funds locked in a channel. Based on these assumptions, we define *HTLC1.0* with an example.

4.1. **Construction of two-party off-chain Revocable HTLC1.0**

Detailed construction of an instance of 2-party *HTLC1.0* is explained with an example shown in Fig. 2. Alice has established an off-chain contract with Bob for transferring 1 msat. Rate of griefing-penalty being 0.001 per minute. The terms of the contract is as follows: *Given \( H = H(x) \) in the contract, Bob can claim fund of 1 msat from Alice contingent to the knowledge of \( x \), within a time-period of 3 days. If Bob fails to do so, then after a timeout of 3 days, it pays a penalty of 4.32 msat to Alice.*

The establishment of two-party *HTLC1.0* between Alice and Bob has been illustrated in Fig. 4, the structure being similar to construction of *Off-Chain Revocable HTLC* [6]. Both parties have locked funds of 5 msat each, which gets included as the Funding Transaction. Bob locks 4.32 msat and Alice locks 1 msat into *HTLC1.0*. Bob can withdraw the entire amount contingent to the knowledge of preimage corresponding to the payment hash. If Bob fails to respond, upon expiration of locktime Alice claims the entire amount. Thus both the parties mutually agree to form second commitment transaction (CT1a/CT1b). Output 2 of CT1a describes how funds get locked in *HTLC1.0*. 5.32 msat will be encumbered in an *HTLC1.0*. If a party wants to unilaterally close the channel then it broadcasts latest Commitment Transaction. The parties are remunerated as per terms of the contract. If CT1a is broadcasted and Bob has produced R within 3 days, it can immediately claim the fund of 5.32 msat by broadcasting *HTLC1.0 Execution Delivery 1a*. Revocable Sequence Maturity Contract (RSMC) embedding [6] used in the output *HTLC1.0 Timeout Delivery 1a* ensures that if Alice broadcasts this transaction, it has to wait for 1000 block confirmation time before it can spend 5.32 msat. This extra waiting time serves as a buffer time for resolving dispute. If Alice had made a false claim of CT1a being the latest state of the channel, Bob will raise a dispute and spend Alice’s channel deposit.
The same state of channel is replicated in CT1b. However, the difference lies in how each party can spend their respective output with respect to the copy of the transaction they have. If CT1b is broadcasted and Bob has not been able to produce R within a period of 3 days, then it can claim fund of 5.32 msat after 3 days by broadcasting HTLC1.0 Timeout delivery 1b. If Bob has the preimage R, it can immediately broadcast HTLC1.0 Execution Delivery 1b. However this output is encumbered by 1000 block confirmation time, the explanation being the same as we had stated for HTLC1.0 Timeout Delivery 1a. These changes can be easily integrated into the Bitcoin script.
4.2. An instance of Multihop HTLC1.0

We describe the use of HTLC1.0 for multihop payment using an example. Alice wants to transfer $p$ coins to Bob. Bob shares the hash $H$ with Alice offline. This is used as the condition for the off-chain contracts established in the route forwarding the payment. Given the rate of griefing-penalty as $\gamma$ per unit of time, $0 \leq \gamma < 1$, and locktime of the contract being $(t + 2\Delta)$, Dave is expected to lock $\gamma(p + 2p')(t + 2\Delta)$ coins as griefing-penalty in the off-chain contract, $(p + 2p')(t + 2\Delta)$ being the collateral cost in channel Alice-Dave. If Dave provides the preimage of $H$ within this period, he will claim $p + 2p'$ coins and withdraw $\gamma(p + 2p')(t + 2\Delta)$ coins locked in the contract. Dave forwards a conditional payment of $p + p'$ coins to Charlie by forming similar off-chain contract using payment hash $H$ and locktime $(t + \Delta)$. Upon griefing, Charlie must pay a compensation of $\gamma(p + p')(t + \Delta)$. However, this amount is not sufficient to compensate both Dave and Alice. Hence he has to lock a cumulative griefing-penalty $\gamma(p + 2p')(t + 2\Delta) + \gamma(p + p')(t + \Delta)$ in the contract. This cumulative griefing-penalty is the summation of collateral cost in channel Alice-Dave and Dave-Charlie. Charlie forwards a conditional payment of $p$ coins to Bob by forming an off-chain contract for locktime of $t$ units. Bob has to lock $\gamma(p + 2p')(t + 2\Delta) + \gamma(p + p')(t + \Delta) + \gamma pt$ coins. This amount is the cumulative penalty to be distributed among Alice, Dave and Charlie, if Bob grieves. The entire payment protocol construction is shown in Fig. 5.

Suppose Bob grieves and refuses to release the preimage of $H$, waiting for time $t$ to elapse. He will pay a compensation of $\gamma(p + 2p')(t + 2\Delta) + \gamma(p + p')(t + \Delta) + \gamma pt$ coins to Charlie, as per the terms of the contract. After the timelock $t$ expires, Charlie goes on-chain. He closes the channel, unlocks $p$ coins and claims $\gamma(p + 2p')(t + 2\Delta) + \gamma(p + p')(t + \Delta) + \gamma pt$ coins as the compensation. He requests Dave to cancel the off-chain contract offering a compensation of $\gamma(p + 2p')(t + 2\Delta) + \gamma(p + p')(t + \Delta)$. Dave cancels the contract off-chain, unlocks $p + p'$ coins from the contract and claims the compensation from Charlie. If Charlie decides to grief, Dave can claim the compensation by going on-chain and closing the channel. Dave requests Alice to cancel the contract by offering a compensation of $\gamma(p + 2p')(t + 2\Delta)$. Thus except Bob, none of the parties lose funds in order to compensate any of the affected parties.

4.2.1. Problem of Reverse-Griefing in HTLC1.0

A major drawback of the protocol is that with the introduction of griefing-penalty, a malicious party can now ascribe the blame of griefing on an honest party as well. In the previous example, if Alice uses a wrong hash value or the HTLC timeout period is closer to the current block height of blockchain or the value of transaction forwarded is less than agreed value $19$, then Bob would cancel the payment. However, Charlie can deny settling the contract off-chain. Bob has no way to prove his innocence. Ultimately, with elapse of locktime, Charlie goes on-chain, claiming Bob’s money, as shown in Fig. 6. This attack is termed as Reverse Griefing.

Though there is problem with this construction, it is quite simple and requires minimal amount of changes to the existing HTLC protocol for the purpose of implementation in Lightning Network. We discuss why an honest rational party may not be easily motivated to mount reverse-griefing attack:
The attacker needs to wait for the entire locktime before it can collect penalty from the counterparty. In a path of length \( n \), locktime of the contract established by the adversary can range from \( \Delta \) units to \( n\Delta \) units, depending upon its position in the path.

The attacker needs to go on-chain with the HTLC1.0 to redeem the penalty and pay the mining fee. If the cumulative penalty earned by the attacker is less than the bitcoin transaction fee, reverse-griefing is a loss making strategy.

Since reverse-griefing affects a single honest node, the adversary’s intention of blocking funds across the network becomes more costly as it needs to coordinate and mount several such attacks. If it wants to avoid paying the griefing-penalty, it has to corrupt more than one nodes in the path and devise a strategy accordingly.

In case the intention was to block the counterparty’s funds, the attack fails to block any of the other edges that the counterparty has with other honest nodes.

In case the attacker has a high degree centrality or betweenness centrality, it is even more foolish to mount a reverse-griefing attack as it is probable that the expected processing fee for the duration of the locktime might be far greater than the cumulative penalty reward.

The rate of griefing-penalty can be lowered in order to disincentivize a party from reverse-griefing.

However, there exist several nodes in the network which earn very low processing fee during their entire channel lifetime. Either they charge very negligible amount of fee for large valued transaction or they remain inactive for most of their lifetime in the network. Such nodes have higher tendency to deviate as the profit earned by reverse-griefing is higher than the total anticipated processing fee. Acceptability of HTLC1.0 is hence subjected to such arbitrary behavior.

### 5. Our Proposed Protocol using Griefing Penalty

It is observed in HTLC1.0 that establishing a single contract with minimal changes to the script is not sufficient to protect a party from being cheated. To avoid the problem of reverse-griefing, we propose a new payment protocol for Lightning Network, termed as Hashed Timelock Contract with Griefing-Penalty or HTLC-GP.

In this protocol, locking of penalty and locking of the transaction amount must be executed in separate rounds. Instead of both parties locking their funds into a single contract, the payer locks fund in one contract, the Payment Contract, and the payee locks his penalty in a separate contract, the Cancellation Contract. The two contracts are bound together using two distinct hashes, termed as Payment Hash and Cancellation Hash. The payee can unlock the penalty deposited in Cancellation Contract either by providing
the preimage to the first hash, i.e. the payment hash, or by providing the preimage to the second hash, i.e. the cancellation hash. The problem with this arrangement is the order in which the contracts must be established. If the payment contract gets established first then the payee can still grief without establishing the cancellation contract. In this way it can avoid any payment of griefing penalty. Thus we put the payer at an advantage by asking the payee to lock penalty into the Cancellation Contract and forward it to the payer in the first round. After the payer receives the contract, it will lock the fund into the Payment Contract and forward it to the payee. Since the payee is in possession of preimages corresponding to both the hashes, even if the payer denies forming the payment contract, it cannot mount a reverse-griefing attack on payee. After a certain time, the payee will cancel the contract by releasing the cancellation hash.

5.0.1. Construction of 2 party HTLC-GP

Consider an example where Alice and Bob have locked funds of 5 msat each, which gets included as the Funding Transaction. Alice intends to transfer 1 msat to Bob. Rate of griefing-penalty being 0.001 per minute. The terms of the contract is as follows: Given \( H = \mathcal{H}(x) \) in the contract, Bob can claim fund of 1 msat from Alice contingent to the knowledge of \( x \), within a time-period of 3 days. If Bob fails to do so, then after a timeout of 3 days, it pays a penalty of 4.32 msat to Alice.

In the first round of Locking, Bob forwards the cancellation contract to Alice by locking 4.32 msat. Bob can withdraw the entire amount contingent to the knowledge of preimage corresponding to the cancellation hash \( Y \) or payment hash \( H \). If Bob fails to respond, upon expiration of locktime Alice claims the entire amount. Thus both the parties mutually agree to form the commitment transaction (CT1a/CT1b) as shown in Fig. 7. The state of the channel is: Alice has balance of 5 msat, Bob has balance of 0.68 msat, money locked in HTLC-GP is 4.32 msat. Output 2 of CT1a describes how funds get locked in HTLC-GP. 4.32 msat will be encumbered in an HTLC-GP. If a party wants to unilaterally close the channel then it broadcasts latest Commitment Transaction. The parties are remunerated as per terms of the contract. If CT1a is broadcasted and Bob has produced \( r \) or \( x \) within 3 days, it can immediately claim the fund of 4.32 msat by broadcasting HTLC-GP Execution Delivery 1a. Revocable Sequence Maturity Contract (RSMC) embedding [6] used in the output HTLC-GP Timeout Delivery 1a ensures that if Alice broadcasts this transaction after 3 days, it has to wait for 144 block relative confirmation time before it can spend 4.32 msat. This extra waiting time serves as a buffer time for resolving dispute. If Alice had made a false claim of CT1a being the latest state of the channel, Bob will raise a dispute and spend Alice’s channel deposit.

The same state of channel is replicated in CT1b. However, the difference lies in how each party can spend their respective output with respect to the copy of the transaction they have. If CT1b is broadcasted and Bob has not been able to produce either \( r \) or \( x \) within a period of 3 days, then it can claim fund of 4.32 msat after 3 days by broadcasting HTLC-GP Timeout delivery 1b. Bob’s transaction HTLC-GP Execution Delivery 1b is encumbered by 144 block relative confirmation time, the explanation being the same as we had stated for HTLC-GP Timeout Delivery 1a.

In the second round of Locking, Alice and Bob update the state of the channel and create new commitment transaction CT2a/CT2b, as shown in Fig. 8. The previous HTLC-GP based on locking of penalty, is carried forward as it is still unresolved. In this round, Alice forwards the payment contract to Bob by locking 1 msat. The state of the channel is: Alice has balance of 4 msat, Bob has balance of 0.68 msat, money locked in first HTLC-GP is 4.32 msat and money locked in second HTLC-GP is 1 msat. Bob can claim the money from both the contracts contingent to the knowledge of preimage corresponding to the payment hash \( H \). If Bob reveals the preimage corresponding to cancellation hash, Alice withdraws 1 msat and Bob withdraws 4.32 msat from the contract. If Bob doesn’t respond before the locktime expires, Alice claims the money locked in both the contracts.

**HTLC-GP Script**

The structure of the script is as per the convention used in [21]. For implementation of HTLC-GP in Lightning Network, we discuss how to design the output scripts for Cancellation Contract and Payment Contract.

**HTLC-GP Offered Cancellation Contract:** Bob offers this script to Alice. This output sends funds to either
the remote node after the HTLC-GP timeout or using the revocation key, or to an HTLC-GP-success transaction either with a successful payment preimage or cancellation preimage. The output is a P2WSH, with a witness script:

- Release the funds if the script is signed by the revocation key (revocationpubkey).
- If the above condition fails, then check if HTLC-GP public key of the party not publishing the commitment (remote public key), i.e. of Alice, was provided. Now check which of the condition holds true:
  - The publisher of the commitment, i.e. Bob, can publish the HTLC-GP-success by using the notif clause. It ignores the condition when the remote public key is not provided. HTLC-GP-success condition can be realized if either of the condition is satisfied:
    - Bob can use the preimage of cancellation hash. Release the funds if the preimage is released and signed by both Alice and Bob,
    - Bob can use the preimage of payment hash. Release the funds if the preimage is released and signed by both Alice and Bob.
Figure 8: Revocable HTLC-GP using two hashes: Second Round Locking
– If Bob didn’t react, Alice can publish the HTLC-GP-timeout transaction.

The Bitcoin script structure is shown in Fig. 9

**Figure 9: Script Structure: Offered Cancellation Contract**

```
OP_DUP OP_HASH160 ( RIPEMD160 ( SHA256 ( revocationpubkey ))) OP_EQUAL
OP_IF
  OP_CHECKSIG
OP_ELSE
  ( remote_htlcgppubkey ) OP_SWAP OP_SIZE 32 OP_EQUAL
  OP_NOTIFY
  OP_IF
    OP_HASH160 ( RIPEMD160 ( payment_hash ) ) OP_EQUALVERIFY
    2 OP_SWAP ( local_htlcgppubkey ) 2 OP_CHECKMULTISIG
  OP_ELSE
    OP_HASH160 ( RIPEMD160 ( cancellation_hash ) ) OP_EQUALVERIFY
    2 OP_SWAP ( local_htlcgppubkey ) 2 OP_CHECKMULTISIG
  OP_ENDIF
OP_ELSE
  OP_DROP ( cltv_expiry ) OP_CHECKLOCKTIMEVERIFY OP_DROP
  OP_CHECKSIG
OP_ENDIF
OP_ENDIF
```

**HTLC-GP Offered Payment Contract:** Alice offers this script to Bob. This output sends funds to either an HTLC-timeout transaction after the HTLC-timeout or to the remote node using either the payment preimage or cancellation image or the revocation key. The output is a P2WSH, with a witness script:

- Release the funds if the script is signed by the revocation key (`revocationpubkey`).
- If the above condition fails, then check if HTLC-GP public key of the party not publishing the commitment (remote public key), i.e. of Bob, was provided. Now check which of the condition holds true:
  - The publisher of the commitment, i.e. Alice, can publish the HTLC-GP-timeout by using the notif clause.
  - Else, Bob can publish HTLC-GP success if any of the condition holds true:
    * Bob can use the preimage of `cancellation hash`. Release the funds if the preimage is released and signed by both Alice and Bob,
    * Bob can use the preimage of `payment hash`. Release the funds if the preimage is released and signed by both Alice and Bob.

The Bitcoin script structure is shown in Fig. 10

In the next section, we provide an instantiation of multihop payment using *HTLC-GP*.

6. Multihop Payment using HTLC-GP

6.1. System Model

The topology of the Lightning Network is known by all the participants in the network since opening or closing of a payment channel is recorded on the blockchain. Pairs of honest users, sharing a payment
channel, communicate through secure and authenticated channels. An honest party willing to send funds to another party, will adhere to the protocol. It will not tamper with the terms and conditions of off-chain contract meant for each of the intermediate payment channels, involved in relaying the payment. The model of communication is considered to be synchronous, with all the parties following a global clock for settling payments off-chain. We assume that all the nodes in the network are rational, whose intention is to maximize the profit earned.

6.2. Objective

- **Guaranteed compensation for an honest node**: All the honest parties affected by the griefing attack will be remunerated by the griefer. Except the griefer, none of them must lose fund in order to pay compensation to any of the affected parties.

- **Payer and Payee’s Privacy**: None of the intermediate nodes involved in routing a payment must be able to identify its exact position in the path as well as figure out the identities of sender and receiver of payment.

6.3. Adversarial Model & Assumptions

An adversary introduces multiple Sybil nodes and places them strategically in the network in order to maximize the collateral damage. Such Sybil nodes may be involved in self-payment or transfer funds from one Sybil node to the other for mounting griefing attack. The Sybil nodes may also act as intermediate nodes in a path of payment. The adversary can perform the following arbitrary actions in order to keep funds locked in the network for a substantial amount of time:

- It withholds the solution without resolving the incoming off-chain payment request.

- It may refrain from forwarding the off-chain payment request to the next neighbour.

- It just refuses to sign any incoming contract request.
6.4. Our proposed Construction

Given an instance of Lightning Network, for secure transfers of funds from sender $U_0$ to the receiver, the former selects an optimal route for transferring funds to the payee, as per its routing strategy. Since the path length $n$, we index the receiver as $U_n$. Let the path be $P = (U_0, U_1, \ldots, U_n)$, via which payer $U_0$ will relay fund of value $\alpha$ to payee $U_n$, each $U_i$ is a node in the graph and $(U_i, U_{i+1}), i \in [0, n-1]$ denotes a payment channel. Each party $U_i, i \in [1, n-1]$ charge a service fee of $fee(U_i)$ for relaying the fund. Hence the total amount that $U_0$ needs to transfer is $\hat{\alpha} = \alpha + \Sigma_{i=1}^{n-1} fee(U_i)$. We denote each $\alpha_i = \hat{\alpha} - \Sigma_{j=1}^{n} fee(U_j), i \in [1, n-1], \alpha_0 = \hat{\alpha}$ and $\alpha_{n-1} = \alpha$. Each node $U_i$ samples pair of secret key and public key $(sk_i, pk_i)$, the public key of each node is used to encrypt the information of establishing contract with the neighbouring node.

Parameters used

- **Rate of Griefing-penalty** $\gamma$: It decides the amount to be deducted per unit time as compensation from the balance of a node responsible for griefing. It is set as a system parameter with $0 \leq \gamma \leq 1$, measured in terms of per minute.

- **Routing Attempt Cost** $\psi$: $U_0$ has to figure out the path by probing channels that will be able to route the transaction. This may require several attempts and hence adds an extra computational as well as resource overhead on $U_0$. Thus $U_0$ adds the cost of routing attempt to the compensation withdrawn from griefer. This is a variable quantity and the quantity is kept hidden from other nodes but generally, a sender sets the value $\psi t_0 > \alpha((k + 1)t_0 + \Sigma_{i=1}^{k}l\Delta), k \in \mathbb{N}$ and preferably $k > 3$. Here $k$ is the masking factor, $t_0$ is the time period of the contract established between $U_0$ and $U_1$ and $\Delta$ is the time taken for a transaction to settle on-chain.

**Computing Griefing-Penalty**

$U_0$ shares $\phi(n)$ with $U_n$, where $\phi$ is a function used for blinding the exact value of $n$, $\phi(n)\alpha t_{n-1} \approx ((\psi + \alpha_0)t_0 + \Sigma_{j=1}^{n-1} \alpha_j t_j)$, adding the extra cost for routing to the compensation it must claim from $U_1$. Similar to HTLC, $t_{n-1}$ is the least timeout period, assigned to the off-chain contract between $U_{n-1}$ and $U_n$ where $t_{n-1} > \Delta$. For rest of the off-chain contracts established between $U_i$ and $U_{i+1}, i \in [0, n-2]$, the timeperiod of the contract $t_i > t_{i+1} + \Delta$. Adding the routing attempt costs to the compensation disallows $U_1$ from inferring the identity of the sender of a particular payment. It cannot distinguish whether $\psi$ is routing attempt fee or the cumulative flow from any predecessors of $U_0$. Even other nodes must not be able to figure out their position in the path with the information of cumulative griefing-penalty. The proof has been discussed in Theorem 2, in Section 7.

The maximum compensation which can be earned by $U_i, i \in [1, n-1]$ is $\gamma\alpha t_i$, where $\alpha_i$ is the amount to be transferred to $U_{i+1}$, if the contract is resolved successfully within $t_i$. If $U_{i+1}$ is at fault, then it has to pay compensation to all the parties which got affected starting from $U_i$ till $U_0$. Hence compensation charged by each channel $(U_k, U_{k+1}), k \in [0, i], \alpha_i$, must be withdrawn from the faulty node $U_{i+1}$. The total griefing-penalty to be paid is $\gamma(\Sigma_{j=1}^{i} (\alpha_j t_j) + (\alpha_0 + \psi) t_0)$, so that each party $U_m, m \in [1, i]$, gets a compensation of $\gamma\alpha m t_m$ and $U_0$ withdraws a compensation of $\gamma(\psi + \hat{\alpha}) t_0$.

6.4.1. Protocol Description

Our protocol involves the following three phases:

**Pre-processing Phase**

- $U_n$ samples the preimages $x$ and $r$, $x \neq r$ and constructs the two hashes: $H = \mathcal{H}(x)$ and $Y = \mathcal{H}(r)$.
It involves two rounds:

- Establishing Cancellation Contract
- Establishing Payment Contract

The cumulative griefing-penalty for node $U_0$ is defined as $\text{tg}_{p_0} = \gamma(\psi + \tilde{\alpha})t_0$ and for any node $U_i, i \in [1, n-1]$ as $\text{tg}_{p_i} = \gamma_i(\Sigma_{j=1}^{i}(\alpha_j t_j)) + (\alpha_0 + \psi)t_0$.

$U_0$ sends $M_0 = E(\ldots E(E(\phi, Z_n, pk_n), Z_{n-1}, pk_{n-1}), Z_{n-2}, pk_{n-2}), \ldots, Z_1, pk_1)$ to $U_1$, where $Z_i = (H, Y, \alpha_i, t_i - 1, \text{tg}_{p_i}, U_{i+1}), i \in [1, n-1]$ and $Z_n = (H, Y, \alpha_{n-1}, t_{n-1}, \text{tg}_{p_{n-1}}, \text{null})$. Here $M_{i-1} = E(M_i, Z_i, pk_i)$ is the encryption of the message $M_i$ and $Z_i$ using public key $pk_i$, $M_n = \phi$.

$U_i$ decrypts $M_0$ to get $Z_1$ and $M_1 = E(\ldots E(E(\phi, Z_n, pk_n), Z_{n-1}, pk_{n-1}), Z_{n-2}, pk_{n-2}), \ldots, Z_2, pk_2)$ is forwarded to the next destination $U_2$. This continues till party $U_n$ gets $E(\phi, Z_n, pk_n)$.

Two-Round Locking Phase It involves two rounds: establishing Cancellation Contract and establishing Payment Contract.

Establishing Cancellation Contract: Since the flow of griefing-penalty is in the opposite direction of the actual payment, it is logical for $U_n$ to initiate this round.

- $U_n$ decrypts to get $Z_n$. It checks $\gamma\phi(n)t_{n-1} = \gamma_{tg_{p_{n-1}}}$ and $\alpha_{n-1} = \alpha_i$. If this holds true, it forms a contract with $U_{n-1}$, locking $tg_{p_{n-1}}$.
- For the rest of the parties, $U_i, i \in [1, n-1]$ first checks $tg_{p_i} = \gamma_i t_i = \gamma_{tg_{p_{i-1}}}$ and then forms the off-chain contract with $U_{i-1}$, locking $tg_{p_{i-1}}$.
- The terms of the contract is defined as follows: $U_{i+1}$ can withdraw the amount $tg_{p_i} = \gamma_i(\Sigma_j^{i}(\alpha_j t_j)) + (\alpha_0 + \psi)t_0$ from the contract provided it reveals either $x : H = \mathcal{H}(x)$ or $r : Y = \mathcal{H}(r)$ within a period of $t_i$ else $U_i$ claims this amount as griefing-penalty after the elapse of the locktime.

Bad Case: If $U_{i-1}$ denies signing the cancellation contract then $U_i$ will abort. However, $U_n$ locks a substantial amount as griefing-penalty, thus it will wait for a bounded amount of time and withdraw the money, thereby aborting from the process. We denote this time as $\delta : \delta \leq t_{n-1}$. If $U_{n-1}$ stops responding after establishment of the cancellation contract, $U_n$ releases on-chain the preimage $r$ corresponding to cancellation hash after waiting for $\delta$ units of time and unlock the penalty $tg_{p_{n-1}}$ from the contract. The preimage $r$ is now used by other parties $U_j, j \in [i+1, n]$ to cancel their respective off-chain contracts with $U_{j-1}$. So even if $U_i$ aborts, $U_{i+1}$ can go on-chain, close the channel and withdraw the amount locked in the contract.

The pseudocode of the first round of Locking Phase for $U_n$, any intermediate party $U_i, i \in [1, n-1]$ and payer $U_0$ is stated in Procedure[1] Procedure[2] and Procedure[3] respectively.

Establishing Payment Contract: $U_0$, upon receiving the cancellation contract, initiates the next round by establishing chain of contracts in the forward direction, till it reaches the payer $U_n$. This proceeds as normal HTLC.

- Each node $U_i, i \in [0, n-1]$ forwards the terms of off-chain contract to $U_{i+1}$, locking $\alpha_i$.
- The off-chain contract is defined as follows: $U_{i+1}$ can claim the amount $\alpha_i$ provided it reveals $x : H = \mathcal{H}(x)$ within a period of $t_i$. If not, then $U_i$ withdraws the amount either contingent to the knowledge of $r : Y = \mathcal{H}(r)$ or after the elapse of locktime.

Bad Case: If $U_{i+1}$ doesn’t sign the payment contract, $U_i$ aborts from the process. Similar to the first round of locking phase, if $U_{n-1}$ doesn’t form the payment contract within time $\delta$, $U_n$ releases the preimage $r$ and unlocks the penalty $tg_{p_{n-1}}$ from the contract. $U_i$ will not be able to reverse-grief $U_{i+1}$ by aborting since the latter can go on-chain and withdraw the amount locked in the contract.

The pseudocode of the second round of Locking Phase for $U_0$ and any intermediate party $U_i, i \in [1, n-1]$ is stated in Procedure[4] and Procedure[5] respectively.
**Procedure 1:** Establishing Cancellation Contract: First Round of Locking Phase for $U_n$

1. **Input:** $(Z_n, φ(n), γ, α)$
2. $U_n$ parses $Z_n$ and gets $H', Y', α', t', tgp_n^{-1}$.
3. If $t' \geq t_{now} + \Delta$ and $α' \geq α$ and $γ(φ(n)α)t' \approx tgp_n^{-1}$ and $H' = H$ and $Y' \approx Y$ and $\text{remain}(U_n, U_n^{-1}) \geq tgp_n^{-1}$ then:
   4. Send $\text{Cancel}\_\text{Contract}\_\text{Request}(H, Y, t', tgp_n^{-1}, γ)$ to $U_n^{-1}$
   5. If acknowledgement received from $U_n^{-1}$ then:
      6. $\text{establish}(\text{Cancel}\_\text{Contract}(H, Y, t', tgp_n^{-1})$ with $U_n^{-1}$
      7. Record $t_{\text{form}} = \text{current}\_\text{clock}\_\text{time}$
   8. End
9. Else:
10. End
11. End
12. Else:
13. End
14. End
15. End

**Procedure 2:** Establishing Cancellation Contract: First Round of Locking Phase for $U_i$, $i \in [1, n-1]$

1. **Input:** $(H', Y', t', tgp_i, γ)$
2. $U_i$ parses $Z_i$ and gets $H, Y, α_i, t_i^{-1}, tgp_i^{-1}$.
3. If $H' = H$ and $Y' = Y'$ and $t' + \Delta \leq t_i^{-1}$ and $tgp_i - γα_i t' \approx tgp_i^{-1}$ and $\text{remain}(U_i, U_i^{-1}) \geq α_i$ and $\text{remain}(U_i, U_i^{-1}) \geq tgp_i^{-1}$ and (current time not close to contract expiration time) then:
   4. Sends acknowledgement to $U_i+1$ and wait for the off-chain contract to be established
   5. Send $\text{Cancel}\_\text{Contract}\_\text{Request}(H, Y, t_i^{-1}, tgp_i^{-1}, γ)$ to $U_i^{-1}$
   6. If acknowledgement received from $U_i^{-1}$ then:
      7. $\text{establish}(\text{Cancel}\_\text{Contract}(H, Y, t_i^{-1}, tgp_i^{-1})$ with $U_i^{-1}$
   8. End
9. Else:
10. End
11. End
12. Else:
13. End
14. Else:
15. End
16. End
Procedure 3: Establishing Cancellation Contract: First Round of Locking Phase for $U_0$

1. **Input:** $(H', Y', t', tgp')$
2. **if** $t' \geq t_0$ and $tgp' \geq tgp_0$ and $H' \geq H$ and $Y' \geq Y$ and $\text{remain}(U_0, U_1) \geq \alpha_0$ **then**
   3. Sends acknowledgement to $U_1$
   4. Confirm formation of penalty contract with $U_1$
   5. Initiate the second round, establishment of payment contract
   6. **else**
   7. **end**

**Release Phase:** $U_n$ waits for $\delta$ units of time before initiating this round. If the payment contract received from $U_{n-1}$ is correct, it releases the preimage $x$ or payment witness and resolves the contract off-chain. If $U_{n-1}$ has not responded with incoming payment contract request or it has encountered any error (wrong payment or penalty value, invalid locktime) in the terms of incoming contract, it releases the cancellation preimage $r$. In case of dispute, it goes on-chain using either of the preimage for settling the contract. This is repeated for other parties $U_i, i \in [1, n-1]$, which upon obtaining the preimage claims payment from the counterparty or withdraws funds from the contract.

If $U_{i+1}$ grieves and refuses to release preimage to $U_i$, it has to pay the a griefing-penalty for affecting the nodes $U_k, 0 \leq k \leq i$, so that all the nodes obtain their due compensation. The pseudocode of the Release Phase for $U_n$ and any intermediate party $U_i, i \in [1, n-1]$ is stated in Procedure 6 and Procedure 7 respectively.

Procedure 4: Establishing Payment Contract: Second Round of Locking Phase for $U_0$

1. **Input:** $(H, Y, \alpha_0, t_0)$
2. **if** ($U_1$ has agreed to form the contract) and (current time not close to contract expiration time)
   3. **then**
   4. $\text{remain}(U_0, U_1) = \text{remain}(U_0, U_1) - \alpha_0$
   5. establish $\text{Payment\_Contract}(H, Y, t_0, \alpha_0)$ with $U_1$
   6. **else**
   7. **abort**
   8. **end**

Procedure 5: Establishing Payment Contract: Second Round of Locking Phase for $U_i, i \in [1, n-1]$

1. **Input:** $(H, Y, \alpha_i, t_i)$
2. **if** $t_{i-1} \geq t_i + \Delta$ and $\alpha_i \geq \alpha_{i-1} + \text{fee}(U_i)$ and ($U_{i+1}$ has agreed to form the contract) and (current time not close to contract expiration time) **then**
   3. $\text{remain}(U_i, U_{i+1}) = \text{remain}(U_i, U_{i+1}) - \alpha_i$
   4. establish $\text{Payment\_Contract}(H, Y, t_i, \alpha_i)$ with $U_{i+1}$
   5. **else**
   6. **abort**
   7. **end**
Procedure 6: Release Phase for $U_n$

1 **Input:** Message $M$, time bound $\delta$

2 **if** $M \overset{*}{=} \text{Payment\_Contract}(H, Y, \alpha', t')$ and current\_clock\_time $- t_n^{\text{form}} \leq \delta$ **then**

3  Parse $M$ and retrieve $(H, Y, \alpha', t')$

4  **if** $t' \geq t_{\text{now}} + \Delta$ and $\alpha' = \alpha$ **then**

5      $z = x$

6  **else**

7      $z = r$

8  **end**

9 **end**

10 **else**

11  $z = r$

12 **end**

13 Release $z$ to $U_{n-1}$

14 **if** current\_time $< t_{n-1}$ **then**

15  **if** $U_n$ and $U_{n-1}$ mutually agree to terminate Payment Contract and Cancellation Contract **then**

16       **if** $z = x$ **then**

17          remain($U_n, U_{n-1}$) = remain($U_n, U_{n-1}$) + $\alpha + tgp_{n-1}$

18       **end**

19  **else**

20       remain($U_n, U_{n-1}$) = remain($U_n, U_{n-1}$) + $tgp_{n-1}$

21       remain($U_{n-1}, U_n$) = remain($U_{n-1}, U_n$) + $\alpha$

22  **end**

23 **end**

24 **else**

25  $U_n$ goes on-chain for settlement by releasing preimage $z$.

26 **end**

27 **end**

28 **else**

29  $U_{n-1}$ goes on-chain for settlement, claims $(\alpha + tgp_{n-1})$.

30  $z = \text{null}$

31 **end**

32 Call Release\_Phase($U_{n-1}, z$)
Procedure 7: Release Phase for $U_i, i \in [1, n-1]$

1. **Input:** $z$
2. Release $z$ to $U_{i-1}$
3. If $z \neq \text{null}$ and current time < $t_{i-1}$ then
   4. If $U_i$ and $U_{i-1}$ mutually agree to terminate Payment Contract and Cancellation Contract then
      5. If $z = x$ then
         6. $\text{remain}(U_i, U_{i-1}) = \text{remain}(U_i, U_{i-1}) + \alpha_{i-1} + tgp_{i-1}$
      7. else
      8. $\text{remain}(U_i, U_{i-1}) = \text{remain}(U_i, U_{i-1}) + tgp_{i-1}$
      9. $\text{remain}(U_{i-1}, U_i) = \text{remain}(U_{i-1}, U_i) + \alpha_{i-1}$
     10. else
     11. end
     12. end
     13. else
     14. $U_i$ goes on-chain for settlement by releasing preimage $z$.
     15. end
     16. end
     17. else
     18. $U_{i-1}$ goes on-chain for settlement after elapse of locktime $t_{i-1}$, claims $(\alpha_{i-1} + tgp_{i-1})$.
     19. end
     20. Call Release Phase($U_{i-1}, z$)

Safeguarding against Reverse-Griefing. Any request for off-chain termination of contract by a party $U_i, i \in [1, n-1]$, without providing any of the preimage, will not be accepted by $U_{i-1}$ unless $U_i$ is compensating it for the loss of time. If the party $U_{i-1}$ mutually terminates the contract without the knowledge of any of the preimage before elapse of locktime, it is quite possible $U_{i-2}$ may refuse to cancel the contract and wait for the contract to expire. This might lead to the problem of reverse-griefing where $U_{i-1}$ loses funds. Hence to safeguard itself, a party will agree to terminate the contract off-chain either on receiving griefing-penalty or on receiving any one of the preimage.

7. Security Analysis

**Theorem 1. (Guaranteed compensation for an honest node).** Given a payment request $(U_0, U_n, \alpha_0)$ to be transferred via path $P = \langle U_0, U_1, \ldots, U_n \rangle$, if at least one party $U_k, k \in [1, n]$ mounts griefing attack then any honest party $U_j \in P, j \in [0, k-1]$ will earn compensation, without losing any funds in the process.

**Proof:**

We consider the worst-case in which we assume only a single node is honest in a path and rest of the nodes acts maliciously. We note that if fewer number of parties are corrupted, the honest nodes still interact with malicious neighbours and hence they get reduced to cases mentioned here. In particular, we analyze interactions between honest and dishonest parties in our system and ensure that honest parties do not get cheated and get their due.

- **Case 1 : $U_0$ is honest**

  In the Pre-processing Phase, the honest sender builds the onion packets containing terms of contract which are propagated through the nodes in the path. While the values in the packets are contingent to the values sent by R, honest S is in no position to verify it. At this point, S just follows the protocol.

  In the Two Round Locking Phase:
- Establishing the Cancellation Contract: Since $U_0$ is the last party to receive the contract, in case any party $U_i$, $i \in [1, n]$ griefs, then it will end up paying a cumulative penalty of $\gamma.(\sum_{j=1}^{n-1}(\alpha_j t_j) + (\alpha_0 + \psi)t_0)$, whereby $U_0$ earns a compensation of $\gamma(\alpha_0 + \psi)t_0$.

- Establishing the Payment Contract: $U_0$ may not be able to forward the contract to $U_1$ if there is discrepancy in the terms of the outgoing contract or if $U_1$ has stopped responding. Since $U_n$ is dishonest, it will not release the preimage $r$ for cancelling the contracts established in the first round. As per the terms of the contract, after the elapse of locktime $t_{n-1}$, it pays a cumulative griefing-penalty $\gamma.(\sum_{j=1}^{n-1}(\alpha_j t_j) + (\alpha_0 + \psi)t_0)$ to $U_{n-1}$. This money is used for compensating rest of the parties, starting from $U_0$ till $U_{n-1}$. Even if $U_1$ griefs, as per the terms of the contract it has to pay the required compensation $\gamma(\alpha_0 + \psi)t_0$ to $U_0$. Hence $U_0$ will not lose funds.

In the Release Phase, a similar argument can be given. Assuming that two round locking phase got executed successfully, if $U_1$ is not able to release the preimage corresponding to either cancellation hash or payment hash before the elapse of contract’s locktime, it will have to pay a penalty to $U_0$.

- **Case 2:** An intermediate node $U_i$, $i \in [1, n - 1]$ is honest

For the Pre-processing Phase, if $U_i$ does not receive the onion packet, the payment won’t get instantiated.

In the Two Round Locking Phase:

- Establishing the Cancellation Contract: $U_i$ may not be able to forward the contract to $U_{i-1}$ if the latter stops responding. Since $U_n$ is dishonest, it will not release the preimage $r$ for cancelling the contracts established in the first round. As per the terms of the contract, after the elapse of locktime $t_{n-1}$, it pays a cumulative griefing-penalty $\gamma.(\sum_{j=1}^{n-1}(\alpha_j t_j) + (\alpha_0 + \psi)t_0)$ to $U_{n-1}$. Even if $U_{i+1}$ griefs, as per the terms of the contract it has to pay the required compensation $\gamma((\alpha_0 + \psi)t_0 + \sum_{j=1}^{i}(\alpha_j t_j))$ to $U_i$. Since no contract has been established between $U_i$ and $U_{i-1}$, $U_i$ retains the entire compensation.

- Establishing the Payment Contract: $U_i$ may not be able to forward the contract to $U_{i+1}$ if $U_{i+1}$ stops responding. The same logic stated for cancellation contract holds true except now $U_i$ can retain $\gamma \alpha_i t_i$ as compensation and forward the rest of the amount to node $U_{i-1}$. Even if $U_{i-1}$ doesn’t respond and wait for the locktime $t_{i-1}$ to elapse, $U_i$ will not lose funds.

In the Release Phase, $U_i$ can be grieved in the following ways:

- $U_{i+1}$ withholds the preimage (either cancellation or payment) from $U_i$ and waits for the contract locktime to expire. In that case, $U_{i+1}$ has to pay compensation of $\gamma((\alpha_0 + \psi)t_0 + \sum_{j=1}^{i}(\alpha_j t_j))$ to $U_i$. Even if $U_{i-1}$ reverse-grieves, $U_i$ will be able to compensate without incurring any loss.

- **Case 3:** $U_n$ is honest

Receiver $U_n$ initiates the release of preimage. It will resolve the payment within a bounded amount of time either by releasing the preimage for payment hash or cancellation hash, as per the situation. $U_{n-1}$ cannot reverse-grief and force receiver to pay a griefing-penalty.

**Theorem 2.** (Payer and Payee’s Privacy). Given the information of griefing-penalty in the off-chain contract, an intermediate node cannot infer its exact position in the path for routing payment.

**Proof:** For routing payment of amount $\alpha$ from $U_0$ to $U_n$ via intermediaries $U_i$, $i \in [1, n - 1]$, several instances of off-chain contract is established across the payment channels. The amount locked by party $U_j$ and $U_{j+1}$ in their off-chain contract is $\alpha_j$ and $\gamma((\psi + \alpha_0)t_0 + \sum_{k=1}^{j}(\alpha_k t_k))$, $j \in [0, n-1]$, respectively. Let us assume that there exists an algorithm $\tau$ which reveals the exact position of any intermediate node $D: D \in \{U_1, U_2, \ldots, U_{n-1}\}$ in the path. This implies that given the information of cumulative griefing-penalty mentioned in the contract, it can distinguish between the penalty charged by channel $(U_j, U_{j+1}), j \in [1, n-1]$ and penalty charged by channel $(U_0, U_1)$, which is $\gamma(\psi + \alpha_0)t_0)$. However, the routing attempt cost $\psi$, was added by node $U_0$ as an extra compensation charged to cover up for routing attempt expense as well as hiding its identity from
its next neighbour. This is information is private and not known by any node except $U_0$. Additionally, the value of $\psi$ is set such that $\psi t_0 \geq \alpha ((k + 1)t_0 + \sum_{j=1}^{k} (\Delta), k \in \mathbb{N}$. Any number being selected from $\mathbb{N}$ being equiprobable, the probability of distinguishing becomes negligible.

Note: In practical application, there is a limit on the routing attempt fee which a sender can charge. The set from which $k$ is selected is significantly smaller compared to $\mathbb{N}$. But even under such circumstances, the best inference made by any intermediate node $U_j$ about its location is that it is located at position $(j + k)$ where $\psi t_0 \geq \alpha t_0 + \alpha t_0 \Delta + \alpha t_0 2\Delta + \ldots + \alpha t_0 k\Delta, k \text{ acts as the blinding factor.}$

8. Performance Evaluation

8.1. Analysis of Profit earned by eliminating a Competitor from the Network

The motivation of the griefer is to eliminate a competitor. It will try to exhaust all the channel capacity of the victim and force all transactions to be routed through itself, with the expectation to earn processing fee.

Return on Investment or RoI is the profit earned by the attacker with respect to the investment made in the network. Here investment means the liquidity utilized by the attacker for simulating an attack. In Lightning Network, the RoI of a node processing transaction request is calculated as follows:

$$\text{profit}_\text{processed} = N_{tx}(\text{base fee} + \text{fee rate} \times \text{tx.value})$$
$$\text{RoI} = \text{profit}_\text{processed} - \text{total griefing penalty}$$

(1)

$\text{profit}_\text{processed}$ is calculated based on [24], [25]. $N_{tx}$ is the total number of transactions processed by the node and $\text{tx.value}$ is the amount transferred from payer to payee. $\text{total griefing penalty}$ is the penalty required to pay as compensation to the affected parties upon mounting griefing-attack. For HTLC, the $\text{total griefing penalty}$ is 0 since there is no concept of penalizing the attacker. Hence the node always earns a non-negative RoI.

For HTLC-GP, if the node mounts a griefing attack, it has to pay a griefing-penalty proportional to the collateral locked for the given timeperiod. If the $\text{total griefing penalty}$ exceeds the $\text{profit}_\text{processed}$, the node incurs a loss. In the next section, we define two strategies which can be opted by the attacker. Based on these strategies, we compare the Return on Investment obtained for HTLC and HTLC-GP for a given budget.

8.1.1. Attacking Strategies
(a) Attacker establishes additional channels

Nodes with high betweenness centrality tend to act as intermediaries for routing payments. The attacker selects such nodes as its victim. We illustrate the situation by studying the structure of an instance of Lightning Network. The snapshot taken on 19th May, 2020, in Fig. [11](a). Nodes marked as Targeted Source and Targeted Sink routes their payment via node Victim. A malicious node establishes new channels with the Targeted Source and Targeted Sink. It selects the route Attacker $\rightarrow$ Targeted Source $\rightarrow$ Victim $\rightarrow$ Targeted Sink $\rightarrow$ Attacker, sends self-payment requests and mounts griefing attack. The path Targeted Source $\rightarrow$ Victim $\rightarrow$ Targeted Sink gets blocked. All the payments from Targeted Source gets routed through the path Targeted Source $\rightarrow$ Attacker $\rightarrow$ Targeted Victim.

(b) Attacker uses existing channels

In the previous strategy, the attacker had to establish channels before mounting the attack. To avoid the cost of establishing new channels, the attacker makes use of its existing payment channels to block payments received by its competitor. Illustrating the attack on the same instance of Lightning Network, as shown in Fig. [11](b), we consider that the node marked as Hub routes all the payment request via Victim node and ignores sending any payment via Attacker node. In order to steal payments being routed via Victim node, it selects the route Attacker Node $\rightarrow$ Hub $\rightarrow$ Victim Node $\rightarrow$ Neighbour... $\rightarrow$ Attacker Node for self-payment and mounts griefing attack. The path Hub $\rightarrow$ Victim Node $\rightarrow$ Neighbour... $\rightarrow$ Attacker Node gets blocked. Now Hub will be forced to route such payment request through Attacker Node.
8.1.2. Experimental Analysis

Setup: For our experiments, we use Python 3.8.2 and NetworkX, version 2.4 - a Python package for analyzing complex networks. System configuration used is Intel Core i5-8250U CPU, Operating System: Kubuntu 20.04, Memory: 7.7 GiB of RAM. The code for our implementation is available on GitHub. From the dataset mentioned in [9], we took twelve snapshots of Bitcoin Lightning Network over a year, starting from September, 2019. Each snapshot provides information regarding the public key of the nodes and the aliases used. The topology of the network is represented in the form of channels, represented as pair of public keys along with the channel capacity and the channel identifier. Each node of the channel follows a node policy which mentions about the base fee in millisatoshi, fee rate per million (in millisatoshi) and time_lock_delta. The capacity denotes the money deposited in the channel and not the balance of individual parties involved in opening of the channel. Thus each snapshot of Lightning Network undergo preprocessing where we filter out channels which is marked as disabled. Next, we select the largest connected component in the network. Since our proposed strategy for countering griefing attack requires both the parties to fund the channel, we divide the capacity of the channel into equal halves and allocate each half as the balance of a counterparty. The preprocessed graphs are used for evaluating both HTLC and HTLC-GP.

Designing transaction set: The best way to approximate the maximum value routed through a node is to map it into a flow problem and compute the maximum flow from multiple payers/sources to multiple payees/sinks. The amount of flow across each channel is the upper bound on the number of transactions being processed. If the attacker manages to block at least one path connecting a payer and payee, then the payer will route its transaction via the attacker. Since it is easier to analyze the situation in a hub-and-spoke network, we select a subgraph of LN having a similar structure. Nodes with high betweenness centrality have high probability of being a potential hub node. We select such a hub node where a subset of the pendant nodes connected to the hub forms the set of source and rest of the neighbours form the sink. Once a maximum flow is computed, the flow through a channel connecting a source to hub and through the channel.

https://github.com/subhramazumdar/GriefingPenaltyCode
from hub to a sink forms the maximum valued payment that can be routed through the path. The attacker targets such a source-sink pair with the hub acting as its victim. The attacker selects the victim, blocks its channels. Once the maximum flow across such source-sink pairs gets computed, the attacker checks the fraction of flow which gets routed through itself. To get an estimate of transaction set size, it divides the flow by the amount per transaction. *Return on Investment* can be calculated for all such transactions based on Eq. (1).

**Data Used:** We vary the range of transaction amount between 1 satoshi to 100000 satoshi [20], increasing the amount by multiple of 10. For the attack involving establishment of new channels by the attacker, we vary the level of budget of the adversary as 3000 satoshi, 30000 satoshi, 300000 satoshi, . . . , 3 BTC. For the attack involving use of existing channels by the attacker, we vary the level of budget of the adversary as 3000 satoshi, 30000 satoshi, 300000 satoshi, . . . , 0.03 BTC. Increasing budget beyond 0.03 BTC is of no use since substantial amount of budget remains unutilized after this point. Note that here budget allotted is utilized by the attacker for instantiating payment and locking cumulative griefing-penalty, ignoring the cost of establishing the channels in the network.

**Simulation Result**

The *Return on Investment (RoI)* is measured in log-scale. For negative RoI, we use log-modulus transformation [28].

- **Using attacking strategy 1:** The first result *RoI vs Average value per transaction*, for a fixed budget of 0.03 BTC and fixed rate of griefing-penalty of 0.001 per minute, shows that as the average value of each

![Graph](image-url)

(a) RoI vs Average value per transaction: Fixed Budget - 0.03 BTC
(b) RoI vs Budget: Fixed Average value per transaction - 10000 satoshi
(c) RoI vs Rate of Griefing-Penalty: Fixed Average value per transaction - 10000 satoshi

Figure 12: When Attacker uses new channels for mounting the attack
transaction increases, the processing fee earned by the attacker decreases due to decrease in the maximum number of payments processed for HTLC. However, for HTLC-GP, as the average value per transaction increases, RoI becomes negative as the processing fee earned becomes negligible compared to penalty incurred, as shown in Fig. 12(a).

The second result RoI vs Budget, for a fixed average value of transaction of 10000 satoshi and fixed rate of griefing-penalty of 0.001 per minute, shows that as the budget of the attacker increases, the processing fee earned by the attacker increases linearly. But a reverse trend is observed for HTLC-GP. RoI decreases linearly, as shown in Fig. 12(b). This is because is the amount of cumulative penalty is directly related to total collateral locked by the attacker.

The third result RoI vs Rate of Griefing-Penalty, for a fixed average value of transaction of 10000 satoshi and fixed budget of 0.03 BTC, the return on investment for HTLC remains constant since rate of griefing-penalty has no impact in this case. But the loss incurred increases with increase in γ, as observed for HTLC-GP in Fig. 12(c).

- **Using attacking strategy 2:** For a fixed budget of the attacker, loss incurred for HTLC-GP using second attacking strategy is higher than the first attacking strategy for all the three cases, as shown in Fig Fig. 13(a), Fig. 13(b) and Fig. 13(c). This is because average path length for self-payment being around 6.5 compared to the first attacking strategy, where the average path length remains fixed at 4.

8.2. Investment made by attacker for stalling the network

For a path of length $n$, the cumulative griefing-penalty is $\gamma((\psi + \alpha_0)t_0 + \sum_{j=1}^{n-1}\alpha_j t_j)$ for transferring an amount of $\alpha_{n-1}$ from sender $U_0$ to receiver $U_n$. In case of HTLC, for blocking liquidity of at least $\alpha_{n-1}$ in each of the $n$ channels, the attacker needs to invest $\alpha_0$ and execute a self-payment. In case of HTLC-GP,
in order to execute a self-payment of $\alpha_0$, the attacker needs to invest $\alpha_0 + \gamma((\psi + \alpha_0)t_0 + \sum_{j=1}^{n-1} \alpha_j t_j)$. If we take the ratio of the investment made for $HTLC$ and investment made by attacker for $HTLC-GP$ for a fixed transaction value,

$$\frac{\alpha_0}{\alpha_0 + \gamma((\psi + \alpha_0)t_0 + \sum_{j=1}^{n-1} \alpha_j t_j)} \leq \frac{1}{1 + \gamma(t_0 + \sum_{j=1}^{n-1} \frac{\alpha_j}{\alpha_0} t_j)} \tag{2}$$

- **Path Length**: Keeping $\gamma$ and $\alpha_n$ fixed, with increase in path length $n$, the ratio will be strictly less than 1 for any $n > 1$, since $\gamma(t_0 + \sum_{j=1}^{n-1} \frac{\alpha_j}{\alpha_0} t_j) > 0$.

- **Rate of Griefing-Penalty**: Keeping path length $n$ and $\alpha_n$ fixed, with increase in rate of griefing-penalty, the ratio will be strictly less than 1 for any value of $\gamma \in (0, 1)$ since $\gamma(t_0 + \sum_{j=1}^{n-1} \frac{\alpha_j}{\alpha_0} t_j) > 0$.

**Evaluation.** We use the same experimental setup and graph instances as in Section 8.1.2.

- **Impact of Path Length.** For a given transaction value and fixed rate of griefing-penalty set to 0.001 per minute, we vary the path length in the range from 4 to 20, and transaction value as 50000 satoshi, 70000 satoshi, 90000 satoshi, 110000 satoshi. The ratio of the adversary budget needed for mounting griefing attack in $HTLC-GP$ and the adversary budget needed for mounting griefing attack in $HTLC$ is around 4.7, when path length is 4 and around 12 when path length is 20. The ratio increases linearly with increase in path length, as observed in Fig. 14(a). Upon varying the transaction value, we do not observe any change in this trend.

- **Impact of Rate of Griefing-Penalty.** For a fixed transaction value of 50000 satoshi and given path length, we vary the rate of griefing-penalty $\gamma$ in the range $\{10^{-8}, 10^{-7}, 10^{-6}, \ldots, 0.01, 0.1\}$ and the path length as 5, 10, 15, 20. The ratio of the adversary budget needed for mounting griefing attack in $HTLC-GP$ and $HTLC$ increases exponentially with increase in rate of griefing-penalty. The rate of increase in ratio is almost equal till $\gamma = 10^{-4}$, invariant of change in path length. When $\gamma > 10^{-3}$, the rate of increase in the ratio is the lowest for path length of 5 and increases faster for path length of 20, as shown in Fig. 14(b).

The result shows that the investment made by the attacker for $HTLC-GP$ is higher than the investment made by the attacker for $HTLC$ thereby strongly disincentivizing griefing attack.

9. Related Works

Several ideas have been proposed for countering griefing attack. A limit on the number of incoming channel as well as the channel capacity was proposed in [13] as countermeasure for node isolation attack. However, the attacker may split the funds over multiple identities and channels to bypass the restrictions imposed. Game theoretic approach for analyzing the strategies of attacker and defender was proposed in [10]. Faster resolution of $HTLC$ has been stated in [9] as another method to avoid the disadvantage of having staggered locktime across payment channels. However, such a feature would violate the purpose of having $HTLC$ timeout which acts as a safety net against other possible malicious activities. All these payment protocols had a staggered locktime per each channel responsible for routing the payment. The collateral cost incurred for staggered locktime protocols is substantial. Sprites [29], an Ethereum styled payment network, first proposed the idea of using constant locktime for resolving payment. However, privacy was violated as the path information, identity of sender and receiver was known by all participants involved in routing the payment. A similar concept of reducing collateral cost using constant locktime contracts was proposed for Bitcoin-compatible payment networks in [13]. However, it violated relationship anonymity and the proposed protocol is yet to be realized practically.

Alternate mitigation strategies by incentivizing or punishing nodes have been stated in the past. Use of up-front payment was first proposed in [30]. In up-front payment, a party has to pay fee to the other party for accepting the $HTLC$. An excess fee paid is returned back to the sender upon successful resolution of payment. This introduces a lot of economic barrier where up-front payment may exceed the transaction fee. For small valued payment, a large up-front payment is a serious problem. Later, in [31], the concept of
reverse-bond was proposed which is similar to our proposed strategy. The counterparty accepting the HTLC will have to pay a hold-fee on a per unit interval basis, as if it has rented the HTLC. However, it has not been stated formally how this can be realized plus there is no way to track per unit interval in a decentralized asynchronous setup. Up-front payments has also been used for disincentivizing griefing attack in atomic swaps \cite{32}. In \cite{12}, a proposal of Proof-of-Closure of channels was proposed, where by each HTLC will have a hard timeout and a soft timeout period. However, a malicious node can setup several sybil nodes just for this purpose so that channel closure doesn’t affect its normal activity in the network. Table 2 provides a summary of the existing countermeasures and their disadvantages. Our proposed protocol addresses these shortcomings.

10. Conclusion & Future Work

In this paper, we have proposed a strategy for mitigating griefing attack in Lightning Network by imposing penalty on the adversary. This increases the total cost for launching such an attack as well as compensates other nodes in the network affected by griefing. We have shown how our proposed strategy works in a timelocked payments by proposing a new protocol HTLC-GP. The proposed construction not
only preserves privacy but also ensures that none of the honest intermediary present in the path gets affected due to imposition of penalty.

As part of our future work, we would like to extend the concept of grieving-penalty to Atomic Cross Chain Swap. A game-theoretic analysis for cross chain swaps using HTLC in [33] states the locking collateral by both the parties results in higher success rate of transaction. However this protocol assumes both the parties lock same amount of collateral in a single smart contract belonging to either of the blockchain. We would like to study the impact of exchange rate volatility, locktime of contract on the cumulative grieving-penalty, with each party locking collateral in different contracts belonging to different blockchains.

Acknowledgement

The authors are immensely grateful to Lightning Network Developers for their invaluable feedback.

References

[1] S. Mazumdar, P. Banerjee, S. Ruj, Time is money: Countering griefing attack in lightning network, in: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2020, pp. 1036–1043. doi:10.1109/TrustCom50675.2020.00138.
[2] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, Cryptography Mailing list at https://metzdowd.com.
[3] G. Wood, Ethereum: A secure decentralised generalised transaction ledger, Ethereum project yellow paper 151 (2014) 1–32.
[4] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, A. Gervais, Sok: Layer-two blockchain protocols, in: International Conference on Financial Cryptography and Data Security, Springer, 2020, pp. 201–226.
[5] C. Decker, R. Wattenhofer, A fast and scalable payment network with bitcoin duplex micropayment channels, in: Symposium on Self-Stabilizing Systems, Springer, 2015, pp. 3–18.
[6] J. Poon, T. Dryja, The bitcoin lightning network: Scalable off-chain instant payments (2016).
[7] Raiden network, http://raiden.network/ (July 2017).
[8] D. Robinson, HTLCs considered harmful, in: Stanford Blockchain Conference, 2019.
[9] A. Mizrahi, A. Zohar, Congestion attacks in payment channel networks, in: International Conference on Financial Cryptography and Data Security, 2021.
[10] S. Tochner, S. Schmid, A. Zohar, Hijacking routes in payment channel networks: A predictability tradeoff, arXiv preprint arXiv:1909.06890.

[11] Z. Lu, R. Han, J. Yu, Bank run payment channel networks, Cryptology ePrint Archive: Report 2020/456.

[12] Proof-of-closure as grieving attack mitigation, https://lists.linuxfoundation.org/pipermail/lightning-dev/2020-April/002608.html (April 2020).

[13] C. Egger, P. Moreno-Sanchez, M. Maffei, Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks, in: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 801–815.

[14] E. Rohrer, J. Mallaris, F. Tschorsch, Discharged payment channels: Quantifying the lightning network’s resilience to topology-based attacks, in: 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), IEEE, 2019, pp. 347–356.

[15] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, M. Maffei, Anonymous multi-hop locks for blockchain scalability and interoperability., in: NDSS, 2019.

[16] BtcDrak, M. Friedenbach, E. Lombrozo, Bip 112, checksequenceverify, https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki (2015-08-10).

[17] F. Beres, I. A. Seres, A. A. Benczúr, A cryptoeconomic traffic analysis of bitcoins lightning network, arXiv preprint arXiv:1911.09432.

[18] Bolt 3: Bitcoin transaction and script formats, https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md#offered-htlc-outputs.

[19] A. Riard, G. Naumenko, Time-dilation attacks on the lightning network, arXiv preprint arXiv:2006.01418.

[20] The lightning network (part 2) - routing fee economics, https://blog.bitmex.com/the-lightning-network-part-2-routing-fee-economics/ accessed: 2019-03-27 (2019).

[21] A proposal for up-front payments: Reverse bond payment, https://lists.linuxfoundation.org/pipermail/lightning-dev/2019-November/002282.html (November 2019).

[22] A proposal for up-front payments: Reverse bond payment, https://lists.linuxfoundation.org/pipermail/lightning-dev/2020-February/002547.html (February 2020).

[23] E. Heilman, S. Lipmann, S. Goldberg, The arwen trading protocols, in: International Conference on Financial Cryptography and Data Security, Springer, 2020, pp. 156–173.

[24] J. Xu, D. Ackerer, A. Dubovitskaya, A game-theoretic analysis of cross-chain atomic swaps with htlcs, arXiv preprint arXiv:2011.11325.