

1 Presentation and publication: General Congestion 2 Attack on HTLC-Based Payment Channel 3 Networks

4 **Zhichun Lu** ✉

5 Cryptape, China

6 **Runchao Han** ✉

7 Monash University & CSIRO-Data61, Australia

8 **Jiangshan Yu** ✉

9 Monash University, Australia

10 — Abstract —

11 Payment Channel Networks (PCNs) have been a promising approach to scale blockchains. However,
12 PCNs have limited liquidity: large-amount or multi-hop payments may fail. The major threat of
13 PCNs liquidity is *payment griefing*, where the adversary who acts as the payee keeps withholding the
14 payment, so that coins involved in the payment cannot be used for routing other payments before
15 the payment expires. Payment griefing gives adversaries a chance to launch the *congestion attack*,
16 where the adversary grieves a large number of payments and paralyses the entire PCN. Understanding
17 congestion attacks, including their strategies and impact, is crucial for designing PCNs with better
18 liquidity guarantees. However, existing research has only focused on the specific attacking strategies
19 and specific aspects of their impact on PCNs.

20 We fill this gap by studying the *general congestion attack*. Compared to existing attack strategies,
21 in our framework each step serves an orthogonal purpose and is customisable, allowing the adversary
22 to focus on different aspects of the liquidity. To evaluate the attack's impact, we propose a generic
23 method of quantifying PCNs' liquidity and effectiveness of the congestion attacks. We evaluate our
24 general congestion attacks on Bitcoin's Lightning Network, and show that with direct channels to
25 1.5% richest nodes, and ~ 0.0096 BTC of cost, the adversary can launch a congestion attack that
26 locks 47% (~ 280 BTC) coins in the network; reduces success rate of payments by 16.0%~60.0%;
27 increases fee of payments by 4.5%~16.0%; increases average attempts of payments by 42.0%~115.3%;
28 and increase the number of bankruptcy nodes (i.e., nodes with insufficient balance for making
29 normal-size payments) by 26.6%~109.4%, where the amounts of payments range from 0.001 to 0.019
30 BTC.

31 **2012 ACM Subject Classification** Security and privacy → Distributed systems security

32 **Keywords and phrases** Blockchain, PCN, Congestion

33 **Digital Object Identifier** 10.4230/OASICS...

34 **1** Introduction

35 Public blockchains suffer from limited throughput. Payment Channel Network (PCN) –
36 introduced by the Lightning Network (LN) [16] – is one of the promising ways to scale
37 blockchains. Payment channels enable *off-chain* payments, i.e. payments that do not need to
38 be recorded on the blockchain. To open a payment channel, two nodes collateralise some coins
39 in a joint address. They can make a payment by signing a new transaction that updates their
40 balances. To close the channel, one of the two nodes commits the transaction recording the
41 latest balance allocation to the blockchain. If two nodes do not have a direct channel, they
42 can make payments to each other using *multi-hop payments*, i.e., payments going through
43 one or more intermediate channels. In a multi-hop payment, the payer has to find a path
44 that directs him to the payee. The payment is made by updating balances of these channels



© Zhichun Lu, Runchao Han and Jiangshan Yu;
licensed under Creative Commons License CC-BY 4.0
OpenAccess Series in Informatics

OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

45 in an atomic way. The atomic update can be achieved by Hash Time Locked Contracts
46 (HTLCs): the payment in each hop is locked by a hash value chosen by the payee, and all
47 payments proceed if the payee reveal a hash value's preimage before a timeout to redeem the
48 payment from the payer, otherwise the payment will expire. In a HTLC-based multi-hop
49 payment, the payee chooses a preimage, and nodes make HTLC payments on all involved
50 channels with this preimage's hash value. Revealing this preimage activates these HTLC
51 payments simultaneously.

52 **Payment griefing.** A well-known attack on HTLC-based multi-hop payments is *payment*
53 *griefing* [18], where the adversary makes a payment and withholds the preimage, so that
54 coins involved in this payment are locked and cannot be used in other payments before the
55 griefing payment expires. Thus, payment griefing can reduce the PCN's liquidity, i.e. the
56 ability of routing payments. In addition, payment griefing is free, as the payer does not need
57 to pay anything for failed payments. Moreover, payment griefing is also unaccountable, as 1)
58 the victim cannot distinguish between a normal failed payment and a griefing payment, and
59 2) the intermediate nodes can not know the payer and payee's identity.

60 **Congestion attacks.** Griefing opens an important attack vector on HTLC-based PCN's
61 liquidity, namely the *congestion attack* [1,13]. In a congestion attack, the adversary initiates
62 a large number of concurrent payments and grieves them. Consequently, some channels hit the
63 limit of *max_concurrent_htlcs*, i.e., the number of concurrent unsettled payments allowed in
64 the channel, and therefore cannot route payments before the adversary's payments expire.
65 By launching a large-scale congestion attack, the entire PCN can be paralysed, i.e., the PCN
66 cannot route further payments.

67 **PCNs' liquidity: what is the real limit?** Understanding congestion attacks is important
68 for understanding PCNs' liquidity and therefore future PCN design. However, congestion
69 attacks are still a new concept and haven't been well-studied yet. While existing research [1,13]
70 only considers *max_concurrent_htlcs* as an exhaustible resource, it's unclear whether there
71 exists other resources that can be exhausted to create congestion. In addition, existing
72 congestion attacks apply a rather straightforward attack strategy, which will be analysed in
73 detail in §7. Moreover, we also observe that liquidity – the congestion attack's target – is not
74 well-defined yet. Besides the amount of locked balance and the number of locked channels
75 mentioned in Mizrahi et al. [13], some other metrics such as success rate of payments, fee
76 of payments, and number of attempts for making a payment have direct indications on the
77 PCN's liquidity. Congestion attacks over these metrics are not explored before.

78 **This work: general congestion attacks.** In this paper, we fill this gap by introducing
79 *general congestion attack*, which generalises the existing congestion attack in terms of attack
80 strategies and targeted metrics. We introduce a framework for launching congestion attacks,
81 where the adversary generates Sybil nodes connecting to a carefully chosen set of nodes,
82 establishes channels with them, initiates numerous multi-hop payments between its nodes,
83 and grieves these payments simultaneously. Compared to existing studies that put less effort
84 on the order of payments to be grieved [13,21], we provide five strategies for ranking these
85 payments, and each strategy focuses on some specific aspects of liquidity. To quantify the
86 effectiveness of congestion attacks, we introduce a generic method of quantifying PCNs'
87 liquidity. We evaluate the congestion attack on Bitcoin's LN – the first and most well-known
88 PCN. Our results show that congestion attacks can significantly damage the liquidity of
89 PCNs. In particular, with direct channels to 1.5% richest nodes, the adversary can launch a
90 congestion attack that locks 47% (~280 BTC) coins in the network; reduces success rate
91 of payments by 16.0%~60.0%; increases fee of payments by 4.5%~16.0%; increases average
92 attempts of payments by 42.0%~115.3%; and increase the number of bankruptcy nodes by



93 26.6%~109.4%, where the amounts of payments range from 0.001 to 0.019 BTC.

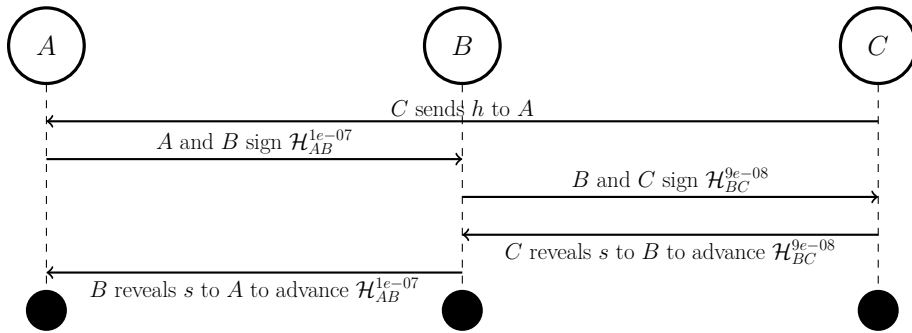
94 While being effective, our general congestion attacks are cheap to launch. The only cost
95 of general congestion attacks is the fee for establishing channels. Our evaluation shows that,
96 a successful attack on LN requires channel fee of approximately 0.0096 BTC. The adversary
97 does not lose its custody (i.e., coins in the channel) during the attack, as payments for
98 grieving will expire.

99 **Roadmap.** Section 2 provides the background of PCNs and grieving. Section 3 describes
100 the security model and the congestion attack. Section 4 describes the method of quantifying
101 PCNs' liquidity. Section 5 evaluates congestion attacks on LN. Section 6 discusses the
102 cost of congestion and strategy to utilise it for making a profit. Section 7 reviews relevant
103 literature, and provides a quantitative comparison between the general congestion attack and
104 the existing ones. Section 8 concludes this paper. Appendix A outlines detailed evaluation
105 results.

106 2 Background

107 2.1 Payment Channel Networks

108 Lightning Network (LN) [16] introduces the idea of Payment Channel Networks. A payment
109 channel allows two parties to pay each other without the need to publish every payment to
110 the blockchain. Instead, they collateralise their coins into a 2-of-2 multi-signature address.
111 They can make payments by mutually signing new transactions with updated balances. They
112 can make payments with each other by mutually signing new transactions with updated
113 amounts of their collateralised coins. To close the channel, one party commits the latest
114 state of channel balance to the blockchain, and coins in the channel will be allocated to both
115 parties accordingly.



116 ■ **Figure 1** A multi-hop payment from A to C via an intermediate node B .

117 The system can be further extended to support multi-hop payments. Most multi-hop
118 payment protocols are based on Hash Time Locked Contracts (HTLCs). HTLC is a contract
119 between two parties which guarantees that a payment will be made if the payee shows the
120 preimage of a hash value before a negotiated block height on the blockchain. If the payee
121 does not show the preimage and the timeout expires, the payment is deemed invalid.

122 Figure 1 describes a multi-hop payment where A pays $9e-08$ BTC to C via an intermediate
123 node B in Bitcoin's LN. First, C chooses a random string s as preimage and sends its hash
124 value $h = H(s)$ to A , where $H(\cdot)$ is a cryptographic hash function. A then signs a HTLC
125 contract \mathcal{H}_{AB}^{1e-07} with B stating “ A will pay $1e-07$ BTC to B if B can show the value of s
within (e.g.) 144 blocks”. B also signs a HTLC contract \mathcal{H}_{BC}^{9e-08} with C saying that “ B will



126 pay $9e-08$ BTC to C if C can show the value of s within (e.g.) 138 blocks”. Then C shows
 127 s to B to redeem $9e-08$ BTC in \mathcal{H}_{BC}^{9e-08} from B . Meanwhile, B can redeem $1e-07$ BTC in
 128 \mathcal{H}_{AB}^{1e-07} from A by revealing s to A . B is incentivised to reveal s , as B does not want to lose
 129 money. The timelock of AB is set to be longer than BC , so B always has sufficient time to
 130 reveal s to A .

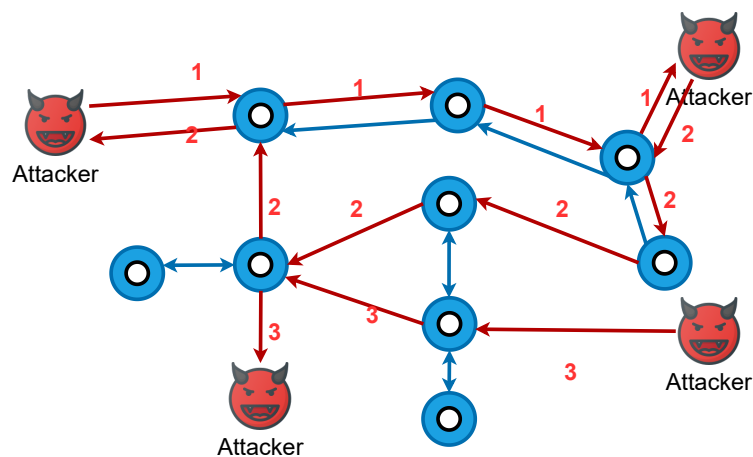
131 By routing this payment, B gets $1e-08$ BTC from A . This is known as “fee”, which is
 132 paid by the payer and is used for encouraging nodes to route multi-hop payments. In LN, fee
 133 consists of a fixed base fee and proportional fee that fluctuates according to the congestion
 134 level of the network. To minimise the cost, payers usually search for a path with the least
 135 fee when making payments.

136 2.2 Payment Griefing

137 If the payee C reveals the preimage on time and the intermediate node B is rational, the
 138 multi-hop payment will happen. However, as we mentioned before, there exists an attack
 139 called *payment griefing* [4], where the payee withholds the preimage until HTLCs expire.
 140 Before HTLC expires, coins involved in all channels of this payment are locked and cannot
 141 route other payments.

142 Payment griefing is a threat to PCNs’ liquidity. If a big portion of coins in a PCN are
 143 locked, the PCN will no longer be able to route payments. Payment griefing is cheap, as
 144 the payment does not really happen and the payer does not pay for the fee to intermediate
 145 nodes. Identifying payment griefing can be hard, as nodes cannot distinguish whether the
 146 withholding is due to network delay, on purpose, or by accident. If the PCN’s routing protocol
 147 is privacy-preserving, payment griefing can even be launched anonymously. For example,
 148 Bitcoin’s LN adopts Sphinx [7], where each intermediate node only has the knowledge of
 149 nodes who directly connect with him.

150 2.3 Congestion attack



■ **Figure 2** Congestion attack.

151 In a congestion attack, the adversary establishes payment channels with existing nodes
 152 in the PCN, and make numerous multi-hop payments between its nodes simultaneously.
 153 Then, the adversary withholds preimages until these payments expire. Before that, coins
 154 locked in these payments cannot be used in other payments. If the adversary has sufficient



155 custody, it can lock a great portion of coins in the PCN so that the PCN may be paralysed.
156 Figure 2 shows the intuition of the congestion attack, where the adversary generates Sybil
157 nodes connecting to a carefully chosen set of nodes, establishes channels with them, initiates
158 numerous multi-hop payments between its nodes, and grieves these payments simultaneously.

159 To reduce the custody required for an attack, Mizrahi et al. [13] proposed to lock the chan-
160 nel by initiating numerous payments with small amounts to occupy all *max_concurrent_htlcs*,
161 the maximum number of concurrent payments in a channel. In addition, they propose three
162 strategies for enumerating payment paths. Tikhomirov et al. [21] provided another strategy,
163 where the adversary attacks a single channel rather than a path for each step.

164 We compare the strategies of our attack with those of Mizrahi et al. [13] and Tikhomirov
165 et al. [21] in §3.2, and compare the cost and effectiveness in §7. The result show that. If
166 the attacker is fee-sensitive, then our attack is preferred because our fees are 16% of and 5%
167 of other two. Whereas, if the attacker has a restricted custody in hand, then the attack by
168 person Mizrahi et al. is more preferred, as the custody required is only 1.5% of our attack
169 (in case of locking 41% network’s capacity).

170 **3 General congestion attack**

171 **3.1 Model**

172 We consider a HTLC-based PCN that is identical to Bitcoin’s Lightning Network as described
173 in §2.1. We model the PCN as a weighted directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. \mathcal{V} is the set of nodes
174 in the network, and all v_* in the remainder of the paper refer to a specific element (i.e. a
175 node) in \mathcal{V} . \mathcal{E} is a set of tuples $(v_s, v_t, capacity)$, which represents a channel with a capacity
176 of *capacity* from v_s to v_t . Since channels in LN is bi-directional, a channel is represented in
177 the graph as two opposite edges $(v_s, v_t, capacity)$ and $(v_t, v_s, capacity)$. A payment P is a
178 dictionary $\{amount : \alpha, path : path\}$, where α is the amount of coins that the payer wants
179 to pay to the payee and *path* is the path of payment (a list of edges). Then, \mathcal{P} denotes list
180 of payments enumerated by the attacker. Meanwhile, we use l to refer the length of *path*.
181 For simplicity, we do not consider the impact of timelocks on our attack. The payer also
182 needs to pay some fee f_i for each intermediate node i on the path. We can now get the size
183 of a multi-hop payment θ as

$$\theta = \alpha \cdot l + \sum_{i=2}^l f_i$$

184 In addiciton, we use Γ_v to refer the total capacity of all channels connecting to v .
185 Therefore, the richest node we defined earlier is the node with the largest Γ_v .

186 We consider nodes in the PCN are rational. Each honest node in a multi-hop payment
187 will reveal the preimage of the hashlock to the upstream node once the node knows it. We
188 assume a malicious adversary, who has sufficient coins and aims at paralysing the entire
189 PCN with minimal cost. The adversary does not control any node in the beginning, but
190 has the knowledge of the network topology and each channel’s capacity and fee policy. The
191 information can be retrieved from PCN’s P2P protocol, evidenced by existing studies [5, 10].
192 When the adversary establishes a channel with a node, the node is willing to provide sufficient
193 capacity. According to liquidity providers such as Bitrefill [3], purchasing capacity from
194 nodes is easy and costs negligible coins. Moreover, if an adversary just wants to attack PCN
195 for a period of time, it can use the channel lease marketplace like lightning pool [14] to get
196 incoming liquidity at a much lower cost.



197 3.2 Attack framework

198 We generalise the congestion attack in terms of attack strategies and liquidity metrics. We
199 propose a framework for launching congestion attacks. The framework consists of four steps
200 as follows.

- 201 1. **Node selection:** the adversary chooses a set of nodes and establishes channels with
202 them.
- 203 2. **Payment enumeration:** the adversary enumerates all payments between its nodes.
- 204 3. **Path ranking:** the adversary orders these payments.
- 205 4. **Launching attack:** the adversary starts to make and abort payments in this order.

206 **Comparison with existing attack strategies.** While existing congestion attacks [13,21]
207 start by enumerating griefing paths or griefing channels, our attack chooses nodes in the
208 beginning and enumerates paths within the given set of nodes. Such design allows us to
209 divide the attack into multiple steps with orthogonal purposes, revealing the complete design
210 space for congestion attacks. Specifically, the adversary decides resources (e.g., channel
211 capacity or the `max_concurrent_htlc` parameter) to congest when enumerating payments,
212 and decides its focus of liquidity metrics when ranking payments.

213 In addition, in existing congestion attacks [13, 21], the adversary has to create new
214 channels when attacking a new channel or path. Therefore, the adversary has to establish
215 a large number of channels, which incurs additional transaction fees on the underlying
216 blockchain. In contrast, our attack chooses nodes in the beginning and enumerates paths
217 within the given set of nodes, and therefore requires much fewer channels.

218 3.3 Node selection

219 The adversary's first step is to join the PCN by establishing channels with existing nodes. We
220 analyse the adversary's strategy on the type of nodes and the number of nodes to establish
221 channels with.

222 **Type of nodes to establish channels with.** We suggest establishing channels with the
223 richest (w.r.t. total capacity of involved channels) nodes (which we call *hubs*) in the network,
224 as they are likely to route more griefing payments than a normal node. If establishing
225 channels with nodes with little capacity, the adversary has to establish channels with more
226 nodes, leading to more fee on establishing channels.

227 **Number of nodes to establish channels with.** The number of nodes to establish
228 channels with depends on how the adversary enumerates griefing payments (i.e., the second
229 step). By establishing channels with sufficient nodes, the total size of enumerated payments
230 will take the majority of the network capacity, and therefore the congestion attack will take
231 effect. Later in §5.1, we will show that for Bitcoin's Lightning Network, by establishing
232 channels with the top 1.5% richest nodes the enumerated payments can occupy 81% of the
233 network capacity ideally (47% in the experiment).

234 3.4 Enumerating payments

235 After establishing payment channels, the adversary enumerates all possible payments between
236 its nodes. To this end, the adversary has to find all paths between each pair of its nodes,
237 and calculate the maximum amount that each path can afford. Our payment enumerating
238 algorithm builds upon the Ford–Fulkerson algorithm [9] - a maximum flow algorithm in
239 graph theory. Maximum flow is a classic problem in graph theory, which aims at finding the



■ **Algorithm 1** Enumerating payments.

Input:

- 1: The entire network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- 2: The adversary's node list \mathcal{N}

Output:

- 3: The list of payments \mathcal{P}
-

```
4:  $\mathcal{P} \leftarrow []$ 
5: for  $(v_1, v_2)$  in  $\mathcal{N}.combination()$  do
6:    $path\_list \leftarrow \text{BFS}(\mathcal{G}, v_1, v_2)$ 
7:   for  $path \in path\_list$  do
8:      $P \leftarrow \{path : [], amount : 0\}$ 
9:      $P.path \leftarrow path$ 
10:     $capacity\_list \leftarrow [edge.capacity \text{ for edge in path}]$ 
11:     $\alpha \leftarrow \min(capacity\_list)$  ▷ Max viable amount
12:    if  $\alpha = 0$  then continue ▷ This path is not viable
13:     $P.amount \leftarrow \alpha$  ▷  $P$  is a viable payment
14:    Append  $P$  to  $\mathcal{P}$ 
15:    Consume  $P$  in  $\mathcal{G}$ 
16:   end for
17: end for
18: return  $\mathcal{P}$ 
```

240 maximum amount of flow that the network allows from a source to a sink. Ford–Fulkerson
241 algorithm is one of the most effective algorithms to solve the maximum flow problem. Given
242 a weighted directed graph and two vertices. The Ford–Fulkerson algorithm uses Breadth-
243 First Search (BFS) [12] to find all paths between these two vertices. For each path, the
244 maximum viable amount is the minimum weight of edges. Algorithm 1 describes the process
245 of enumerating payments in Python syntax. First, it enumerates the binary combinations
246 of nodes who the adversary establishes channels as the starting and ending points of the
247 path. Second, it executes BFS to find all paths between each two adversary nodes similar
248 Ford–Fulkerson. Third, we derive the most viable amount using the least channel capacity
249 for each path. Last, it consumes this payment from the graph and adds this payment to our
250 payment list, i.e., we subtract the amount of this payment from the capacity of all channels
251 on the path.

252 3.5 Ranking payments

253 Different grieving payments have different impacts on the PCNs' liquidity. With limited
254 balance, the adversary has to start from grieving important payments for maximising the
255 attack's effect.

256 **Rank-by-length, -amount and -size.** We first consider three ranking criteria, namely
257 the length, amount, and size θ of a payment. *Rank-by-length* aims at maximising the effect
258 while minimising the cost, as long payments lock most capacity with the least amount.
259 *Rank-by-amount* aims at attacking the channel with large capacity. Since real-time balance
260 in LN cannot be seen, payer tends to prefer to go through channels with large capacities to
261 reduce the number of attempts. *Rank-by-size* aims at maximising the attack effect without
262 considering the custody, as payments with large sizes cost most collateral.



263 **Rank-by-fee.** Inspired by B’eres et al. [5], we consider *rank-by-fee*, where the adversary
 264 starts from attacking channels with lower fees. This aims at maximising the average channel
 265 fee of normal payments after the attack. The ranking criterium $Score(P)$ for payment P is
 266 calculated as

$$Score(P) = -\frac{\sum_{i=2}^l f_i}{l}$$

267 **Rank-by-bankrupt.** In addition, we consider the bankruptcy rates presented by Dandekar
 268 et al. [6, 17, 20] as a ranking criterium. The adversary first attacks channels that make most
 269 nodes “bankrupt”. Dandekar et al. introduced credit networks [6], where nodes are connected
 270 by edges with a limited resource called *credit*. We model the PCN as a credit network
 271 following Ramseyer et al. [17], where each channel’s credit is its capacity. Liquidity in a credit
 272 network is quantified as the probability that nodes become *bankrupt*, i.e., loss of all credit.
 273 Dandekar et al. [6] proved that the probability that a node v goes bankrupt is upper-bounded
 274 by $\frac{1}{\Gamma_v+1}$, where Γ_v is the total capacity of all channels connecting to v . Griefing can be
 275 seen as “removing” the capacity of nodes, and therefore increases the probability of nodes
 276 becoming bankrupt. In *rank-by-bankrupt*, the adversary first attacks payments that reduce
 277 the most mathematical expectations of the probability of nodes becoming bankrupt.

278 The bankruptcy criterium is quantified as the total increased probability $Score(P)$ of
 279 nodes in P becoming bankruptcy

$$Score(P) = Score_t(v_1, \alpha) + \sum_{i=1}^l Score_i(v_i, \alpha) + Score_t(v_{l+1}, \alpha)$$

where $Score_t(v_1, \alpha)$ and $Score_t(v_{l+1}, \alpha)$ are the increased probability of node v_1 and v_{l+1} ,
 respectively, and $Score_i(v_i, \alpha)$ is the increased probability of intermediate nodes v_i where
 $i \in [2, l]$. For node $v = v_1$ or v_{l+1} , the capacity is reduced by α , so

$$Score_t(v, \alpha) = \frac{1}{\Gamma_v - \alpha + 1} - \frac{1}{\Gamma_v + 1}$$

Meanwhile, for intermediate nodes $v = v_i$ where $i \in [1, l - 1]$, the capacity is reduced by 2α ,
 so

$$Score_i(v, \alpha) = \frac{1}{\Gamma_v - 2\alpha + 1} - \frac{1}{\Gamma_v + 1}$$

280 3.6 Launching attack

281 To obtain a list of griefing payments before launching the attack, we use channels’ capacities
 282 rather than their balances for determining the amounts of payments. As balances are
 283 fluctuating in real-time, some of the enumerated payments may not succeed during the attack.
 284 In real-world PCNs, if a node cannot route a payment, the node will reply to the payer with
 285 an error message, e.g., LN calls this error *InsufficientFunds*. Thus, we introduce a retry
 286 mechanism that, when a griefing payment is rejected, the adversary reduces its amount by a
 287 parameter *step*, and retries the same path until it is successful or the amount reaches zero.
 288 Algorithm 2 describes the attack process. To avoid being detected due to the retry pattern,
 289 the adversary can obfuscate the payment pattern, e.g., by dividing payments into multiple
 290 ones with random amounts.



■ **Algorithm 2** Launching attack.

Input:

```
1: The list of ranked payments  $\mathcal{P}_{ranked}$ 
2: The dropping step ratio  $step\_ratio$ 
3: The custody of the adversary  $B$ 
```

```
4: for  $P$  in  $\mathcal{P}_{ranked}$  do
5:   if  $B \leq 0$  then
6:     return ;
7:   end if
8:    $step\_amount \leftarrow step\_ratio * P.amount$ 
9:   while  $True$  do
10:     $response \leftarrow make\_payment(P)$ 
11:    if  $response = \text{InsufficientFunds}$  then
12:       $P.amount = P.amount - step\_amount$ 
13:      if  $P.amount \leq 0$  then break
14:      continue
15:    end if
16:     $B = B - P.amount$ 
17:    break
18:  end while
19: end for
```

291 **4** Quantifying PCNs' liquidity and congestion attacks' impact

292 We propose a generic method of quantifying PCNs' liquidity and congestion attacks' impact.
293 In our method, we generate a batch of payments, simulate them on the PCN, and calculate
294 liquidity metrics. The liquidity metrics include the success rate, the average cost and the
295 number of attempts of payments, and the number of bankruptcy nodes. A congestion attack's
296 impact is quantified as the liquidity difference before and after the attack.

297 **4.1** Generating payments for simulation

298 We follow the approach of Béres et al. [5] to test PCNs' liquidity. Specifically, we generate a
299 batch of n payments, of which payers and payees are random and the amount x_t is fixed. We
300 test multiple batches of payments with different amounts to cover regular payment scenarios,
301 which will be discussed in §5.

302 We simulate these payments in the PCN. Each payment is allowed to try r times to find
303 a viable path. If it finds a path within r tries, we consider it successful, otherwise failed.
304 Then the payments can be categorized into three states according to the status before and
305 after the attack, namely *added*, *survived*, or *removed*. *Added* means the payment fails before
306 the attack but is successful after the attack. *Survived* means the payment is successful both
307 before and after the attack. *Removed* means the payment is successful before the attack but
308 fails after the attack. Since our analysis is based on successful payments, payments that fail
309 both before and after the attack are ignored here.

310 Interestingly, some payments are *added*. Assuming a payments $P_1 = A \rightarrow B \rightarrow C \rightarrow D$
311 fails before the attack since BC has insufficient balance. For example, if some successful
312 payments that would have gone through BC failed after the attack, then channel BC would



313 become available to P_1 . Another example is that the attack may cause some payments to
314 change their paths. Suppose a successful payment P_2 originally went through EF and was
315 forced to go through CB after the attack, then the balance of BC will increase and make it
316 have enough balance to route P_1 .

317 4.2 Calculating liquidity metrics

318 We derive the PCNs' liquidity from the execution results of the simulated payments. We
319 consider the following five metrics: 1) amount of locked funds, 2) success rate of payments,
320 3) fee of payments, 4) average attempt times of payments and 5) the number of bankruptcy
321 nodes. A congestion attack's impact is quantified by the difference of liquidity metrics before
322 and after the attack.

323 5 Evaluation of congestion attacks

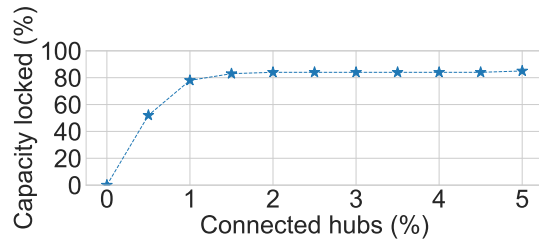
324 In this section, we evaluate congestion attacks on Bitcoin's Lightning Network (LN), the
325 first and most well-known PCN. We analyse the impact of congestion attacks with different
326 strategies in terms of the defined five liquidity metrics. Our results show that the adversary
327 who adopts congestion can severely limit the functionality of the entire PCN. Specifically,
328 the adversary can launch a congestion attack that locks 47% (~ 280 BTC) coins in the
329 network; reduces success rate of payments by 16.0% \sim 60.0%; increases fee of payments by
330 4.5% \sim 16.0%; increases average attempts of payments by 42.0% \sim 115.3%; and increase the
331 number of bankruptcy nodes by 26.6% \sim 109.4%, where the amounts of payments range from
332 0.001 to 0.019 BTC.

333 5.1 Experimental setting

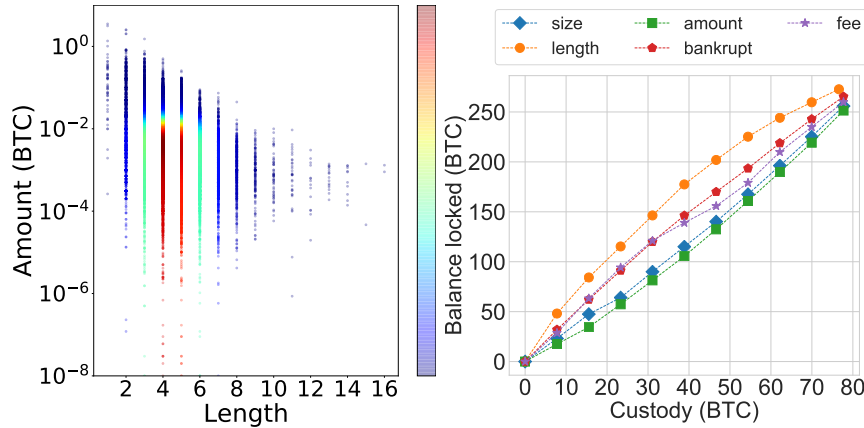
334 **Implementation and simulation setting.** We simulate and implement our attack
335 using Python 3.7.4 and NetworkX. For simplicity, we implement all algorithms sequentially.
336 Adversaries can use multi-threaded programming to speed up the algorithm if they prefer
337 efficiency. The topology provided by B'eres et al. [5] is the snapshot of the LN in 2019 (we
338 checked the network snapshot for 2021 and found the topology to be similar to 2019, so
339 we believe the results are similar of simulation on the 2021 snapshot). The snapshot also
340 includes the fee policy for each channel as well as the capacity. Since the balance distribution
341 characteristics of LN are not publicly available, we apply the random uniform distribution
342 for initialising the channels' balances similar to existing studies [5]. To amortise the bias
343 from randomness, we run each group of simulations with a certain strategy and custody level
344 for ten times. Our results show that the coefficient of variation for the quantitative impact
345 of the different balance distributions is only 1%.

346 **Payment routing mechanism.** In the real-world scenario, payments may sometimes fail,
347 as nodes cannot know the real-time balances of channels they do not involve. LN introduces
348 a *success probability* mechanism to optimise the routing. Specifically, if intermediate node
349 A is unable to forward a payment because of insufficient balance, then it will return an
350 error to the sender. The sender will temporarily reduce the success probability of this node.
351 The path finding mechanism of LN is finding the shortest path on a weighted graph. For
352 simplicity, we set the weight as channel fee. The routing algorithm is the plain Dijkstra [8]
353 algorithm. When an attempt fails, we temporarily remove the first node on the current path
354 with insufficient balance and try again. A payment is allowed to try r times for finding a
355 viable path. If it finds a path within r tries, it is successful, otherwise we consider it fails.





■ **Figure 3** The percentage of connected hubs v.s. locked capacity.



(a) Distribution of enumerated payments. (b) Locked balance.

■ **Figure 4** Characterisation of enumerated payments and the amount of locked balance.

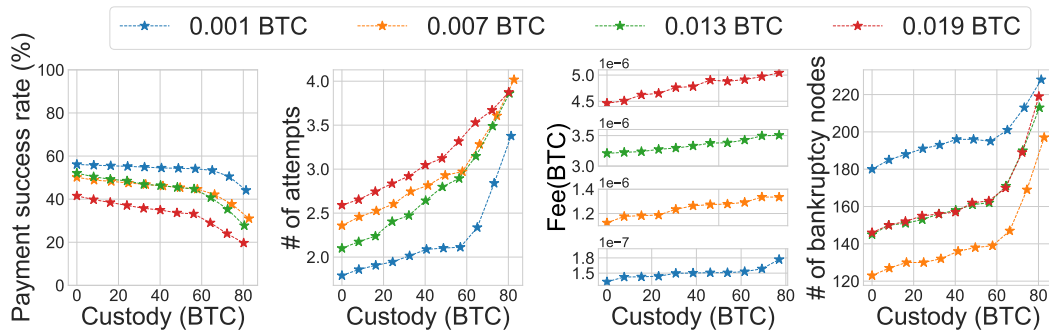
356 **Parameters.** We test attacks with different levels of custody of the adversary, i.e.,
 357 $\{7.7, 15.4, \dots, 77\}$ BTC, all ranking criteria in Section 3.5, and $step_ratio = 0.1$ in Al-
 358 gorithm 2. When testing LN’s liquidity, we pick batch size $n = 7000$ for testing liquidity (which
 359 is identical to existing works [5]), payment amount $x_t \in \{0.001, 0.007, 0.013, 0.019\}$ BTC,
 360 and payment retry times $r = 10$. In total, we ran $10 * 4 * 10 * 5 = 2000$ (retry times * # of
 361 payment amounts * # of different custody levels * # of strategies) simulations. We consider
 362 the threshold of bankruptcy as 0.006 BTC, which is the average amount of payments in
 363 LN [5].

364 **Choosing entry nodes.** We test the percentage of the capacity that the adversary can lock
 365 by establishing channels with different numbers of richest nodes in LN. Figure 3 shows that,
 366 by establishing channels with the top 1.5% (42) richest nodes, the enumerated payments take
 367 $\sim 83\%$ of the capacity of the entire network. In addition, the total amount of enumerated
 368 payments converges with the percentage of hubs increasing.

369 5.2 Impact of congestion attacks

370 We simulate the general congestion attack with five strategies in §3.5, and evaluate their
 371 impact in terms of the five metrics defined in §4. Figure 4 summarises the results, and
 372 Figure 7 in Appendix A provides the results in more detail. For Figure 4(b) and Figure 5,
 373 the baseline (when $x = 0$) is the scenario without any attack.

374 **Characterisation of enumerated payments.** As mentioned before, we establish chan-
 375 nels with the 42 richest nodes in the network. Algorithm 1 enumerates 35,402 payments in
 376 total. Figure 4(a) visualises the distribution of these payments w.r.t. their amounts and



■ **Figure 5** Overview of impacts of rank-by-fee.

377 lengths. Red indicates there are many grieving paths under that path length and payment
 378 amount, while blue means the opposite. The amount ranges from zero to 0.1 BTC, while
 379 the length ranges from 1 to 13. On average, most payments have a length of $3 \sim 6$ and an
 380 amount of $1e - 05 \sim 0.01$ BTC.

381 **Locked balance.** With a custody of 80 BTC (13% of the total capacity), an adversary can
 382 lock 280 BTC (47% of the total capacity) in LN, where rank-by-length is the most efficient
 383 strategy for locking balance. The average length of grieving payments is 3.8, which implies
 384 that there is room for optimisation of our path enumeration algorithm, since LN allows a
 385 maximum payment length of 20 hops.

386 **Impact on different liquidity metrics.** While Appendix A outlines the evaluation
 387 results of all attack strategies on all liquidity metrics, Figure 5 shows the result of the
 388 rank-by-fee strategy on LN as an example. With the rank-by-fee strategy and $7 \sim 80$ BTC
 389 as custody, the attack can reduce the payments' success rates by 21.4%~52.3%, increase the
 390 fee by 9.3%~27%, increase the number of attempts by 50%~88.7%, and increase the number
 391 of bankrupt nodes by 26.7%~60%.

392 6 Discussion

393 **Budget analysis.** The budget of launching congestion attacks is twofold: 1) *channel*
 394 *fee* for establishing channels and 2) *custody* deposited into channels. When preparing for
 395 a congestion attack, the adversary needs to pay the transaction fee for opening channels.
 396 Transaction fee is negligible as analysed in §5.1. After the congestion attack, the custody
 397 is refunded as payments are expired. For LN, the required custody is 77 BTC (13% of the
 398 network capacity). In Bitcoin, there are more than 10,000 addresses with more than 157
 399 BTC [2], making them having sufficient capacity to launch a congestion attack.

400 **Profit from congestion attacks.** The adversary can apply grieving on other nodes'
 401 channels, so that more payments go through its controlled channels. To receive most fees
 402 following this approach, the adversary redirects as many payments to its channels as possible.
 403 Existing research [22] shows that the probability is proportional to the adversarial node's
 404 *betweenness centrality*, while maximising the *betweenness centrality* by removing channels
 405 can be formalised as the *destructive betweenness improvement* problem that is NP-hard [11].
 406 To our knowledge, there exists no approximation algorithm to solve this problem, and we
 407 consider designing such algorithm as future work.



408 **7** Related work and comparison

409 **7.1** Attacks on PCNs

410 **Congestion attacks.** Congestion attack was informally discussed by Lightning Network
411 community [1]. Mizrahi et al. [13] first systematically studied the congestion attack on PCNs.
412 In their proposed attack, the adversary makes a large number of small payments, in order to
413 make channels hit *max_concurrent_htlcs*, the maximum number of concurrent payments.
414 Tikhomirov et al. [21] used the same idea to lock the balance of the channel, but they only
415 grief a single channel at a time.

416 The two congestion attacks focus on a single attack liquidity metric, or put limitations
417 on the attack strategy, and therefore can be seen as special cases of our general congestion
418 attack. In addition, as their attacks focus on a single path or channel at a time, the adversary
419 has to establish new channels when attacking a new path or channel. Establishing a large
420 number of channels makes the adversary easier to be identified, and existing nodes may
421 not be willing to establish too many channels in a short time period. Moreover, to occupy
422 *max_concurrent_htlcs*, the adversary in their two attacks has to make a large number of
423 concurrent payments compared to our attack. This also makes the adversary's behaviour
424 easier to be identified.

425 **Other attacks on PCNs.** There have been attacks on PCNs with different goals. In
426 the lockdown attack [15], the adversary grieves the victim's channels to isolate it from the
427 network. In the hijacking attack [22], the adversary publishes channels with small fee to
428 attract payments, and withhold all payments through its channels. Rohrer et al. [19] discussed
429 two attacks, namely channel exhaustion and node isolation. While congestion attacks aim
430 at paralysing the entire PCN, these three attacks aim at exhausting individual channels or
431 isolating individual nodes.

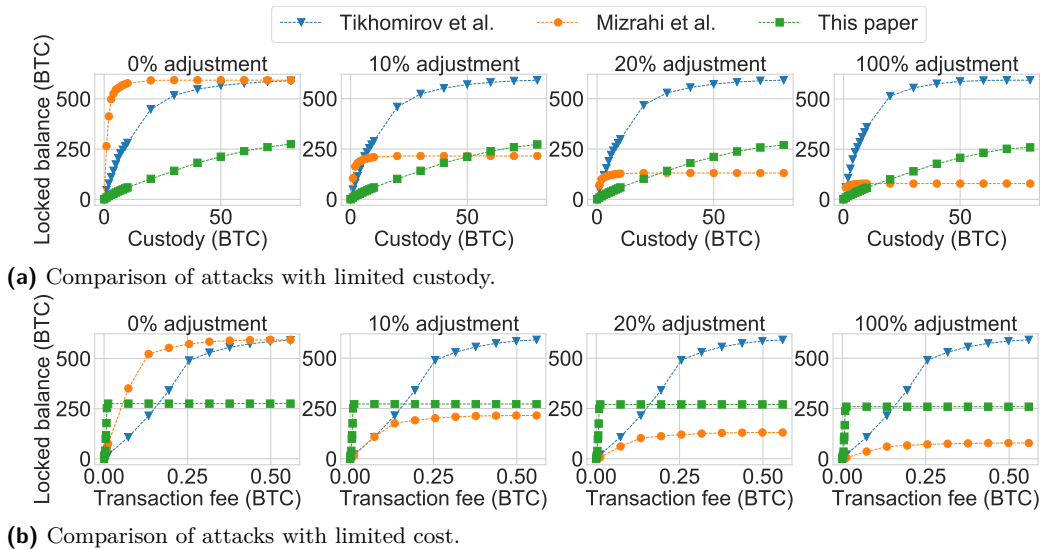
432 **7.2** Quantitative comparison with existing congestion attacks

433 We quantitatively compare existing congestion attacks [13,21] with ours w.r.t. different
434 budget level of custody and channel fee and different *max_concurrent_htlcs* value distribution.
435 For both attacks, we simulate the capacity-first strategy. The strategy iterates the following
436 process: when a path is enumerated, calculate the total capacity of involved channels whose
437 *max_concurrent_htlcs* values have been filled, then remove these channels from the network.
438 Locking a channel by using *max_concurrent_htlcs* takes *max_concurrent_htlcs* * 2 payments
439 (as a channel has two directions). The smallest payment amount is 5.46e-06 BTC (i.e. the
440 dust limit). Thus, the custody required for grieving a path is $2 * max_concurrent_htlcs * 5.46e-06$ BTC.
441 When enumerating a path, we check whether both ends have channels with
442 the adversary. If not, the adversary has to establish channels with them, leading to a fee of
443 0.0002 BTC (~ 18.89 USD at the time of writing).

444 To quantify the impact of *max_concurrent_htlcs*, we test the locked capacity when
445 different portions of channels adjust *max_concurrent_htlcs*. Given the size limit of Bitcoin
446 transactions, the maximum value of *max_concurrent_htlcs* in Bitcoin's LN is 483. Thus,
447 we assume the adjusted value of *max_concurrent_htlcs* is uniformly distributed in interval
448 [1, 483]. When the custody is limited, we assume the fee is unlimited, and vice versa.

449 Figure 6 shows the experimental results. Each experiment is repeated 10 times, and
450 the variation of experimental results is about 2.4%. As the results are similar after the
451 20% channel adjustment, we skipped the simulation in 30%-90% for brevity. Figure 6(a)
452 shows the performance of the three attacks under different custody. When all channels share





■ **Figure 6** Comparison with congestion attack. $x\%$ adjustment means $x\%$ channels adjust their $max_concurrent_htlcs$.

453 the same $max_concurrent_htlcs$, Mizrahi et al.’s attack locks most capacity. When more
 454 channels adjust $max_concurrent_htlcs$, the locked capacity becomes less. This is because
 455 when channels in a path have different $max_concurrent_htlcs$ values, the adversary can only
 456 congest the channel with the smallest $max_concurrent_htlcs$ in this path by spamming this
 457 path only, making the strategy of Mizrahi et al. less effective. Meanwhile, Tikhomirov et
 458 al.’s attack and our attack are not affected by $max_concurrent_htlcs$. This is because our
 459 attack does not rely on $max_concurrent_htlcs$ and the number of concurrent htlcs occupied
 460 by our attack averaged only 3.8 per channel, and Tikhomirov et al.’s attack focuses on a
 461 channel at a time. Figure 6(b) shows that, both Tikhomirov’s and Mizrahi’s attacks require
 462 more transaction fee compared to our attack. This is because, in their attacks, the adversary
 463 has to open a new channel when attacking a new path. With sufficient transaction fee,
 464 Tikhomirov’s locks more money compared to our attack.

465 To lock in 250 BTC of liquidity. The attack by Mizrahi et al et al. requires 1 BTC of
 466 custody and pays a transaction fee of 0.05 BTC, the attack by person Tikhomirov et al.
 467 requires 8 BTC of custody and a fee of 0.15 BTC, while our attack requires 65 BTC of
 468 custody and a fee of 0.008 BTC. Therefore, if the attacker is fee-sensitive, then our attack is
 469 preferred because our fees are 16% of and 5% of other two. Whereas, if the attacker has a
 470 restricted custody in hand, then the attack by person Mizrahi et al. is more preferred, as the
 471 custody required is only 1.5% of our attack.

472 8 Conclusion

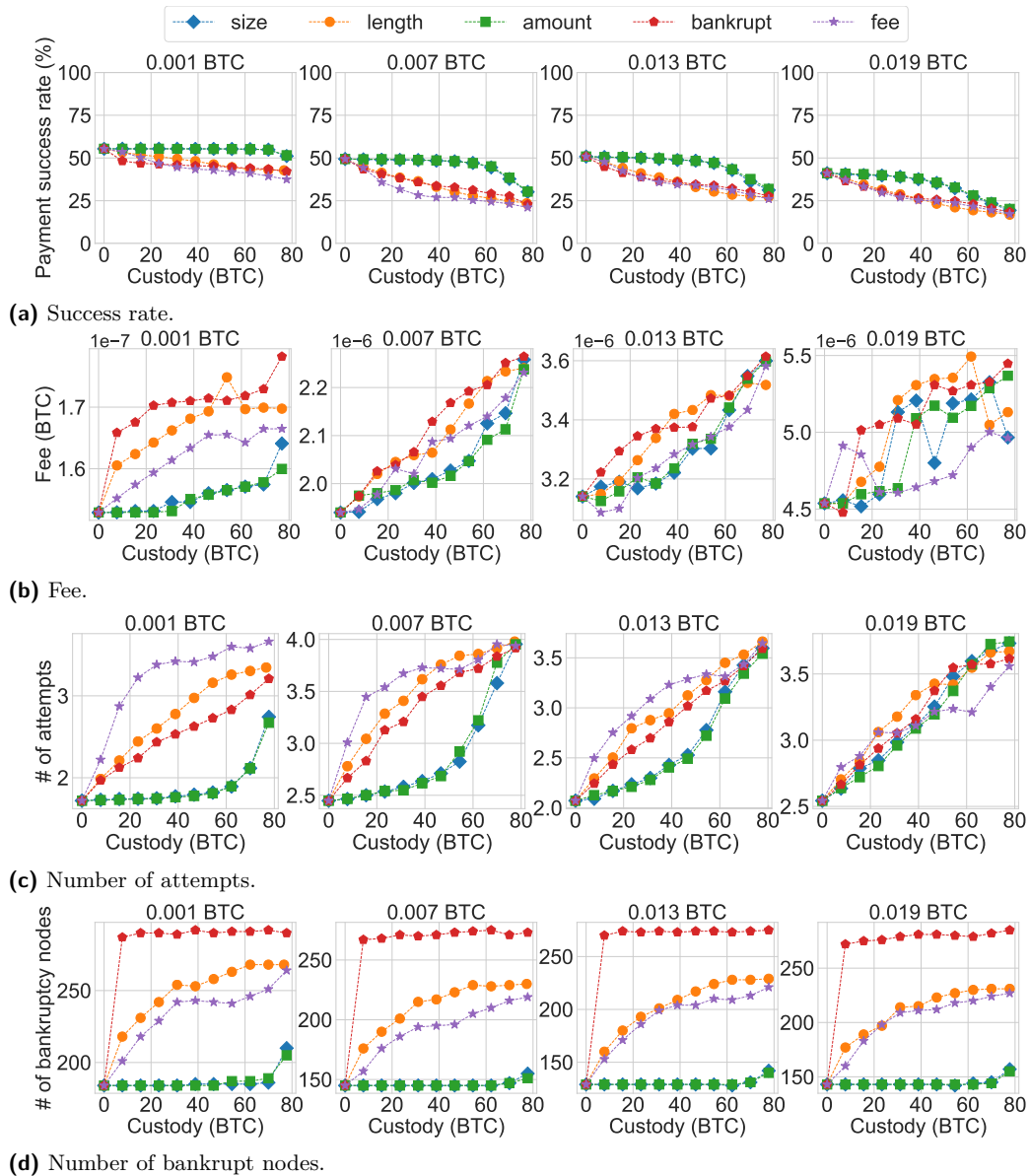
473 In this paper, we propose the general congestion attack on payment channel networks
 474 (PCNs). Our general congestion attack generalises the existing congestion attacks in terms
 475 of attack strategies, targeted metrics and optimisation techniques. We develop concrete
 476 steps for launching congestion attacks, and provide a generic method of quantifying PCNs’
 477 liquidity and effectiveness of congestion attacks. We evaluate our congestion attacks on
 478 Lightning Network – the first and most well-known PCN. Our evaluation results show that
 479 the congestion attack is cheap to launch and can greatly reduce the LN’s liquidity.



- 481 1 Payment channel congestion via spam-attack. [https://github.com/lightningnetwork/](https://github.com/lightningnetwork/lightning-rfc/issues/182)
482 [lightning-rfc/issues/182](https://github.com/lightningnetwork/lightning-rfc/issues/182). Github, 2017.
- 483 2 BitInfoCharts. [https://bitinfocharts.com/top-100-richest-bitcoin-addresses-1.](https://bitinfocharts.com/top-100-richest-bitcoin-addresses-1.html)
484 [html](https://bitinfocharts.com/top-100-richest-bitcoin-addresses-1.html), 2020. [Online; accessed 20-September-2020].
- 485 3 Bitrefill. <https://www.bitrefill.com/>, 2020. [Online; accessed 20-September-2020].
- 486 4 Evan Schwartz Akash Khosla and Adrian Hope-Bailie. Interledger rfcs, 0018 draft 3, connector
487 risk mitigations. <http://j.mp/2m20vfP>, Github, 2019.
- 488 5 Ferenc Béres, Istvan Andras Seres, and András A Benczúr. A cryptoeconomic traffic analysis
489 of bitcoins lightning network. *arXiv preprint arXiv:1911.09432*, 2019.
- 490 6 Pranav Dandekar, Ashish Goel, Ramesh Govindan, and Ian Post. Liquidity in credit networks:
491 A little trust goes a long way. In *Proceedings of the 12th ACM conference on Electronic*
492 *commerce*, pages 147–156, 2011.
- 493 7 George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In
494 *2009 30th IEEE Symposium on Security and Privacy*, pages 269–282. IEEE, 2009.
- 495 8 Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*,
496 1(1):269–271, 1959.
- 497 9 Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian*
498 *journal of Mathematics*, 8:399–404, 1956.
- 499 10 Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, Alejandro Ranchal-Pedrosa, Cristina
500 Pérez-Solà, and Joaquin Garcia-Alfaro. On the difficulty of hiding the balance of lightning
501 network channels. In *Proceedings of the 2019 ACM Asia Conference on Computer and*
502 *Communications Security*, pages 602–612, 2019.
- 503 11 Clemens Hoffmann. Algorithms and complexity for centrality improvement in networks. 2017.
- 504 12 Dexter C Kozen. Depth-first and breadth-first search. In *The design and analysis of algorithms*,
505 pages 19–24. Springer, 1992.
- 506 13 Ayelet Mizrahi and Aviv Zohar. Congestion attacks in payment channel networks. *arXiv*
507 *preprint arXiv:2002.06564*, 2020.
- 508 14 Olaoluwa Osuntokun, Conner Fromknecht, Wilmer Paulino, Oliver Gugger, and Johan Halseth.
509 Lightning pool: A non-custodial channel lease marketplace. 2020.
- 510 15 Cristina Pérez-Sola, Alejandro Ranchal-Pedrosa, J Herrera-Joancomartí, Guillermo Navarro-
511 Arribas, and Joaquin Garcia-Alfaro. Lockdown: Balance availability attack against lightning
512 network channels, 2019.
- 513 16 Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant
514 payments, 2016.
- 515 17 Geoffrey Ramseyer, Ashish Goel, and David Mazières. Liquidity in credit networks with
516 constrained agents. In *Proceedings of The Web Conference 2020*, pages 2099–2108, 2020.
- 517 18 Daniel Robinson. Htlcs considered harmful. In *Stanford Blockchain Conference*, 2019.
- 518 19 Elias Rohrer, Julian Malliaris, and Florian Tschorsch. Discharged payment channels: Quanti-
519 fying the lightning network’s resilience to topology-based attacks. In *2019 IEEE European*
520 *Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 347–356. IEEE, 2019.
- 521 20 Vibhaalakshmi Sivaraman, Weizhao Tang, Shaileshh Bojja Venkatakrishnan, Giulia Fanti, and
522 Mohammad Alizadeh. The effect of network topology on credit network throughput. *arXiv*
523 *preprint arXiv:2103.03288*, 2021.
- 524 21 Sergei Tikhomirov, Pedro Moreno-Sanchez, and Matteo Maffei. A quantitative analysis
525 of security, anonymity and scalability for the lightning network. In *2020 IEEE European*
526 *Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 387–396. IEEE, 2020.
- 527 22 Saar Tochner, Aviv Zohar, and Stefan Schmid. Route hijacking and dos in off-chain networks.
528 In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages
529 228–240, 2020.



A Detailed evaluation results and analysis

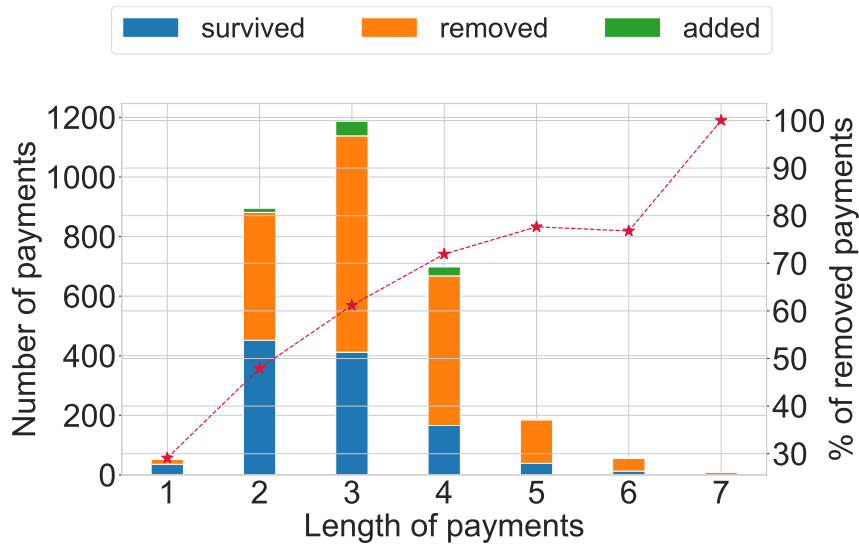


■ **Figure 7** Overview impact of all strategies.

531 **Payment success rate (Figure 7(a)).** Larger payments are less likely to succeed both
 532 before and after the attack. This is because if a payment's amount is bigger, fewer channels
 533 can route this payment. In addition, rank-by-fee is more effective than the other four
 534 strategies in terms of the success rate. We suspect the reason to be that the rank-by-fee
 535 strategy exhausts some channels that route a large number of payments by specifying a low
 536 fee rate.

537 Rank-by-amount does not perform well when custody is limited and the amount of
 538 payments is low. This is because the adversary starts from attacking channels with high





■ **Figure 8** Status of payments. Red line indicates the percentage of removed payments. The bar shows the number of payments in different states at different lengths and the line chart shows the percentage of payments blocked at different lengths.

539 capacity, and therefore concerns less about payments with small amounts. As the amount of
 540 payment increases, the gap between rank-by-amount and other strategies begins to narrow,
 541 which confirms our thinking. In addition, rank-by-amount and rank-by-size have similar
 542 performance in success rate. This is because the distribution of payment lengths is centralised
 543 as shown in Figure 4(a), and the payment amount becomes the main factor determining the
 544 payment size.

545 **Average fee of payments (Figure 7(b)).** Note that we only count *survived* payments
 546 when calculating the fee. Payments with long paths are more likely to be blocked by the
 547 attack and removing them will decrease the average fee, offsetting the rise caused by the
 548 attack. Thus, the average fee of successful payments fail to reflect the increase in fees caused
 549 by the congestion.

550 We use the scenario where the payment amount is 0.013 BTC as an example. The result
 551 (in Figure 8) shows that, long payments are more likely to be influenced by our attack.
 552 Specifically, all 7-hop payments are blocked by the attack, while only about 30% of 1-hop
 553 payments are affected. This is because a long payment indicates that there are more channels
 554 in the path, so the payments are more likely to be attacked. These long path payments that
 555 fail as a result of the attack will result in a decrease in the average fee. Therefore, we only
 556 count *survived* payments to give a true evaluation of the effect of the attack.

557 The five strategies performed similarly in terms of fees, similar to payment success rates.
 558 The fee upward trend of the fee is because the attack will make channels with low fee
 559 unavailable, and payments will be re-directed to channels with high fee. However, sometimes
 560 the fee decreases as the custody increases. This is because the attack does not only block
 561 payments, but also add some payments, as we observe in §4.1. Both *blocked* payments and
 562 *added* payments may make some channels with low fee to be available again.

563 Congestion attacks can be launched together with the route hijacking attacks [22]. In
 564 a route hijacking attack, the adversary uses channels with zero fee to attract and hijack
 565 payments. Congestion attacks can increase payment fee, and therefore make the adversary's
 566 channels more attractive to payments.



■ **Table 1** The proportion μ of deceptive channels out of channels with capacities greater than the payment amount. Parameter \bar{a} is the number of attempts before the attack.

	0.001	0.007	0.013	0.019
μ	15.0%	31.8%	31.2%	39.2%
\bar{a}	1.76	2.59	2.66	3.11

567 **Average attempt times of payments (Figure 7(c)).** When the amount of payments
 568 increased from 0.007 BTC to 0.013 BTC, the number of attempts does not rise significantly.
 569 In addition, the average number of attempts with 0.001 BTC is much lower than payments
 570 with a bigger amount. We observe that the average number of attempts is proportional
 571 to the number of “deceptive channels”. We say a channel is deceptive for a payment with
 572 amount x when the capacity of the channel is greater than x , but the balance is less than x .
 573 The number of attempts depends on the ratio between the number of deceptive channels and
 574 the number of channels with balance greater than the payment amount. As aforementioned,
 575 we assume the balance of each channel is uniformly distributed. Under this assumption,
 576 given the amount of simulated payments x , this proportion μ is calculated as

$$\mu = \frac{\sum_{i=1}^n \frac{x}{c_i}}{n}$$

577 where n is the number of channels with capacities greater than x , and c_i is the i -th
 578 channel’s capacity. Table 1 shows the trend of μ and \bar{a} is consistent, which confirms our
 579 suspicions.

580 **Number of bankruptcy nodes (Figure 7(d)).** When it comes to bankruptcy rates,
 581 the rank-by-bankrupt strategy is particularly effective in the number of bankruptcy nodes
 582 and the success rate of small payments. Figure 7(d) compares their performance. The
 583 results show that when the attack custody is limited, rank-by-bankrupt outperforms all other
 584 strategies in terms of the number of bankruptcy nodes. The additional bankruptcy nodes
 585 caused by rank-by-bankrupt have the characteristic of poor capacity. Specifically, the average
 586 capacity of these victims is 0.02 BTC, while the average capacity of all nodes in the network
 587 is 0.42 BTC, which is consistent with the theory we cited in §3.5.

