REBAL: Channel Balancing for Payment Channel Networks

Nitin Awathare†, Suraj†, Akash *, Vinay Joseph Ribeiro† and Umesh Bellur†
†Department of Computer Science and Engineering and *Electrical Engineering, Indian Institute of Technology Bombay
{nitina, surajyadav}@cse.iitb.ac.in, {akash.kumar,vinayr, umesh.bellur}@iitb.ac.in

***Abstract*—Cryptocurrency networks are a promising infrastructure for pseudonymous online payments. However, low throughput has prevented their widespread acceptance. A promising solution to scale throughput is the Payment channel network (PCN), exemplified by the Lightning Network (LN), that uses a network of off-chain bidirectional payment channels between parties that wish to transact often. Since payments use the shortest paths with sufficient funds over this network, channel balances get exhausted in the direction transactions flow and eventually become unidirectional. This results in transactions failing and consequently a lower transaction success ratio. Our observations on the production LN show that over 63% of the channels lose over 80% of the channel balance in one direction over time, which makes the success ratio of a real-world workload drop from 71% to 29%. A unidirectional channel along a path results in a failure message back to the source that recomputes the path, excluding the failed channel and reattempts the transaction, thus adding to the completion latency even for those transactions that do complete.**

We propose REBAL**, a distributed re-balancing mechanism, and a new routing scheme to address the above issues.** REBAL **maximizes the extent to which channels can be re-balanced across the entire network.** REBAL **addresses the completion latency issue by re-routing transactions from intermediate nodes around a unidirectional channel rather than propagating the failure back to the source.**

Our comprehensive evaluation of REBAL **shows that the success ratio improves from 30.18% to 79.54% and success volume from 3.98% to 29.99% for a real-world workload derived from the Ripple network, without adversely impacting the transaction latency. Even at very high transaction rates,** REBAL **outperforms Lightning Network Daemon** (LND**- a Golang implementation of LN) (12%) with a success ratio of 43.76%.**

I. Introduction

Blockchains are decentralized platforms that support crytocurrency applications. Bitcoin is the first such cryptocurrency built on a permisionless blockchain. Despite being popular, its practicality is questionable because of two main factors: 1) high transaction fees and; 2) long block confirmation time and consequently low throughput (≈7 transactions per second).

One solution to these issues is a Payment Channel Network (PCN) [3], [15], [21] that consists of a set of off-chain bidirectional payment channels between interested parties. Once a channel is created between users on the PCN, they can transact off-chain as long as there are sufficient funds along the path connecting them. Since establishing a channel requires on-chain transactions that are expensive, it is recommended not to create a direct payment channel for every pair of users that need to transact. By routing payments through intermediate payment channels, participants in the PCN can transfer funds even if they do not share a direct channel.

Lightning Network (LN) [21], the best known PCN, has over 22K nodes [4]. However, PCNs, including the LN, have their challenges. A major issue is that as transactions flow through the network, channel balances of those channels in the transaction path get progressively exhausted. This makes such channels "unidirectional or imbalance" over time, resulting in failed transactions on paths containing these channels and consequently results in a lower success ratio (ratio of successful to total transactions issued).

Existing work to improve the transaction success ratio can be categorized as follows: 1) replenishing capacity by going to the main chain [20], 2) re-balancing the channels in-place [13], [16] and 3) load balancing transactions across paths from the source so that it takes longer to exhaust the capacity of all paths between a source-destination pair, but it doesn't entirely avoid it [14], [18]–[20]. Replenishing channel capacity is expensive since it requires going to the main chain. Dynamic re-balancing in place can be centrally controlled or distributed. The former suffers from the issue of halting the PCN's transactions while re-balancing is in progress. It also requires the nodes to disclose its individual contribution in the channel to the central controller, thus violating privacy. The distributed re-balancing strategy in [16] attempts to split the capacity of a node equally across all channels it is connected to and suffers from not being cognizant of the node's workload, which is shown to be non-circular and deals with transactions primarily in one direction [9].

Based on our analysis of real world workloads and our simulations of these transactions on the real LN topology, we propose REBAL - a distributed in-place re-balancing approach. It re-balances the maximum number of channels in the network to the greatest possible extent by preferentially selecting the longest cycle each node is involved in to start the re-balancing. Unlike previous work [13], REBAL preserves privacy and does not halt the PCN when the re-balancing process is active. Finally, REBAL takes cognizance of the workload history while making decisions about the re-balancing amounts.

To avoid rejection of transactions when a channel balance is temporarily locked during re-balancing, REBAL allows an intermediate node to re-route the transaction through an alternative path to the next node.

We have implemented REBAL and evaluated its effectiveness using a series of real-world LN topology snapshots in addition to a few synthetic topologies [1], [8] against the Ripple [5] transaction workload. REBAL outperforms all the state-of-the-art algorithms, even when the transaction generation rate is high. For the LN topology, REBAL improves the success ratio (ratio of successful to total transactions issued) from 30.18% to 79.54% and success volume (ratio of successful to total transactions amount processed) from 3.98% to 29.99%, without significantly affecting the transaction latency.
In summary, we make the following contributions in this paper:

- A distributed in-place re-balancing strategy for PCNs, REBAL, that reduces channel imbalances in the network and hence improves the transaction success ratio. Coupled to this, we also introduce a dynamic re-routing scheme from intermediate nodes to reduce transaction latency of successful transactions.
- A theoretical proof that REBAL indeed minimizes the imbalance in the network.
- A concrete, real implementation and comprehensive evaluation of REBAL, with real transaction data on a real topology and empirical evidence that REBAL outperforms the state-of-the-art routing schemes.

The rest of this paper is organized as follows: §II describes the background required to understand the work. §III describes the re-balancing strategy, followed by re-routing at an intermediate node. §IV describes a prototype implementation of REBAL on a simulated environment, experimental setup and the observations from the obtained results. Further we explained the related work in §V and end the paper with discussion and future direction in §VI.

II. Background & Motivation

In this section we lay out the background material to understand our work.

Payment Channel Networks A Payment channel network (PCN) is a set of bidirectional payment channels between users/nodes that wish to transact often with each other. Each channel is created using a funding transaction on the blockchain that locks the funds from both parties. Now, the parties can transact with each other off-chain instantly and without fees and keep a log of the latest balance. Suppose nodes X and Y initially contribute an amount of a and b to the channel X–Y respectively. A transaction of amount z from X to Y (with the constraint that z does not exceed X's current balance) updates the balances of X and Y to $a - z$ and $b + z$ respectively. Similarly Y can pay X any amount not exceeding its current balance, and the balances of the two parties are updated accordingly.

Since payments can take place in both directions, such channels are called "bidirectional". In case the channel balance of one part (e.g. X) becomes zero, then that party (X) can no longer use the channel to pay the other party (Y), in which case we say the channel has become "unidirectional". A payment from Y to X would then make X's balance non-zero, and the channel bidirectional again. When the channel is no longer required, an on-chain transaction signed by both parties refunds to them their latest channel balances and terminates the channel.

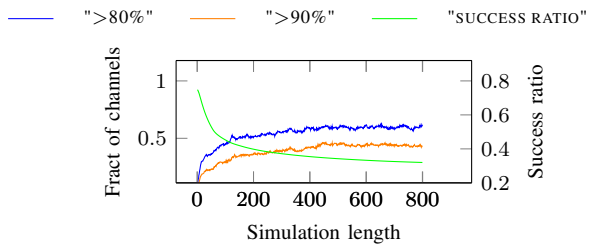


Figure 1: Channel count for particular % of imbalance and average success ratio, with time (in terms of simulation length; 1 unit of simulation length = 1 sec)

A PCN allows a sender and receiver to transact without sharing a direct channel. For a transaction, if the sender and receiver are connected via a direct channel, they can easily transact as described above. But in case they are not directly connected, the sender finds a (multi-hop) route to the receiver and sends the funds along the route [17], thereby using other payment channels for routing by providing a fee to the intermediate nodes.

The blockchain is involved only for creating and closing the channels and for dispute resolution. Limiting the use of blockchain for these purposes reduces the transaction confirmation latency from approximately 60 minutes to a few seconds in LN. Furthermore, sending the transaction through the private channels requires routing fees which is ≈500 times lower than the fees for on-chain transactions.

A party can send a payment to another provided there exists a multi-hop path from the source to destination and the payment amount is less than the minimum channel balance of all channels along the path (in the direction of the payment). Each node has complete visibility of the topology of the network, but is not aware of the latest balances of channels, except for channels with its own neighbours. Therefore, a transaction may not succeed. This is because an intermediate channel on the path may not have sufficient balance to forward the incoming transaction. In which case, it sends an error message back to the source. The source retries the transaction along alternate paths till it times-out, runs out of paths, or the transaction succeeds.

Motivation We have simulated the performance of the LN topology using a real-world Ripple workload [6] to understand the reasons for transaction failures. Our experiments, where all the channels are balanced initially (at time 0), reveal that with time (approximately 200 seconds), for over 63% of the channels, 80% of the channel balance is along one side of a bidirectional channel. Due to channel balance getting exhausted in one direction, the success ratio for this workload drops from 71% to 29% in about 600 seconds , as depicted in figure 1.

An analysis of the state of the LN topology network with the Ripple workload reveals that when the network reaches a state of imbalance indicated above, it is possible to re-balance the network such that the imbalance of around 60% of channels

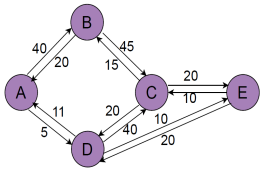


Figure 2: Small LN topology where number on each edges represents the contribution of the node in the channel.

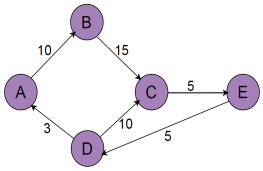


Figure 3: Imbalance graph obtained from the LN Topology in figure 2

can be cut in half without the need for an on-chain transaction. So, if we can periodically re-balance the network, it will help the PCN to complete a large number of transactions without putting any load to the main chain.

III. REBAL: Distributed In-Place Re-balancing

This section describes our re-balancing strategy. We start with an algorithmic approach, analyze its feasibility and then suggest a heuristic.

A. Problem formulation and optimistic algorithm

Recall that in the LN, each node has visibility of the entire LN topology but has access to the latest balances of only the channels to which it is connected. Each node can thus calculate how (im)balanced the channels to which it is connected are. It does this as follows. Let Δ be the maximum time a node takes to perform re-balancing. If i and j are the two parties involved in the channel C_{ij}, let Θ_i and Θ_j be the average incoming transaction amount for re-balancing time (Δ) for channel C_{ij} at node i and j respectively. Note that Θ is calculated from past transaction history. Let θ be the average rate of arrival of transactions amount. Then $\Theta = \theta * \Delta$. Each REBAL node maintains a queue of transactions waiting to be processed. Let λ_i and λ_j be the sum of transaction amounts waiting in the queue for C_{ij} at i and j respectively. Also, bal_{ij} and bal_{ji} are the individual contributions of i and j in the channel C_{ij}, and we define channel capacity $Cap(C_{ij}) = bal_{ij} + bal_{ji}$. To consider incoming and outgoing Θ and λ while making the re-balancing decision, we define the projected imbalance factor of the channel for time Δ as follows:

$$ImBal_{C_{ij}} = \left| \frac{(\Theta_i + \lambda_i + bal_{ij}) - (\Theta_j + \lambda_j + bal_{ji})}{Cap(C_{ij})} \right| \quad (1)$$

Each node i aims to reduce $ImBal_{C_{ij}}$ to be below a certain threshold, Thr. Ideally we would like $ImBal_{C_{ij}} = 0$. Node i does this by transferring amount $idealBal_{ij}$ (calculated using eq 2) to j if $(\Theta_i + \lambda_i + bal_{ij}) > (\Theta_j + \lambda_j + bal_{ji})$ and $bal_{ij} \geq idealBal_{ij}$.

$$idealBal_{C_{ij}} = \frac{Cap(C_{ij}) * (ImBal_{C_{ij}} - Thr)}{2} \quad (2)$$

Given this, we now define a *Imbalance graph* (G_{ImBal}) that has all the nodes of the original graph representing the PCN. Each directed edge of G_{ImBal} represents the amount one node should send another to reduce $ImBal_{C_{ij}}$ to Thr. For example, G_{ImBal} of the LN topology in figure 2 for $Thr = 0$ is given in figure 3 (For simplicity of representation we have omitted the Θ and λ from it). *Note:* no single node knows the entire Imbalance graph. They only have a local view of it, i.e., the channels the node is connected to. A pair of nodes <*i*,*j*> do not have an edge in the imbalance graph if the $ImBal_{C_{ij}} \leq Thr$. Throughout the re-balancing process, we do not allow a node to reverse the edge direction because we are considering pending as well as future transactions while calculating $ImBal_{C_{ij}}$. From now on, unless explicitly stated, all the further discussion in this section is about the Imbalance graphs.

The chance that a transaction succeeds is directly related to the extent the network is balanced. So our aim is to reduce the imbalance factor over as many edges in G_{ImBal} to the maximum possible extent, i.e., minimize $\boldsymbol{I} = \sum_{i,j \in V(G_{ImBal})} |ImBal_{C_{ij}}|$. This is similar to finding the maximum circulation in a network [23]. The strategy is to re-balance channels by a node sending some amount to the other across the channel that is imbalanced. If f_{ij} is the (re-balancing) amount that i sends through channel C_{ij} to j, then, transferring the maximum value of f_{ij} ($0 \leq f_{ij} \leq idealBal_{C_{ij}}$), from i to j can reduce the $ImBal_{C_{ij}}$ to the maximum possible extent. At the same time i should receive the same amount from its neighbours who have incoming edges to it, in imbalance graph, so that it does not touch its total amount. This can be done by transferring the amount along a *cycle*. If N_i^-, N_i^+ are incoming and outgoing neighbours of node i respectively, then the problem can be formulated as 3.

$$\begin{aligned} &\text{maximize} \sum_{<i,j> \in E(G_{ImBal})} f_{ij} \\ &\text{such that} \\ &0 \leq f_{ij} \leq idealBal_{C_{ij}} \text{ and} \\ &\sum_{k \in N^-(i)} f_{ki} = \sum_{j \in N^+(i)} f_{ij} \quad \forall i \in V(G_{ImBal}) \end{aligned} \quad (3)$$

Note that to minimize I we have to maximize our objective function in 3. For this, the maximum possible amount should be transferred through each cycle in G_{ImBal}. In other words, no further transfer is possible, i.e., $idealBal_{C_{ij}}$ for at least one channel C_{ij} along the cycle should become 0. But the maximum possible amount can be transferred in many possible ways, which may or may not minimize I. For example, the value of maximum possible amount for the imbalance graph in figure 2 is 5 which can be transferred in 4 possible ways (assuming integer amount only for ease of representation) as depicted in figure 4a, 4b, 4c and 4d. Note that we are not

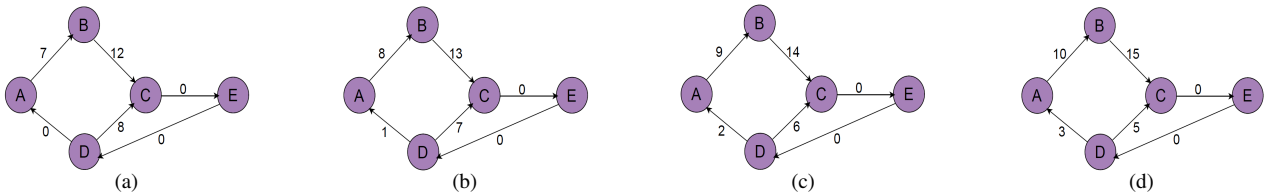


Figure 4: Ways of transferring maximum possible amount in the imbalance graph

allowing any edge to reverse its direction (handled by first constrain in 3)

Definition III.1. (*Cycle saturation*) A cycle in the imbalance graph is said to be saturated if it contains at least a link C_{ij} with zero $idealBal_{C_{ij}}$ in the forward direction of the cycle i.e. a node can not transfer any further amount through that cycle.

Definition III.2. (*Flow-dependent cycle*) A set of n cycles $C_1, C_2, \cdots, C_n$ in the imbalance graph are flow-dependent if they share at least an edge and a node, along the shared edge, cannot send the amounts that it receives from other nodes along all the cycles In other words, it is not possible to saturate (III.1) all n cycles.

Theorem 1. *Given n flow-dependent cycles $C_1, C_2 \cdots, C_n$ such that $L(C_1) > L(C_2) > \cdots L(C_n)$, where $L(C_i)$ is the length of the cycle. If k out of n cycles can be saturated (following constraints in eq 3) and order of saturation is $L(C_1), L(C_2), \cdots L(C_k)$ then the objective function 3 will be maximized.*

Proof. We will prove this by contradiction. The value of the objective function in 3 is given as follows:

$$f_{cur} = f_1 * L(C_1) + f_2 * L(C_2) + \cdots + f_n * L(C_k) \tag{4}$$

Lets assume that order of saturating the cycle is C_1 followed by C_2 followed by C_k and so on, and f_{cur} is not maximum i.e. there exist f_{max} such that $f_{max} > f_{cur}$. If f_{cur} is not maximum then more amount can be transferred. So one has to reduce the previously transferred amount from either of $C_1, \cdots, C_k$ and transfer it along another cycle, say C_z. If x is the minimum amount value that one can transfer and we will reduce that amount from C_i where $1 \leq i \leq k$ then updated value of equation 3 is given as follows:

$$f_{max} = f_1 * L(C_1) + f_2 * L(C_2) + \cdots + f_n * L(C_k) + (x * L(C_z) - x * L(C_i)) \tag{5}$$

If $f_{max} > f_{cur}$ then $x * L(C_z) - x * L(C_i)$ should be positive which says that $L(C_z) > L(C_i)$ which is contradicting our assumption, so f_{cur} should be maximum. □

Theorem 2. *If cycles in G_{ImBal} are saturated preferring the longest cycle first, then the objective function 3 will be maximized.*

Proof. G_{ImBal} contains flow-dependent cycles, defined in III.2 and/or flow-independent cycles.

Case 1: For flow-independent cycles, i.e. the cycles which don't share any edge or don't affect the maximum amount transferred through each other, we can saturate them in any order. So, preferring the longest cycle will also result in maximizing the objective function in 3.

Case 2: For flow-dependent cycles, we can see from theorem 1 that preferring the longest cycle will give the maximum value of equation 3. □

We can see that figure 4a gives the minimum value of I. We can see from theorem 2 that this is achieved if the node prefers to saturate the longest cycle over other. But, because finding the longest path is an NP-Complete problem, we propose an heuristic based re-balancing approach, REBAL.

B. REBAL Design

The idea is that every node selects the shortest cycle for each channel it belongs to and initiates the re-balancing request. This is because a standard polynomial-time algorithm finds the shortest cycle that consumes fewer resources than finding the longest cycle. Also, shorter cycles merge to form the longer cycle. For example in figure 3, consider node A and E sends the re-balancing request for amount 3, 5 by selecting the shortest cycle which is C_1:A-B-C-D-A, C_2:E-D-C-E respectively. Node D receives the re-balancing request through C_1 (R_{C_1}) and through C_2 (R_{C_2}) for channel CD. Note that R_{C_2} will increase its imbalance by 3 while R_{C_1} will decrease its imbalance by 5, i.e., the overall decrease in imbalance for channel CD is 2. So it will serve both R_{C_1} and R_{C_2}. After processing both requests, the resulting imbalance graph will be the same if we follow either approach (i.e., longest cycle first or REBAL). The decision on serving or denial of re-balancing request depends on the time instance when it arrives. If R_{C_1} and R_{C_2} arrives Δ (time to complete a REBAL request) apart then node D will process R_{C_2} and it may reject R_{C_1} based on new value of $idealBal_{CD}$ calculated using eq 2. Note that each node may receive many re-balancing requests. The node prefers to serve (saturate) the one for the longest cycle first. For example in figure 3 if there exist one more cycle C_3:D-F-G-H-I-D then node D first saturate C_3 and then C_1.

The algorithm runs in two phases: 1) find the maximum amount amt_{max} that can be transferred along the cycle 2) Sends re-balancing requests with amt_{max}.

find the maximum amount amt_{max} : Each node initiate re-balancing request for all its outgoing edges in G_{ImBal}. Let amt be the amount a node i needs to balance its channel, say

Ch_{out}. i finds the shortest cycle, say C_r for the channel Ch_{out} which has some channel Ch_{in} into i. It then sends a re-balance request <$+amt$, $cycle_{id}$> through C_{out} and <$-amt$, $cycle_{id}$> through C_{in}. Here, amt is calculated using equation 2 and $cycle_{id}$ of C_r is calculated by appending node's public keys involved in the cycle.

On the arrival of a request <$+amt$, $cycle_{id}$> at node i, it checks if it can send amt through the next channel in C_r. Similarly, for <$-amt$, $cycle_{id}$>, node i checks whether it can receive amt through the previous channel in C_r. If not, i rejects the request and drops it, else forwards it to the next or previous node in C_r correspondingly (lines 32-44 in algorithm 1). If amt requested is in the direction of edge in C_r then it is added to the list of positive amounts, else to that of negative amounts (line 7). Note that many nodes along a cycle will initiate the re-balancing request while the one with the minimum amt among all, which is the maximum amount that can be transferred along the cycle, will get served finally. For example, in figure 3 if nodes B and D initiate a re-balancing request for amt 3 and 10 respectively, then only D's request will go through because B's request is dropped at node A and/or D.

On arrival of re-balancing request <$\pm amt, cycle_{id}$>, a node signs and forwards it if it can be served, else drops it. When request returns back to the source node i, it must be signed by all the nodes along the cycle and they lock the amt for the time Δ. Locking the amount stop the routers from processing the new transactions which are not already considered while calculating the imbalance factor in 1. Note that node will lock the amount only after signing both $+amt$ as well as $-amt$ message for particular $cycle_{id}$.

Sends re-balancing requests with amt_{max} : After getting the agreement on the maximum amount that can be transferred along the cycle, i will send the re-balancing transaction tx_{rebal} to itself through cycle C_r. tx_{rebal} is similar to the normal transaction, except it reveals the source (and hence destination too). Further, it includes the request message reference.

On arrival of tx_{rebal}, the node verifies the re-balancing amount and the signature using the request message reference included in it. If a node hasn't verified the respective message and/or amt doesn't match, then it drops tx_{rebal} by reverting to the source with an "invalid transaction" message, else forwards it. Note that nodes don't charge a fee to forward tx_{rebal} with the assumption that they are going to benefit eventually from fees coming from transactions that will succeed as a result of re-balancing.

C. Re-routing at an intermediate node

and λ before time interval Δ, as Θ is the predicted value. In which case the node holds off on processing the new transactions (not considered in Θ) and send the error message *insufficient balance* back to the sender. To avoid this REBAL allows an intermediate node to forward the transaction to next hop through an alternative route. and λ before time interval Δ, as Θ is the predicted value. In which case the node holds off on processing the new transactions (not considered in Θ) and send the error message *insufficient balance* back to the sender. To avoid this REBAL allows an intermediate node to forward

Algorithm 1 CycleSelection

```
1:  FinalTxList ← {}
2:  On Event<Time expire and/or threshold reached>:
3:  call SendReq()
4:
5:  On Event<Receive mList>:
6:  segregate msg in mList to mList_ch of channel ch
7:  call OnReceive(mList_ch)
8:
9:  procedure FINDSHORTESTCYCLE(ch)
10:     return shortest cycle c involving ch
11: end procedure
12:
13: procedure SENDREQ
14:     for each ch in channels do
15:         c ← call findShortestCycle(ch)
16:         Send the re-balancing req < ±amt, c >
17:     end for
18: end procedure
19:
20: procedure ONRECEIVE(mList)
21:     calculate idealBal using equation 2
22:     edgeBal ← idealBal
23:
24:     for each msg in mList do
25:         c = msg− > c
26:         if msg− > amt ≥ next edge weight in c then
27:             remove msg from mList
28:         end if
29:     end for
30:
31:     Divide mList into two parts pList and nList
32:     sort nList and pList based on cycle length
33:     reverse nList and pList
34:     pSum, nSum ← sum(pList), sum(nList)
35:
36:     while nList or pList is non-empty do
37:         while idealBal − pList− > next.amt is +ve do
38:             add pList− > next to FinalTxList
39:             pList ← (pList− > next)
40:             idealBal = idealBal − pList− > next.amt
41:         end while
42:
43:         while idealBal+nList− > next.amt < edgeBal
    do
44:             add nList− > next to FinalTxList
45:             nList ← (nList− > next)
46:             idealBal = idealBal + nList− > next.amt
47:         end while
48:     end while
49:     return FinalTxList
50: end procedure
```

the transaction to next hop through an alternative route.

REBAL **Host:** Each sender (host) finds a route to the destination while generating the transaction. Each generated transaction is bounded with a timeout value, and it will be treated as a failure if it fails to reach the destination by then. Each receiving host confirms the receipt by sending an acknowledgment to the source.

In REBAL, the source node/host specifies the maximum timeout value (cltv) that a transaction will be valid for and the maximum amount of fee it can offer to the intermediate routers to route the transaction.

REBAL **Router:** Each routing node maintains a queue of transactions which can't be forwarded immediately (not even through an alternative route). It drops the transaction if the queue is full and sends back a negative acknowledgement.

In case an intermediate node runs out of funds to route the tx to the next hop, unlike LND, REBAL finds an another route to the next hop and sends tx through it. There might be more than one path between a router and next hop, it selects the one through the channel with a lowest imbalance factor.

Each intermediate router along the transaction path monitors the timeout value of the incoming transaction and sends a negative acknowledgment if it violates the timeout. Also, intermediate routers on the path deduct their routing fee and forward the remaining amount. An intermediate router denies (to forward) the transaction if the total fee exceeds the maximum amount of fee offered by the source.

IV. Evaluation

We now present results of a comprehensive evaluation of REBAL.

A. Experimental setup

We simulated REBAL using OMNeT++ (v5.6.2). We used the Spider [20] simulator to evaluate the state-of-art protocols listed below. Transactions in the simulation are implemented as messages passing between the nodes with simulated real-time delay [2] representative of a real network. All the experimental results reported use five independent runs.

Base cases We have considered four state-of-the-art routing algorithms for comparison:

- **LND** [17] : The baseline protocol that uses the path with minimum transaction fee and minimum hop count between the sender and receiver.
- **Landmark routing** [14], [19] : This algorithm works by assigning coordinates to nodes to find paths with reduced overhead between sender-landmark and landmark-receiver.
- **Spider and Waterfilling** [20]: It splits the transaction into smaller sub-units termed *mtu*. It also uses price variables to perform dynamic balance-aware routing. The waterfilling heuristic uses the path that has maximum available capacity.

Workload We have evaluated each strategy against the real-time Ripple workload downloaded on Nov 07, 2019 [22]. Transaction size distribution in real-time ripple workload has mean, median and largest value of 2438.79, 3.3, 880000 XRP (a unit of ripple currency).

Topology: We downloaded the LN topology data along with the respective channel capacities on Oct 03, 2020 [7]. The data consists of 5630 nodes with 34069 edges. Similar to Spider, we Snowball sampled [12] the topology data to reduce it to 206 nodes and 5468 edges to make if amenable for simulation without making it too simple We have converted the LND payment channel capacity to XRP to make it compatible with the Ripple real-time workload [6].

In addition, we simulated two synthetic topologies with 200 nodes each: a Watts-Strogatz small world topology [8] with 800 channels, and a Barabasi-Albert scale-free topology [1] with 1536 channels which follow the same channel capacity and delay distribution as that of LND topology.

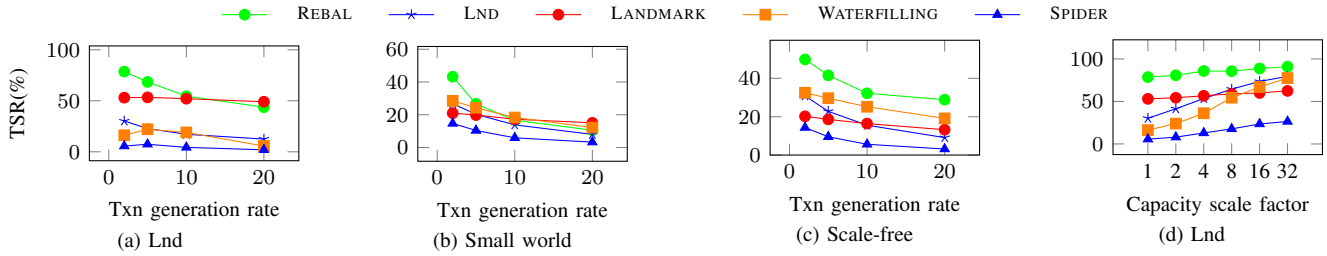
Metrics We report on four metrics- (1) Transaction success ratio (TSR), (2) Transaction success volume (TSV), (3) Transaction latency (of successful transactions) and (4) Throughput. We compare these metrics for different channel capacities (obtained by multiplying them with different constants termed as capacity scale factors) and transaction generation rate (transactions per sec per node).

B. Results

Parameter tuning: We observe that a single re-balancing operation takes around 2 seconds for completion ($\Delta = 2$). So ideally, it should happen once every 2 seconds. But frequent balance locking will decrease the TSR, hence is not advisable. Figure 7c depict that, maximum value of TSR is obtained when the re-balancing rate (R) lies between 1/40-1/45 per second. A lower value of the re-balancing threshold (Thr) will reject re-balancing requests from other nodes. On the contrary, for high Thr node will serve the re-balancing request at the cost of increasing imbalance in its own channel. Figure 7d depicts that maximum TSR is obtained when the threshold is between 0.4-0.5. So for further experiments we have set $R = 1/40$, $Thr = 0.4$, $\Delta = 2$ sec and transaction timeout value is set to 5 seconds.

Effect of transaction rate: We can see from figure 5a that even for 10x more transaction rate (i.e. from 2 to 20 transactions per sec per node) REBAL performs much better compared to other routing strategies for LND topology. Landmark routing has 5% more TSR than REBAL however, REBAL processes a larger transaction volume (figure 6b). This is the result of locking up a large portion of channel balances, for re-balancing, resulting in insufficient funds for further transactions in REBAL. Figure 6b shows similar behaviour as that of success ratio for transaction volume but the extent of decrease is more compared to the transaction rate. This is because of the increase in number of high-valued transaction failure for higher transaction generation rate.

From figures 5b and 5c, we can see that REBAL outperforms all the state-of-the-art protocols in terms of TSR. For the small world network, TSR of REBAL degrades because of the lower number of edges for which both REBAL and actual transaction processing compete, hence REBAL is not



(a) Lnd (b) Small world (c) Scale-free (d) Lnd

Figure 5: a), b), c) Variation of TSR with transaction generation rate (capacity scale factor = 1) & d) Variation of TSR with Capacity scale factor (transaction generation rate = 2 tx per sec per node)

able to find an alternative path to next hop. Similar behavior is observed for TSV in case of synthetic topologies (results are omitted from current imprint because of space constraint).

The throughput decreases with an increase in transaction generation rate because of a decrease in TSR, but it still outperforms all other protocols, as depicted in Figure 6d. REBAL experienced a slightly higher latency, specifically for higher transaction generation rate, because of locking some balance at the intermediate nodes as a part of re-balancing, as depicted in Figure 7b. Note that transaction latency is calculated for successful transactions only. A spike in transaction latency at a transaction generation rate of 5, as depicted in figure 7b, is because of an increase in waiting time at the queue.

Impact of channel capacity: Figure 5d shows that REBAL outperforms all the baseline protocols. But, REBAL has 7% lower TSV than LND for higher scale factor (above 8) as depicted in figure 6a. This is because REBAL locks some funds for the re-balancing process - consequently, it processes more smaller value transactions and higher value transactions timeout at the intermediate node while re-balancing completes. We also observe similar behavior for synthetic topology (results are omitted from current imprint because of space constraint).

Figure 6c shows that REBAL can process around 70 more transactions (per sec) compared to LND for the scale factor of 1. Also, it outperforms all the baseline protocols for higher scale factors. REBAL experienced negligible increase in the latency compared to the LND with the increase in the capacity scale factor. In fact, for the higher scale factor, it decreases a bit because of the availability of the shortest path as a result of re-balancing, as depicted in figure 7a.

In summary, we observe that REBAL outperforms all the baseline protocols at the cost of negligible increase in the transaction latency compared to LND.

V. Related Work

We broadly categorize the related work in two parts: a) Routing b) Re-balancing algorithms

Routing strategies: Flare [18] uses the beacon node whose role is to enhance the other nodes' visibility of the network and help them in route computation. Landmark routing approach, like SilentWhispers [14], [19], have highly connected nodes as landmarks. All transactions are routed from source to landmark and then from landmark to destination. However, both beacon and landmark based routing create centralization and are prone to targeted attacks. In Speedy Murmurs [19], each node is associated with a vector such that the hop distance between two nodes is reflected by their distance in the vector space. Ant routing [11] obfuscates the topology of the network and treats each node equally. All the above algorithms don't take into account the dynamic channel balances thus leading to poor performance. Spider [20] considers channels' balance for route selection. It uses a packet switched architecture, where payments are broken down into multiple smaller payments. This leads to increase in latency and bandwidth. [10] proposes the fee mechanism and uses that to make the routing decisions which keep the channels balanced. Flash [22] focuses mainly on increasing success-volume by breaking high value transactions into multiple smaller transactions, but doesn't provide significant improvement in success ratio.

Re-balancing: Revive [13] proposes a centralized re-balancing solution that compromises the privacy of channel balances. [16] tries to make an equal contribution of the node's funds to all its channels which doesn't work for all the scenarios (specifically the case where most of the payment are uni-directional, i.e., from payer to merchants which is the case with production Lightning Network). [9] proposes active and passive re-balancing. Active re-balancing sends loop-back transaction from its high balance channel to its low balance channel while passive re-balancing charges the transaction fee inversely proportional to channel balance in the direction of payment.

VI. Conclusions & Future Work

This paper presents a novel channel re-balancing scheme for PCNs that outperforms state-of-the-art algorithms on real workloads without any workload assumption and compromising privacy.

There are 3 possible directions for future work - i) In this paper, we haven't evaluated any possible attacks on the REBAL scheme. Future work might include a thorough analysis of possible attacks and their impacts. ii) For REBAL, we have used current and future transaction load as a parameter to decide when to trigger re-balancing. The estimation of the future load can be improved using prediction models, which can lead to better trigger decisions. iii) Atomic Multi-path Payments (AMP) allows a transaction to be split in multiple smaller transactions, such that the original transaction

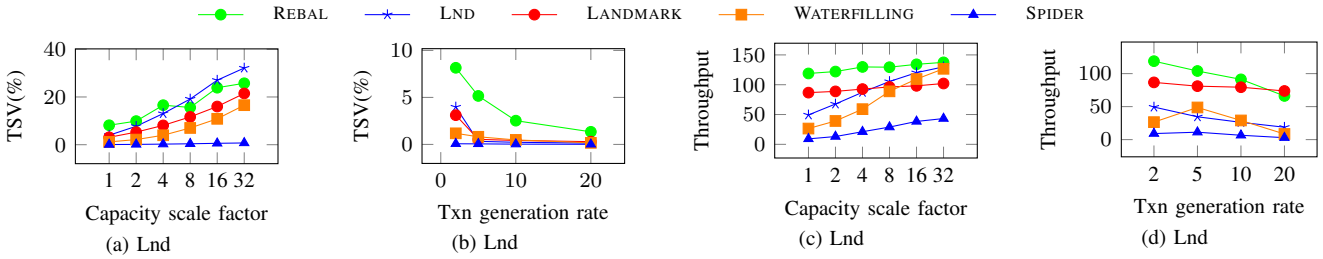


Figure 6: a) & b) Variation of transaction success volume with capacity scale factor (transaction generation rate = 2 tx per sec per node) and transaction generation rate (capacity scale factor = 1), c) & d)Variation of transaction throughput with capacity scale factor (transaction generation rate = 2 tx per sec per node) and transaction generation rate (capacity scale factor = 1)

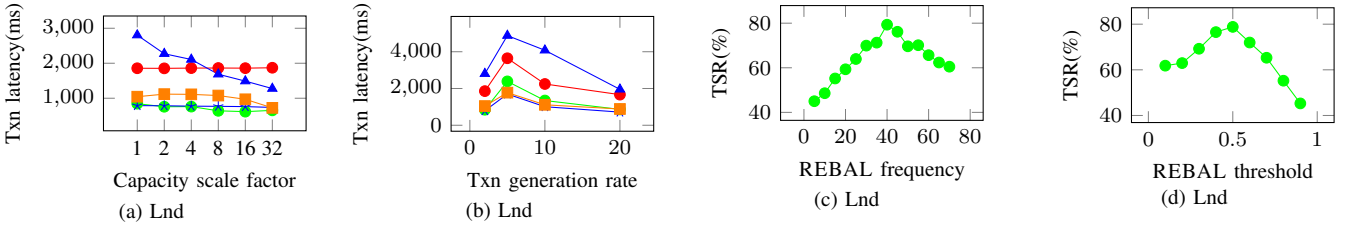


Figure 7: a) & b) Variation of latency with capacity scale factor (transaction generation rate = 2 tx per sec per node) and transaction generation rate (capacity scale factor = 1), c) & d) Behavior of transaction success ratio with REBAL frequency (1/x per sec) and REBAL threshold

is only considered complete if all smaller transactions are complete. Performance of REBAL in combination with AMP needs to be investigated.

REFERENCES

[1] "Barabasi albert graph." [Online]. Available: https://networkx.github.io/documentation/networkx-1.9.1/reference/generated/networkx.generators.random$_{g}$raphs.barabasi$_{a}$lbert$_{g}$raph.html

[2] "Global ping latency," [Online; accessed 16-May-2019]. [Online]. Available: https://investoon.com/mining$_{p}$ools/eth

[3] "Raiden network." [Online]. Available: https://raiden.network/

[4] "Real-time lightning network statistics." [Online]. Available: https://1ml.com/statistics

[5] "Ripple." [Online]. Available: https://ripple.com/

[6] "Ripple real time wrokload." [Online]. Available: https://github.com/NetX-lab/Offchain-routing-traces-and-code/tree/master/sim/traces/ripple

[7] "Spider simulator." [Online]. Available: https://github.com/spider-pcn/spider$_{o}$mnet

[8] "Watts strogatz graph." [Online]. Available: https://networkx.github.io/documentation/networkx-1.9/reference/generated/networkx.generators.randomgraphs.wattsstrogatzgraph.html

[9] M. Conoscenti, A. Vetrò, and J. C. De Martin, "Hubs, rebalancing and service providers in the lightning network," *IEEE Access*, vol. 7, pp. 132 828–132 840, 2019.

[10] G. Di Stasi, S. Avallone, R. Canonico, and G. Ventre, "Routing payments on the lightning network," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 1161–1170.

[11] C. Grunspan, G. Lehéricy, and R. Pérez-Marco, "Ant routing scalability for the lightning network," 2020.

[12] P. Hu and W. C. Lau, "A survey and taxonomy of graph sampling," *CoRR*, vol. abs/1308.5865, 2013. [Online]. Available: http://arxiv.org/abs/1308.5865

[13] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 439–453. [Online]. Available: https://doi.org/10.1145/3133956.3134033

[14] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Silentwhispers: Enforcing security and privacy in decentralized credit networks," in *Proceedings of the 24th Annual Symposium on Network and Distributed System Security (NDSS '17)*, February 2017, pub_id: 1148 Bibtex: MaMoKaMa_17:silentwhispers URL date: None. [Online]. Available: https://publications.cispa.saarland/898/

[15] A. Miller, I. Bentov, R. Kumaresan, and P. McCorry, "Sprites: Payment channels that go faster than lightning," *CoRR*, vol. abs/1702.05812, 2017. [Online]. Available: http://arxiv.org/abs/1702.05812

[16] R. Pickhardt and M. Nowostawski, "Imbalance measure and proactive channel rebalancing algorithm for the lightning network," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2020, pp. 1–5.

[17] J. Poon and T. Dryja, "The bitcoin lightning network:scalable off-chain instant payment," 2016. [Online]. Available: https://lightning.network/lightning-network-paper.pdf

[18] P. Prihodko, S. Zhigulin, M. Sahno, A. Ostrovskiy, and O. Osuntokun, "Flare : An approach to routing in lightning network white paper," 2016.

[19] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," 2017.

[20] V. Sivaraman, S. B. Venkatakrishnan, M. Alizadeh, G. Fanti, and P. Viswanath, "Routing cryptocurrency with the spider network," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, ser. HotNets '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 29–35. [Online]. Available: https://doi.org/10.1145/3286062.3286067

[21] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *Financial Cryptography and Data Security*, R. Böhme and T. Okamoto, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 507–527.

[22] P. Wang, H. Xu, X. Jin, and T. Wang, "Flash: Efficient dynamic routing for offchain networks," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, ser. CoNEXT '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3359989.3365411

[23] Éva Tardos, "A strongly polynomial minimum cost circulation algorithm," *Springer Access*, vol. 5, p. 247–255, 1985.