

Balance Transfers and Bailouts in Credit Networks using Blockchains*

Lalitha Muthu Subramanian, Roopa Vishwanathan, Kartick Kolachala
Department of Computer Science, New Mexico State University, USA
{lalitha,roopav,kart1712}@nmsu.edu

Abstract—In this paper, we propose a technique for rebalancing link weights in decentralized credit networks. Credit networks are peer-to-peer trust-based networks that enable fast and inexpensive cross-currency transactions compared to traditional bank wire transfers, which has led to their increasing popularity and use. Although researchers have studied security of transactions and privacy of users of such networks, and have invested significant efforts into designing efficient routing algorithms for credit networks, comparatively little work has been done in the area of *replenishing* credit links of users in the network. Replenishing links at regular intervals in a credit network is important to keep users solvent, the network viable with enough liquidity, and to prevent transaction failures. This is achieved by a process called *rebalancing* that enables a poorly funded user to create incoming as well as outgoing credit links.

We propose a system where a user with zero or no link weights can create incoming links with existing, trusted users in the network, in a procedure we call *balance transfer*, followed by creating outgoing links to existing or new users that would like to join the network, a process we call *bailout*. Both these processes together constitute our proposed rebalancing mechanism. Our techniques would also serve to make the network more competitive by offering users lower rates of interest, and enable users to earn routing fees-based revenue by participating in high throughput transaction paths.

Index Terms—rebalancing, blockchain, payment networks

I. INTRODUCTION

Blockchain and cryptocurrencies such as Bitcoin [1] have disrupted the banking industry, enabled new business models, and helped in designing new, efficient financial infrastructure. A blockchain is an append-only distributed ledger, where users post messages or transactions, which are usually considered immutable. Blockchains have enabled the growth of IOU (I Owe You) credit networks in recent years. A credit network is a decentralized peer-to-peer lending network, where users lend out financial credit based on social trust. Credit networks provide the means to do path-based payments between two users, where the payment is routed through multiple intermediate users. Credit networks offer several advantages over traditional banks, such as low end-to-end transaction time, lower routing fees, and the ability to perform cross-currency transactions in the order of seconds. Some examples of real-world credit networks are Ripple [2] and Stellar [3].

Credit networks are usually modeled as a directed graph with vertices and weighted edges/links. The vertices represent the various users in the system and the weight on a link is the credit a user is willing to offer to an adjacent user. The directionality of the link is used to denote the lender and borrower, e.g., $u \xrightarrow{20} v$ denotes v has extended 20 units of credit to u . Payments are routed along (and in the direction of) credit links, and once a payment is made from a sender to receiver, the link weights are decremented along the transaction path.

There has been growing interest in finding solutions for efficient routing, and enabling private and secure transactions in credit networks [4, 5, 6] but not much work has been done in the area of *rebalancing* link weights of a user that has run out of credit, and cannot participate in transactions. Rebalancing in credit networks is a significant problem to study, since, if the credit on a given link gets exhausted, i.e., if the weight on a link connecting two nodes drops to zero, no transactions can be done on any path containing those nodes until the link is refunded, a process which involves expensive on-chain transactions. Such nodes will be unable to participate in any transaction due to their inability to transfer or flow money, and will eventually be shunned by other nodes in the network. Networks which have large sections of such dormant nodes will eventually become inefficient, have progressively low throughput and may not even remain operational. In this paper, we study the problem of rebalancing links in an efficient way, while making minimal use of the blockchain, with the goal of avoiding mining and blockchain write fees.

We propose a two-step rebalancing process wherein a node whose link weights are low or close to zero, can create fresh incoming links in a process called *balance transfer*, and then create fresh outgoing links in a process called as *bailout*. Upon completion of these two steps, the poorly connected node will become an active participant in the network, become involved in high throughput transactions, thus enabling it to collect routing fees. At a high level, balance transfer involves existing nodes moving outstanding debt to a new lender, incentivized by lower interest rates, and bailout involves the lender temporarily being assisted by a bank, who infuses capital into them to shore up their credit reserves. Once their earnings exceed their debts, the bank exits the system.

Balance Transfer in Credit Networks: Balance transfer in the real world occurs when the outstanding balance of one credit card (or several credit cards) is moved to another credit card account. This is often done by consumers looking for

*Research supported by NSF award #1800088. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

lower interest rates. Many credit card issuers offer introductory balance transfer APRs that are lower than the standard rates. Another advantage of balance transfer is that it makes financial management easier by transferring consolidated balances to a new credit provider. Although lending in blockchain has become quite common, the idea of doing balance transfers in credit networks has not been explored. In this paper we design a system where any lending node with low connectivity can advertise a lower interest rate, and thus gain more borrowers.

Bailout in Credit Networks: A bailout is a process where an organization or a government injects capital into a failing business to save it from bankruptcy, and to help make the business competitive again. In the context of credit networks, a bank does a “bailout” of a user, Alice, with no outgoing credit links by lending credit to her, and temporarily helps her by connecting her with other users with whom Alice can establish permanent links. Once she establishes permanent links with other users, the bank exits the network, possibly after collecting a small fee from Alice.

Our Contributions: In this paper, we give a new approach for rebalancing depleted credit links in credit networks.

1) We propose a two-step approach for rebalancing consisting of *balance transfer* and *bailout*. In the balance transfer step, a poorly connected node, called as *requestor*, will establish incoming links with other nodes in the network by advertising a lower interest rate. This enables the requestor to become a lender to other nodes. In the bailout step, a well-known party such as a bank will infuse capital into the requestor node, by helping it connect to, and establish outgoing links with several other nodes in the network. After the requestor node establishes outgoing connections with other nodes, the bank will leave the network, possibly after collecting a fee from the requestor. At the end of this process, the requestor node will have several incoming and outgoing links, which will enable it to help facilitate several transactions, thus increasing the overall throughput and robustness of the credit network.

2) Since the performance of our balance transfer step is highly dependent on being able to find routes efficiently, we compare and analyze two different routing algorithms for doing balance transfers: Chord [7] and VOUTE [8], and evaluate their performance experimentally.

II. RELATED WORK

In this section, we review literature on credit networks, payment channels, and rebalancing and loaning in credit networks.

Credit networks: Fulgor and Rayo [9] were the first to setup a peer-to-peer payment channel network that provides provable security and privacy properties, with Rayo being the first payment network that enforces non-blocking transactions. Fulgor and Rayo, both, establish a path between sender and receiver, assuming all users in the path to be honest, and users have at least partial knowledge about network topology. Unlike Fulgor and Rayo, in our system, the entire network topology is not known to the users and we do not assume all the users

in the path to be honest. Also, we do not propose any payment operations, instead focus only on rebalancing credit links.

SilentWhispers [5] presents a decentralized credit network (DCN) architecture which consists of subsets of paths between the sender and receiver calculated via several trusted entities called *landmarks*. PathShuffle [10] presents a path mixing protocol for Ripple network providing complete anonymity. PathShuffle leverages existing infrastructure from Ripple [2] to create and maintain consistent credit links with users in the networks. Both [5] and [10] present solutions for routing payment in a secure and privacy-preserving way in credit networks, and neither tackle the challenge of rebalancing depleted credit links. Our goal is to design a mechanism for rebalancing credit links.

We leverage the concept of *landmark* nodes proposed in [5] in our system to assist the requestor node in our bailout phase. The landmark node is a well connected node such as a bank, and hence, can potentially help the requestor node contact several other nodes in the network for establishing outgoing links. We use the landmarks in our system to assist the requestor node establish outgoing links without placing enormous trust on landmarks, unlike [5]. Roos et al [11] used graph embedding for efficient routing with concurrent transactions overcoming some inefficiencies in [5]. They too do not focus on the rebalancing problem. Malavolta et al [12], recently proposed a novel linkable ring signature scheme for refund transactions natively in Monero [13] and extend the same scheme into having scalable off-chain transactions by establishing payment channels using Monero. Our system could be deployed in such kind of payment channel network to enable the system to be more competitive and enhance the connectivity in such channels simultaneously achieving rebalancing within the network.

Panwar et al. [6] proposed a DCN system where users can perform path-based transactions that preserves sender, receiver and value privacy but involves a significant number of blockchain writes in the course of a normal, successful transaction (more in the case of transaction rollbacks, re-tries, and other edge cases). Our system also involves blockchain writes, but in our system, a single blockchain write is done only after the completion and execution of the entire rebalancing protocol, unlike [6], which would help it scale much better.

Rebalancing: REVIVE [14] is a payment network that allows users to rebalance their channel without having to communicate with the blockchain. Although very efficient in rebalancing bidirectional networks (i.e., cyclic networks), REVIVE does not present any solution for rebalancing in a unidirectional (acyclic) credit network, which is our use-case scenario. REVIVE has a leader elected in the network who stays online all the time in order to facilitate rebalancing requests. We do not place trust in any leader or centralized entity to establish new incoming or outgoing links in the system although we make use of landmark nodes to assist the requestor node *temporarily* in our bailout step. Lightning network [15] is a highly scalable payment channel network

that is constructed on top of Bitcoin. Lightning network does re-balancing off-chain, but again only for *cyclic* networks. To rebalance a credit link, a node does a payment in cyclic path to itself and such payments usually comes with a fee for each node in the circular path. In our approach, a node establishes new incoming and outgoing credit links, without having to pay every node in a payment path.

Ripio Credit Network (RCN) [16] is a peer-to-peer global credit network protocol based on cosigned smart contracts and blockchain technology. A user can join as a lender, cosigner or borrower in the network. A cosigner will act as a go-between a borrower and a lender in the network. In case a borrower defaults, the cosigner works out an alternative mechanism for managing debt collection. However, this places a great responsibility on the cosigner, and if they are incapable of actually enforcing debt collection for any reason, the overall value of the network would decrease.

III. SYSTEM DESIGN

Credit networks are usually dense networks, e.g., Ripple [2], with several incoming and outgoing links from the nodes. If a node has depleted credit links, then, intuitively, one way for it to rebalance its links would be to extend credit to, and borrow from new users. This could be problematic for several reasons: the new users might not be trusted, or at the moment when a node's credit links are depleted there might not be enough new users in the system. Ultimately, whether to lend or borrow from a user, we believe, should be a matter of choice, and no new node should be compelled to accept credit from, or lend credit to an existing user, simply because the existing user needs to rebalance their credit links. With this design goal in mind, we introduce the concept of balance transfer and bailout where any existing node can transfer its credit links to a new lender who offers a lower rate of interest, and an existing use can actively look for credit lenders in the system, with some help from a partially-online trusted bank. We now give an overview of the two steps that comprise our system.

Balance transfer: Figure 1 shows how the balance transfer process takes place in a credit network. Here any user can disconnect from an existing lender and transfer credit links to a new lender node offering a lower rate of interest.

In Figure 1 Part a) denotes a simple credit network where A is a highly connected node. D is a poorly connected node and would need to send requests to establish new incoming connections. To this end, D first raises a request to add incoming nodes, by advertising a lower rate of interest, which will incentivize some of the nodes in the network to transfer their credits from their existing lender to D . It is important to note here that, in our design, every node that wants to transfer to D would have to transfer their full credit that exists with the current lender, and nodes are not allowed to establish an incoming link to D with partial amount. This closely models the real-world balance transfer mechanisms where a user either transfers their debts (to a new lender) in the debt's entirety, or not at all, but cannot partially transfer debts. Figure 1 Part b) depicts the network after two of A 's borrowers, E, G have

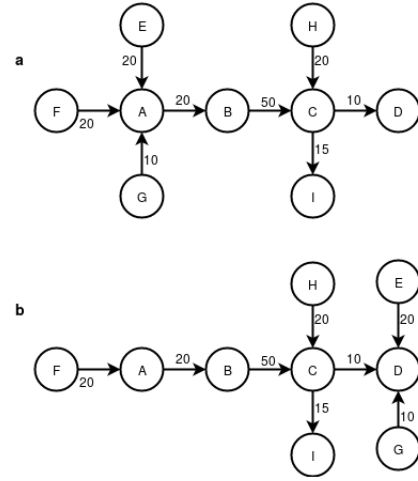


Fig. 1: Balance transfer in credit networks

voluntarily transferred their balances to node D , after severing their links with A , thus D has established several incoming links. At this point, all affected nodes will locally store their new links and link weights. After the bailout step, the changed network topology will be written to the blockchain.

Bailout: In the bailout process, a trusted, highly connected party such as a bank, or a credit union *temporarily* lends credit to node D , so that D can establish outgoing connections. We refer to this trusted party as a *landmark*, or *LM*. The high-level idea is that *LM* will use the fact that it is highly connected, and temporarily connect D with several other nodes in the network with whom *LM* has a direct connection. D will then request each of these nodes if they would like to lend credit to D , thus establishing outgoing links from D to them. Note that any or all nodes can decline D 's request, at which point *LM* will connect D with a fresh set of nodes.

Figure 2 Part a) shows the bailout phase where the link between *LM* and other nodes are established after the balance transfer state. At this point, the poorly connected node D has two incoming links after the execution of the balance transfer step. In order to make D an active participant in transactions in the credit network, D also needs outgoing links. After the balance transfer scenario, node D sends request to the highly connected node, *LM*. *LM* provides a list of nodes, F, A, B, I, C, H, G , that can potentially establish a outgoing link with D , i.e., willing to lend credit to D . D will then send request to all of these nodes for an outgoing link. If none of the neighboring nodes are willing, *LM* helps D to establish an outgoing link with any of the newer nodes that joined the system, or will give D a fresh list of nodes. Figure 2 Part b) shows the network after nodes A and B agree to D 's request for outgoing links, D establishes outgoing links with A, B , at which point *LM* exits the network.

IV. ADVERSARIAL MODEL

In our system, we assume that any adversary can corrupt a single or a set of users in the network. The corrupted user(s) can be either the requester, who raises a rebalancing request,

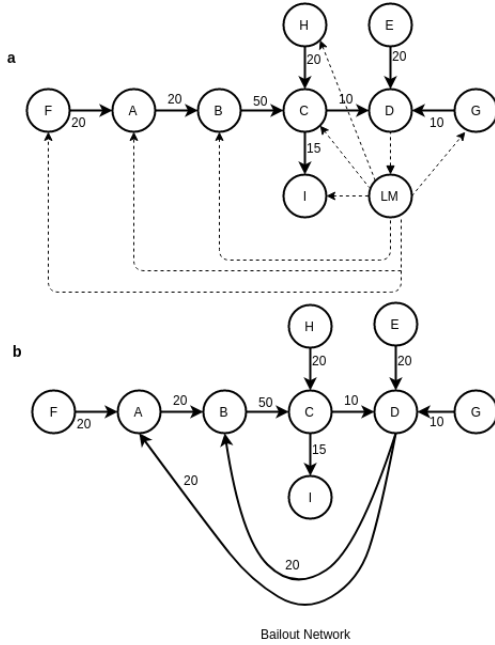


Fig. 2: Figure showing before and after scenario of the bailout step.

the nodes that respond to the request, or any intermediate node. During the bailout phase, we need a trusted landmark, LM , who is temporarily assists the requestor node. We assume the adversary cannot corrupt the LM node. Each user i has her own signing and verification key pair (sk_i, vk_i) and an encryption, decryption key pair (pk_i, dk_i) . Once any user is corrupted, their corresponding signing and verification keys are compromised, the adversary can misreport the credit link between a user and her neighbor, not respond to any request, respond selectively to requests and relay fraudulent balance transfer requests to its neighbors. We assume that an adversary cannot corrupt all users in the network, and thus may know partial network topology, but does not know the entire network. We now give the desired security/privacy properties of a payment network that enables balance transfers and bailouts.

A. Privacy and Security properties

Link privacy: Link privacy is achieved when an adversary only knows the value of links adjacent to her and will not have access to other nodes' links, even if they were part of the balance transfer or bailout steps.

Corrupted users: We now discuss which users could possibly get corrupted, what can a corrupted user do, and how to mitigate the situation.

1) *Corrupt balance transfer requestor node:* Any user who sends the balance transfer request and can act maliciously, by claiming to have committed to one value (credit limit) in the request tuple, and later renege on their commitment. The requestor node could also refuse to provide a low rate of interest, as advertised.

2) *Corrupt responder node:* Any node that responds to the balance transfer request can act maliciously by claiming to

have responded with a different amount than the actual amount linked to the request id.

3) *Corrupt intermediaries:* Any user along the route from requestor to responder can be a malicious. Such adversarial users can either drop messages with or without making an entry into their logs, or mis-direct messages to other collaborating nodes in the network.

Accountability: Any malicious user should not be able to misreport her link value. In our system, each user maintains a record of their link weights with their next-hop neighbors in a local hash table. Each user involved in a rebalance transaction also holds signed contracts containing the current and updated link weights as a proof of link weight update. In case where any user behaves maliciously, the honest peers would be able to detect such malicious activity, also third-party arbiters can adjudicate based on the signed contracts. An arbitrator can be any law enforcing authority who on receiving a complaint can enforce legal punitive action according to local laws, or revoke nodes' access to network. The exact nature of the remedial action taken by arbiters is beyond the scope of this paper.

V. ROUTING IN BALANCE TRANSFERS

In this section we discuss two different routing protocols we use for balance transfers in credit networks: prefix embedding [17] and Chord [7], and analyze and compare their efficiency. Note that these routing algorithms are not used for routing payments among nodes in our system, rather only for doing a balance transfer.

A. Routing using Prefix Embedding

The first part of the balance transfer algorithm is the *Find Route* phase in which the responder node finds a route from itself to the requester. We make use of prefix embedding [17] and VOUTE [8] to establish a route after which rebalancing occurs. Prefix based embeddings are a part of greedy embeddings [17] [18]. They are, in general, created by embedding a spanning tree into a suitable metric space. An ID is assigned to the root node, and the tree consists of several child nodes where each child computes the ID based on the ID of its parent node. Prefix embedding is an adaptation of PIE embedding for unweighted graphs. The idea is that, every node is assigned an ID using a custom metric space such that the node ID is equivalent to the hop distance or the depth of a spanning tree. A child's id is essentially the ID of the parent, an additional coordinate equal to the index of the child. The prefix embedding algorithm uses the following equation to find the length of the shortest path between node u and node v , where cpl is the common prefix length.

$$d(id(u), id(v)) = id(u) + id(v) - 2cpl(id(u), id(v)) \quad (1)$$

B. Routing using Chord

Chord [7] is a routing algorithm built using distributed hash tables for peer-to-peer networks, without any centralized monitoring authority. Chord uses a $\langle key, value \rangle$ pair to map to a specific node across the system. The keys are assigned to nodes using Consistent Hashing [19] across the network.

Consistent hashing in Chord reduces the load in the system, since each node in a network requires the same number of keys and requires little movement of keys when nodes leave or join the system, making the system dynamically compatible.

Chord assigns a $\langle key, value \rangle$ pair to each of the nodes in the system, where the key is the identifier using SHA-1 [20] and maps the keys to the nodes that are responsible for them. A peer identifier is chosen by hashing the data key. The length of the identifier is usually large to ensure the probability of keys hashing to the same identifier is negligible. The identifiers are arranged from 0 to $2^m - 1$, where m is the digest size of the hash function used. Key k is assigned to the first peer whose identifier is equal to or follows k in identifier place and the first peer, clockwise from k is called the successor peer of k , represented by $successor(k)$. When a peer n joins or leaves the system, the keys that previously belonged to n , is reassigned to n 's successor. This enables maintaining consistent hashing in the system. For helping users join and leave the Chord network, we run a *stabilization* protocol at regular intervals that updates the *finger table* stored at each node. The finger table (FT) stored at each node is a table containing the IDs of its successors. Due to space constraints, we do not elaborate further on the working of prefix embedding and Chord; below we briefly compare their efficiency in terms of routing efficiency and network restrictions.

C. Prefix Embedding vs. Chord

Routing efficiency: The main advantage using a Chord-based routing algorithm is that the number of hops to receiver is reduced, based on the density of the network. A peer leaving or joining the system does not involve too many changes to the key distribution to the system, although the successor pointers of some peers need to be changed. In prefix embedding, although, the users finds the shortest path, the worst case scenario for routing efficiency would be to find a route that traverses along the entire depth of the network.

Network Restrictions: A peer leaving or joining the system does not involve too many changes, although the successor pointers of some peers need to be changed. It is important to ensure that the successor pointers are up to date otherwise, the routing will fail in such a system. Hence there is a need to constantly update the finger table, as and when a change occurs. When peers fail, it is possible that a peer does not know its new successor, and that it has no chance to learn about it. Hence, the efficiency at which the finger tables are updated are $O(\log^2 N)$ (N is the number of nodes). In prefix embedding or VOUTE, the network need not monitor their nodes constantly and there is no maintenance cost incurred. The users can join and leave when they want making the network more adaptable and does not involve handing over keys, updating peers about leaving the system etc.

VI. CONSTRUCTION

In this section, we describe the construction of our system, comprising of the balance transfer and bailout steps. We first

present two different ways of doing the balance transfer step: using prefix embedding-based routing, and using Chord-based routing. Then we present our algorithms for the bailout step.

A. Balance Transfer using Prefix Embedding

In this step, a responder node responds to a balance transfer request BT broadcast by another node in the network. The requester node broadcasts the amount available for accommodating the incoming nodes and the rate of interest. Any user j who wants to transfer to user i , needs to find a route to i , and then initiate the balance transfer. This process is depicted in Algorithm 1.

In Algorithm 1, the parties involved are the requestor node, i , the responder node, j , and other intermediate nodes along the length of the response path. Node i raises a balance transfer request by broadcasting the tuple BT . The tuple consists of the amount that i can offer as credit, interest rate $intr$ and a response time tp until when i can accept responses from different nodes in the system (line 1,2). In line 3, j finds the length of the route path $|L|$ using prefix embedding, where $|L| = |\vec{id}_j| + |\vec{id}_i| - 2 * cpl(\vec{id}_j, \vec{id}_i)$. Node j finds the nearest node on the route, and computes hash of it's co-ordinates \vec{id}_j in line 5. This hashed co-ordinates are used in prefix embedding to find the next neighbor and is computed using the common prefixes between j and next node k . Node j computes \vec{id}_k in line 7, and creates a signature $Sign_{sk_j}(\vec{id}_k) \rightarrow \sigma_j$. Node j does a $j.write(\vec{id}_k)$ and $j.write(\sigma_j)$ in line 8 of this algorithm. By using hashed coordinates, the address of the users are not made public, and the actual addresses are only available to the next-hop neighbors on the path. Then, j sends a response $resp_j = \langle E_{pk_i}(amt_j) \rangle$ to k . In line 11-13, j waits to ensure that k does a $k.write$ of next address along the path, which ensures that the response is sent to the hashed address that is written to the shared hash-table between k and its neighbors. If j finds a malicious entry on the shared hash-table, j can choose to send the response $resp_j$ through a different route. Node i receives responses $resp_j$ in line 15 of this algorithm. Node i saves all responses in $i.resp[]$ if the responses are received within time tp and the amount in response $amt_j \leq amt$ in lines 16, 17. In line 19 of this algorithm, source node src is set to k , $user = j$, and $dest = i$. For every response that i receives, i verifies if updated $amt \neq 0$, and calls MultiSig algorithm in line 21 and produces contract $CtBal$. The MultiSig function creates shared, signed contracts between two adjacent nodes; we do not give the algorithm here due to space constraints. Once i initiates the MultiSig algorithm, node j pays the amount lent by k in line 22. Then, node j updates her hash-table after receiving $k.resp$ and deletes the hash-table entry with k . Node k deletes her entry in hash-table and they sign an acknowledgement of the update (line 24, 25), and produces $Sign_{sk_{src}}(ack) \rightarrow \sigma_{ack_{src}}$. The MultiSig operation returns $\{CtBal, \sigma_{dest}, \sigma_{user}\}$, which is written to blockchain by i in line 26 of this algorithm.

Algorithm 1: Balance Transfer Algorithm using Prefix Embedding

Initial State : Node i requesting for Balance transfer

Final Outcome: Nodes j establish a route to i for balance transfer.

```
1 begin
  /* Find Route Phase */
2   $i$  raises Balance Transfer request by broadcasting
   tuple  $BT = \langle amt, intr, tp \rangle$ 
3   $j$  the receiver computes
    $|L| = |\vec{id}_j| + |\vec{id}_i| - 2 * cpl(\vec{id}_j, \vec{id}_i)$ 
4  for  $j \in |L|$  do
5     $j$  computes hashed co-ordinates  $\vec{id}_j$  with
     padding added
6     $j$  finds the id of next node  $k$ , using common
     prefix coordinates
7     $j$  hashes  $\vec{id}_k$  and creates signature
      $Sign_{sk_j}(id_k) \rightarrow \sigma_j$ 
8     $j$  does  $j.write(id_k)$  and  $j.write(\sigma_j)$ 
9     $j$  creates response  $resp_j = \langle E_{pk_i}(amt_j) \rangle$  and
     sends to  $k$ 
10    $j$  waits for time  $t_j$ , verifies if node  $k$  does
      $k.write$ 
11   if  $k.write = "False"$  then
12      $j$  finds a different route until  $resp_j$  reaches
      $i$ 
13   end
14 end
  /* Balance Transfer Phase */
15 Node  $i$  receives  $resp_j$ 
16 if  $ts < tp$  and  $amt_j \leq amt$  then
17   Node  $i$  accepts and saves responses in  $i.resp[]$ 
18 end
19 Set  $src = k$ ,  $user = j$ ,  $dest = i$ 
20 for each  $user \in i.resp$  do
21    $dest$  node calls  $MultiSig(dest = i, val, user) \rightarrow (CtBal, \sigma_{dest}, \sigma_{user})$ 
22    $user$  pays back  $lw_{src, user}$  and creates
      $req(user, src)$  with  $lw_{src, user} = 0$ 
23    $src$  signs  $ack$  of  $lw_{src, user} = 0$ , produces
      $Sign_{sk_{src}}(ack) \rightarrow \sigma_{ack_{src}}$ 
24    $user$  verifies  $\sigma_{ack_{src}}$  and creates  $\sigma_{ack_{user}}$ 
25    $src$  and  $user$  deletes the hashtable entries on
      $src$  and  $user$  nodes respectively
26    $dest$  node writes  $\{CtBal, \sigma_{dest}, \sigma_{user}\}$  to
     blockchain
27 end
28 end
```

B. Balance Transfer Algorithm using Chord

In this section, we discuss the balance transfer process using Chord for routing, Algorithm 2. The first phase of this algorithm consists of the Find Route phase, where the user j , referred to as the responder node, responds to a BT request raised by a requestor node i . Node i raises a request in line 2 of this algorithm with a tuple $BT = \langle amt, intr, tp, key \rangle$, where, amt is the amount that is extended as credit, $intr$ is the rate of interest offered, tp is the time period within which the response is accepted and key which is used in finding the route to the user in a chord network. Node i broadcasts the request BT to all users in line 3. Node j responds to BT by first locating the key of i , in line 4. If node i is an immediate neighbor of j from its finger table FT_j , j responds with $resp_j = \langle E_{pk_i}(amt_j) \rangle$ in line 6. If node i is not among j 's immediate neighbors, j finds the nearest node k from FT_j and sends the response in line 9 of this algorithm. Every intermediate node k along the length of the path L , does a lookup(key) in their corresponding FT_k in line 11. Every node k does the steps 6 to 13 until the $resp_j$ reaches node i (requester). Node i receives all responses accepted within time tp and stores in $i.resp[]$ in line 16. In line 18, source node src is set to k , $user = j$, and $dest = i$. For every response that i saved, i verifies if updated $amt \neq 0$, calls the Multisig algorithm in line 20, produces contract $CtBal$. Each $user$ node in $i.resp[]$ then updates the link weight between $user$ and src to 0 in line 21. The src node signs ack of $lw_{src, user} = 0$, produces $Sign_{sk_{src}}(ack) \rightarrow \sigma_{ack_{src}}$ in line 22 of this algorithm, $user$ verifies $\sigma_{ack_{src}}$ and creates $\sigma_{ack_{user}}$ in line 23, and in line 24, the nodes src and $user$ delete their corresponding hash table entries. The $dest$ node writes $\{CtBal, \sigma_{dest}, \sigma_{user}\}$ to blockchain in line 25. Once the MultiSig operation is complete, node j repositions itself in the Chord ring after informing the successors and predecessors in the Chord FT in line 27 and 28. Every node from j 's previous position to new position, update their finger table by calling the $stabilize(FT)$ function to update their respective finger table entries. j also calls the $Stabilize$ function to update her finger table with new successors after relocating in line 32.

C. Bailout Phase

The bailout step is the final step in the rebalancing process and is given in Algorithm 3. Node i , in this algorithm requests for outgoing nodes through the LM node. The main aim of this algorithm is to connect i with multiple outgoing nodes successfully with the help of the landmark node (LM). In line 2 of this algorithm, node i sends a request to LM to connect or find m nodes that would ideally, like to establish an outgoing link with i . LM returns a list of j identities, where $j \in [1, 2, ..m]$. LM waits for a response time tr within which atleast one outgoing link is established and this is done in line 4. In line 6, i calls the $OutReq \rightarrow val$ to j through LM . In line 7 of this algorithm, if any j responds with a "val", then j calls the $MultiSig(dest = j, user = i, val) \rightarrow (CtBal, \sigma_{dest}, \sigma_{user})$ function. If all j nodes respond with a " \perp ", then LM responds with a new set of $j \in [v_1, v_2, v_3...v_m]$

Algorithm 2: Balance Transfer Algorithm using Chord

Initial State : Node i requesting for Balance transfer

Final Outcome: Nodes j establish a route to i for balance transfer.

```
1 begin
  /* Find Route Phase */
2  Node  $i$  raises Balance Transfer request with tuple
    $BT = \langle \text{amt}, \text{intr}, \text{tp}, \text{key} \rangle$ , where  $i \in N$ 
3   $i$  broadcasts  $BT$  to all users
4  Any  $j \in N - i$  does a  $\text{lookup}(\text{key})$  in her finger
   table  $FT_j$ 
5  if  $i \in FT_j$  then
6    Node  $j$  responds to  $BT$  with
    $\text{resp}_j = \langle E_{pk_i}(\text{amt}_j) \rangle$  and sends to  $i$ 
7  end
8  else
9    Node  $j$  finds nearest node  $k$  from  $FT_j$  and
   sends response  $\text{resp}_j = \langle E_{pk_i}(\text{amt}_j) \rangle$  to  $k$ 
10   for each  $k \in L$ , where  $L$  is the length of the
   path do
11      $k$  does a  $\text{lookup}(\text{key})$  in her finger table
    $FT_k$ 
12     Node  $k$  does steps 6 to 13 until  $\text{resp}_j$ 
   reaches  $i$ 
13   end
14   /* Balance Transfer Phase */
15   Node  $i$  receives  $\text{resp}_j$ 
16   if  $ts < tp$  and  $\text{amt}_j \leq \text{amt}$  then
17     Node  $i$  accepts and saves responses in
    $i.\text{resp}[]$ 
18   end
19   Set  $\text{src} = k$ ,  $\text{user} = j$ ,  $\text{dest} = i$ 
20   for each  $\text{user} \in i.\text{resp}$  do
21      $\text{dest}$  node calls  $\text{MultiSig}(\text{dest} =$ 
    $i, \text{val}, \text{user}) \rightarrow (CtBal, \sigma_{\text{dest}}, \sigma_{\text{user}})$ 
22      $\text{user}$  pays back  $lw_{\text{src}, \text{user}}$  and creates
    $\text{req}_{(\text{user}, \text{src})}$  with  $lw_{\text{src}, \text{user}} = 0$ 
23      $\text{src}$  signs  $\text{ack}$  of  $lw_{\text{src}, \text{user}} = 0$ , produces
    $\text{Sign}_{sk_{\text{src}}}(\text{ack}) \rightarrow \sigma_{\text{ack}_{\text{src}}}$ 
24      $\text{user}$  verifies  $\sigma_{\text{ack}_{\text{src}}}$  and creates  $\sigma_{\text{ack}_{\text{user}}}$ 
25      $\text{src}$  and  $\text{user}$  deletes the hashtable entries
   on  $\text{src}$  and  $\text{user}$  nodes respectively
26      $\text{dest}$  node writes  $\{CtBal, \sigma_{\text{dest}}, \sigma_{\text{user}}\}$  to
   blockchain
27   end
28   Node  $j$  informs  $\text{successors}, \text{predecessors}$ 
   about re-positioning
29   Successors of node  $j$  update their
   corresponding  $FT$ 
30   Successors call  $\text{Stabilize}(FT)$  to update their
   finger tables
31    $j$  hands over  $\text{key}_j$  to  $j$ 's successors
32    $j$  re-positions in Chord table, re-assigns  $\text{nodeID}$ 
33    $j$  calls  $\text{stabilize}(FT_j)$ 
34    $j$  updates  $FT_j$ 
35 end
```

in line 11 of this algorithm. In line 13, j calls $\text{MultiSig}(\text{dest} = j, \text{user} = i, \text{val}) \rightarrow (CtBal, \sigma_{\text{dest}}, \sigma_{\text{user}})$. In line 16, i writes $(CtBal, \sigma_{\text{dest}}, \sigma_{\text{user}})$ to blockchain. In line 17, LM exits the network after collecting a small fee from i , which is computed based on the number of links established. In the unlikely event that none of the nodes in the list sent by LM is interested in establishing a outgoing connection from i , LM does steps 2-13 again with a new list of j .

Algorithm 3: Bailout Algorithm

Initial State : Node i requesting for outgoing credit links

Final Outcome: Node i establishes outgoing links with j nodes

```
1 begin
2   $i$  sends  $m$  outgoing nodes request to  $LM$ .
3   $LM$  creates links  $LM \rightarrow j, j \in [1, 2...m]$ 
4   $LM$  waits for  $tr$  for  $i$  to respond with outgoing
   request
5  while  $tr \neq 0$  do
6     $i$  calls  $\text{OutReq}(i, j, \text{Inlink}, t) \rightarrow \text{val}$  to  $j$ 
   through  $LM, t < tr$ 
7    if  $j.\text{resp} = \text{"val"}$  then
8       $j$  calls  $\text{MultiSig}(\text{dest} = j, \text{user} =$ 
    $i, \text{val}) \rightarrow (CtBal, \sigma_{\text{dest}}, \sigma_{\text{user}})$ 
9    end
10   if  $j.\text{resp} = \perp$ , for all  $j \in [1, 2...m]$  then
11      $LM$  responds with new nodes
    $\langle v_1, v_2, v_3...v_m \rangle$ 
12      $i$  calls  $\text{OutReq}(i, j, \text{Inlink}, t) \rightarrow \text{val}$  to
    $j, j \in [v_1, v_2...v_m]$  through  $LM, t < tr$ 
13      $j$  calls  $\text{MultiSig}(\text{dest} = j, \text{user} =$ 
    $i, \text{val}) \rightarrow (CtBal, \sigma_{\text{dest}}, \sigma_{\text{user}})$ 
14   end
15 end
16  $i$  writes  $(CtBal, \sigma_{\text{dest}}, \sigma_{\text{user}})$  to blockchain.
17  $LM$  disconnects from  $i$  after collecting her fee
18 end
```

VII. IMPLEMENTATION AND EVALUATION

We simulated Chord-based routing using [21] as a base, and implemented cryptographic primitives in Charm [22]. For signatures, we used ECDSA on curve prime192v2 and used RSA 2048 for encryption. Our experiments were run on a desktop class computer with Intel(R) Core(TM) i3-7100 CPU @ 3.90GHzx4 and 8GB RAM on Ubuntu-18.04 platform. Table I shows time taken for cryptographic operations in every phase.

In order to simulate the chord network, we have made use of [21] simulator. The experiment was run on an intel i-3 generation-7 desktop with 8GB of RAM. We recorded the time taken for setting up the chord ring, broadcasting a message and we also calculated the minimum and maximum number of hops it takes for a source to reach its destination.

TABLE I: Time for cryptographic Operation by Phase

Operation	Signature (msec)	Verification (msec)	Encryption (msec)
Balance Transfer (Prefix Embedding)	3.355	3.7733	0.1707
Balance Transfer (Chord)	1.9678	2.2786	-
Bailout	3.355	3.7733	-

In the ‘‘Setup’’ operation, we record time taken for setting up the Chord network ring for a total of 1000 nodes. The ‘‘Lookup’’ operation involves finding the route by performing a $lookup(key)$ from the finger table. Time taken for response phase is the total time taken for doing a $lookup(key)$ and respond to the broadcast message. ‘‘Re-assign and Stabilize’’ is the final operation which involves updating finger tables for all the successors and predecessors of the re-positioned node, after the balance transfer operation has been carried out. The time taken for setup varies linearly with the number of users added to the Chord network.

TABLE II: Balance Transfer in Chord - Time recorded for operations

Operation	Time taken
Setup (1000 users)	23.0650 sec
Broadcast BT	13.913 sec
Lookup and Response	20.7730 sec
Re-assign and Stabilize	10.6676 sec

We give the timings for balance transfer using prefix embedding in Table III. We used the GTNA package [23] to implement balance transfer using Prefix Embedding. The time taken for ‘‘Setup’’ which involves, creating a network of 1000 users, assigning links and link weights and creating embedding co-ordinates is given in table III. The ‘‘Broadcast BT’’ represents the time taken for a requester sending out a BT (Balance Transfer) request tuple, which contains the position and amount for balance transfer. ‘‘FindRoute and Respond’’ operation is the process of finding route between a requester and responder, and also includes the time taken for sending out the response which is the encrypted value of the amount that any user is willing to balance transfer with the requester. This also includes the time taken for the user to establish an edge with the requester (create an edge between two nodes in the graph). The setup time increases linearly with increase in number of users, as expected. For ‘‘Find Route and Response’’, the time taken to route from the first node to the farthest node in the graph is given here, since that would be the longest routing path, a node would encounter. Since the graph generated by GTNA is unstructured, and randomly generated each time, we give the average value of 100 runs for 1000 users in Table III.

Table IV shows the time for bailout operation after performing balance transfer. We recorded time for setting up the network from 1000 to 10000 users. Here the setup refers to establishing connections from landmark to all the other nodes in the network. For 1000 users, the setup time for Bailout

TABLE III: Balance Transfer using Prefix embedding - Time recorded for operations

Operation	Time taken
Setup (1000 users)	26.64 msec
Broadcast BT	0.144 msec
FindRoute	10.340 msec
FindRoute and Response	25.38 msec

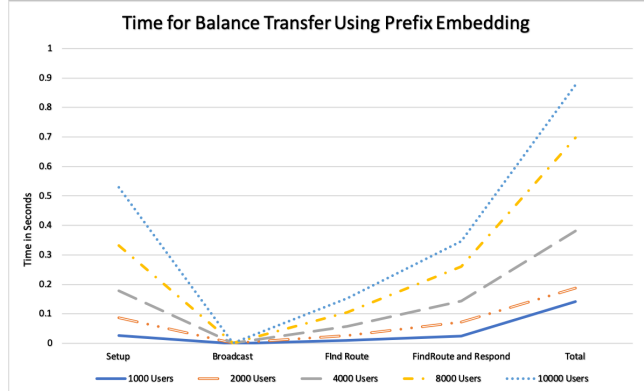


Fig. 3: Graph showing time taken for balance transfer using Prefix embedding. The time for all the operations increase with the number of users except Broadcast operation

is 0.0336 seconds and the value increases with increase in number of users.. ‘‘Create Edges’’ involves establishing edge and edge weights between the requester and other nodes who are willing to extend credits. The time taken for FindRoute is the total time taken for all the 10 nodes selected by landmark node to route to requester node.

TABLE IV: Bailout - Time recorded for operations

Number of Users	Setup (time in msec)	FindRoute (time in msec)	Create Edges (time in msec)
1000	33.672	0.072495	3.63636
2000	58.514	0.075099	7.1581
4000	123.363636	0.076168	16
8000	269.089109	0.1378	42.297
10000	369.6347	0.2779	52.7722

Figure 3 shows the time taken for operations in balance transfer in prefix embedding. We obtain the values for 1000, 2000, 4000, 8000, 10000 users by taking an average on 100 values for each operations. We plotted the time taken for operation against number of users and the time for all the operations (except broadcast) with increasing number of users. The time taken for broadcast will not vary or increase with the number of users since, this operation involves creating a balance transfer tuple and sending it out into the network, regardless of the number of users in the network.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a new technique for rebalancing in credit networks that would help a poorly connected node rebalance its links, and become an active

participant in the network, thus making the network robust, more competitive, and increasing the overall throughput of the network. Our method involves a poorly connected node creating incoming links using a process called *balance transfer* and creating outgoing links using a process called *bailout*. We present the high-level ideas and prototype them in this paper; as a part of future work, we envision to implement the idea on a real world credit network such as Ripple [2].

REFERENCES

- [1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
- [2] F. Armknecht, G. O. Karame, A. Mandal, F. Youssef, and E. Zenner, "Ripple: Overview and outlook," in *International Conference on Trust and Trustworthy Computing*. Springer, 2015, pp. 163–180.
- [3] S. Nakamoto, "Stellar," 2008.
- [4] P. Prihodko, S. Zhigulin, M. Sahno, A. Ostrovskiy, and O. Osuntokun, "Flare: An approach to routing in lightning network," *White Paper*, 2016.
- [5] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Silentwhispers: Enforcing security and privacy in decentralized credit networks." in *NDSS*, 2017.
- [6] G. Panwar, S. Misra, and R. Vishwanathan, "Blanc: Blockchain-based anonymous and decentralized credit networks," in *In Ninth ACM Conference on Data and Application Security and Privacy (CODASPY19)*, 2019.
- [7] I. Stoica, R. T. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, 2003. [Online]. Available: <https://doi.org/10.1109/TNET.2002.808407>
- [8] S. Roos, M. Beck, and T. Strufe, "Voute-virtual overlays using tree embeddings," 2016.
- [9] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 455–471.
- [10] P. Moreno-Sanchez, T. Ruffing, and A. Kate, "Pathshuffle: Credit mixing and anonymous payments for ripple," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 3, pp. 110–129, 2017.
- [11] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," *arXiv preprint arXiv:1709.05748*, 2017.
- [12] P. Moreno-Sanchez, RandomRun, D. V. Le, S. Noether, B. Goodell, and A. Kate, "DLSAG: non-interactive refund transactions for interoperable payment channels in monero," *IACR Cryptology ePrint Archive*, vol. 2019, p. 595, 2019. [Online]. Available: <https://eprint.iacr.org/2019/595>
- [13] S. Noether and A. Mackenzie, "Monero research lab," *Ring Confidential Transactions. Ledger*, vol. 1, pp. 1–18, 2016.
- [14] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 439–453.
- [15] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.
- [16] "https://ripiocredit.network/."
- [17] S. Roos, L. Wang, T. Strufe, and J. Kangasharju, "Enhancing compact routing in CCN with prefix embedding and topology-aware hashing," in *Proceedings of the 9th ACM Workshop on Mobility in the Evolving Internet Architecture, MobiArch 2014, Maui, HI, USA, September 11, 2014*, R. L. Aguiar and K. Guo, Eds. ACM, 2014, pp. 49–54. [Online]. Available: <https://doi.org/10.1145/2645892.2645900>
- [18] T. Leighton and A. Moitra, "Some results on greedy embeddings in metric spaces," *Discrete & Computational Geometry*, vol. 44, no. 3, pp. 686–705, 2010. [Online]. Available: <https://doi.org/10.1007/s00454-009-9227-6>
- [19] A. Dury, "Peer-to-peer computing in distributed hash table models using a consistent hashing extension for access-intensive keys," in *Agents and Peer-to-Peer Computing, Third International Workshop, AP2PC, 2004, New York, NY, USA, July 19, 2004, Revised and Invited Papers*, ser. Lecture Notes in Computer Science, G. Moro, S. Bergamaschi, and K. Aberer, Eds., vol. 3601. Springer, 2004, pp. 185–192. [Online]. Available: https://doi.org/10.1007/11574781_17
- [20] F. P. NIST, "180-1: Secure hash standard," 1995.
- [21] I. Kazmi and S. Bukhari, "Peersim: An efficient & scalable testbed for heterogeneous cluster-based p2p network protocols," in *Proc. of the 2011 UKSim International conference on modeling and simulation*, 2011, pp. 420–425.
- [22] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: a framework for rapidly prototyping cryptosystems," *J. Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013.
- [23] B. Schiller and T. Strufe, "GTNA 2.0 - a framework for rapid prototyping and evaluation of routing algorithms," in *2013 Summer Simulation Multiconference, SummerSim '13, Toronto, Canada - July 07 - 10, 2013*, 2013, p. 23.