Rebalancing in Acyclic Payment Networks*

Lalitha Muthu Subramanian, Guruprasad Eswaraiah, Roopa Vishwanathan
Department of Computer Science, New Mexico State University, USA
{lalitha,guru,roopav}@nmsu.edu

***Abstract*—In this paper, we propose a technique for rebalancing link weights in acyclic payment networks, which are peer-to-peer networks with payment channels or links between users. Payment networks such as Ripple and Stellar offer a range of services such as real-time payment transfers, including cross-currency transactions, and are growing in popularity. They incentivize users to use them by offering much lower transaction fees, and quick transfer times as compared to traditional bank wire transfers. The link weights are positive integers, and once two users connected by a link spend their link weight, the link gets exhausted, and no more transactions can be done over the link. In order to allow users to do any more transactions, the link weight must be increased, a process known as *rebalancing* link weights. Existing methods of rebalancing require users to perform expensive blockchain transactions of closing and refunding the channel; we consider the problem of rebalancing a payment channel in real time in an efficient way. Our decentralized technique of rebalancing will help users in acyclic payment channel networks to rebalance their link weights on an as-needed basis, and with minimal computational cost, and only off-chain transactions.**

***Index Terms*—rebalancing, blockchain, payment networks**

I. Introduction

Since the growth of Blockchain and Bitcoin [1], many payment channel networks have come into existence to make service delivery more efficient in the financial services industry. Ripple [2] exemplifies the idea of a payment network that provides high transaction throughput, and low end-to-end transfer time, along with providing users the ability to perform cross-currency transactions. Ripple uses a permissioned blockchain to write all its transaction logs. Unlike Bitcoin and other cryptocurrencies, payment networks [3], [4], [2], [5] have two users i and j who locally maintain a directed credit link from user i to j that represents the amount of funds that i can transfer to j. To provide even a basic modicum of security, such networks need to provide mechanisms to correctly determine a link weight at a given point of time, prevent arbitrary changes to link weights by malicious users, and provide accountability.

There has been a growing interest in finding solutions for various privacy and security issues in payment networks such as [6], [7], but not many works focus on rebalancing techniques in payment networks. Rebalancing in payment networks is an important problem to study, since, if the link weight between any two nodes drops to 0, no transactions can be done until the link is refunded, a process which involves expensive on-chain transactions. We model a payment network as a directed acyclic graph, where the set of users is the set of nodes in the graph, the links represent peer-to-peer payment channels, and directed edges represent the directionality of payment flow, and present the following contributions.

Our Contributions: In our system, each node maintains some local meta-data that it can use for determining which of its immediate neighbors can best help it rebalance its links when required. Our rebalance protocol is simple, and provides security against malicious nodes that try to mis-report link weights, or otherwise misbehave during the rebalance process. Our rebalance protocol is completely decentralized, as opposed to previous work, which required a centralized authority. We provide link privacy, such that no node in the network knows the value of link weights it is not directly connected to. In terms of efficiency, our protocol is linear in the number of nodes in the network in the worst-case, and requires minimal cryptographic operations.

Outline: In Section II, we briefly discuss related work in the area, in Section III, we give our system design, in Section IV, we give the adversary model, in Section V, we describe our three-phase rebalancing protocol, and in Section VI, we give an asymptotic analysis and other estimates of our system.

II. Related Work

Khalil et al. proposed REVIVE [8], a payment channel network that allows users to rebalance their channel without having to communicate with the blockchain. Although impressive, their channel rebalancing process is not transparent, requires a centralized, elected party called "leader" that has to always be online, whom all nodes will contact for and during the rebalancing process, and the leader knows all nodes' link weights (not privacy-preserving). Moreover, REVIVE only works in a restricted class of network topologies that has *cyclic* graphs, and the authors point out that their mechanism will not work in the case of acyclic graphs. Lightning network [9] constructed on top of Bitcoin is composed of one-to-one payment channels that scale well (105 million users). Lightning network does re-balancing off-chain, but again only for *cyclic* graphs. To rebalance a low-weight link, a node makes a payment to itself across a circular path of payments. Circular payments are not free (every node in the path needs to receive a relay fee, and the cycles could be arbitrarily large). In our design we have a maximum of three participants, and avoid the cost of paying every node in the path. There are several other notable works in the general area of payment channel networks [3, 6],

*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.

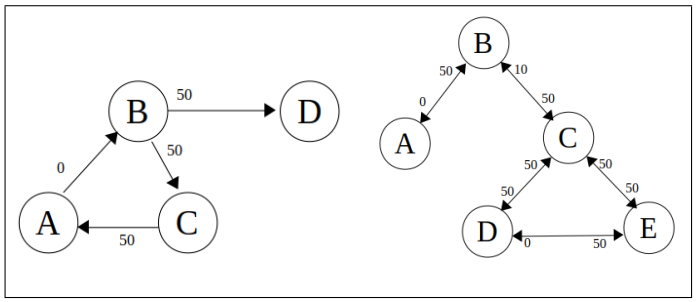
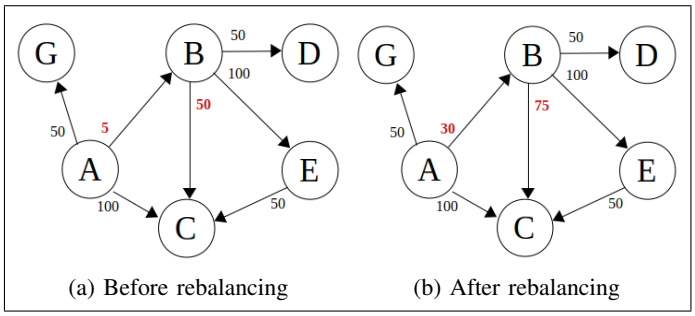
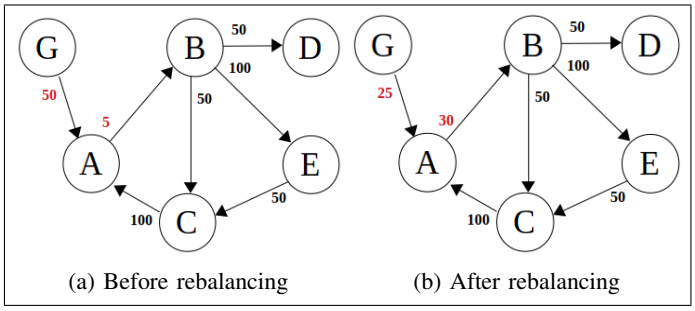


Fig. 1: (a) Unidirectional payment network. (b) Bidirectional payment network.



(a) Before rebalancing (b) After rebalancing

Fig. 2: Unidirectional network before/after rebalancing.



(a) Before rebalancing (b) After rebalancing

Fig. 3: Rebalancing link weight A-B through G-A-B using splicing.

among others, but we do not cover them here as they do not specifically deal with the rebalancing problem.

III. System Design

The topology of a payment network graph could be one of the following:

- Unidirectional links with cycles
- Unidirectional links without cycles
- Bidirectional links with cycles
- Bidirectional links without cycles

We depict the four kinds of topologies in Figure 1. We note that the first two cases: networks with unidirectional links have received less attention in the research community than compared to networks with bidirectional links. Figure 1a represents a network with unidirectional payment links. The payment path A-B-C depicts a unidirectional cyclic path, and A-B-D represents a unidirectional acyclic path. Figure 1b represents a bidirectional payment network. The path C-D-E represents a bidirectional cyclic path, and path A-B-C-D represents a bidirectional acyclic path.

Figure 2 describes a scenario which depicts rebalancing in a unidirectional payment network without cycles. Initially, A's link weight is 5 with respect to B and A wants the A-B link to be rebalanced. For this purpose, A sends out a request to B for rebalancing. B in turn contacts its next-hop neighbor C, requesting C to increase the B-C link weight by 25 units[1] If C agrees to the increase, the B-C link gets incremented by 25, which enables B to increase the A-B link by 25. This maintains the zero-sum balance at node B, i.e., the difference between B's outgoing and incoming links stays the same, which is an important property to provide while rebalancing payment networks [8].

Bidirectional networks such as Lightning network use a mechanism called *splicing*, a feature that enables users to combine the open channel, close channel and on-chain Bitcoin outputs into one transaction, which could arguably be the most critical feature on the network. In splicing, any node can transfer partial funds from one of its incoming links to an outgoing link. Unidirectional networks such as the one depicted in Figure 3 will be a trivial case of the bidirectional network splicing idea, where a node A can get funds transferred from an incoming link and increase the weight of an outgoing link. In Figure 3a, A can decrement the G-A link by 25 units, and transfer 25 units to the A-B link. But what happens when A *does not have* any incoming links? We focus on finding ways for a node in an unidirectional acyclic network when the node that wants to perform rebalancing does not have any incoming links.

Generalizing the idea, in our system, we have n nodes, where a node i wants to rebalance an outgoing link between (i, j). i sends a rebalance request to j and waits for j to respond within time t_s with a value val. The idea is node j will ask all of its next-hop neighbors on outgoing links, if they would be willing to increment their links with j by a value val. If node j's next-hop neighbor k is willing to help, it increments its incoming (j, k) link by val, and j will increment its incoming (i, j) link by val. We assume that the number of neighboring nodes connected through j will be large. To deal with dense networks, we specify a $Hopcount$ variable in our algorithms, whose exact value is implementation-dependent. We give our table of notations in Table I.

IV. Adversary Model and Security Properties

In our system, the adversary can adaptively corrupt a single user or a set of users involved in rebalancing. The corrupted user(s) can be either the initiator of the rebalancing request, or peer nodes receiving the request. Each user i has her own signing and verification key pair (sk_i, vk_i). Once i is corrupted, her links will be controlled by the adversary, the adversary can misreport i's link value, not respond to requests, and relay fraudulent rebalance requests to neighbors. We assume that an adversary cannot corrupt all users in the network, and thus may

[1]In payment networks, units can be any fiat currency, e.g., $, or , or cryptocurrency.

TABLE I: List of notations

Variable	Definition
t_s	Timestamp
(sk_i, vk_i)	Signing/verification keys of user i
$table_i$	Local hash table storing info about link weights
$Hopcount$	Maximum hop-count
$CtRes$	Contract created in reserve phase
$CtBal$	Contract created in pay phase
σ_{Res}	Signature created on contract $CtRes$
σ_{Bal}	Signature created on contract $CtBal$
t_e	Contract expiry time
$w(i,j)$	Link weight between i and j
t_u	Time of link weight update

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 rebalancing, and briefly explain how our system provides them.

Desired Security/Privacy 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 rebalancing. This is done in our system by including just the update value and not specifying the link value in the rebalance contract.

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.

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

Corrupt rebalance requestor node: Any user who sends the rebalancing request and can act maliciously, claiming or misreporting a different value after having rebalanced with any of its peers. In our model, we prevent such claims by providing each of the user in rebalancing with two signed contracts, $CtRes$, that is signed during the reserve phase, and $CtBal$ that is signed during the pay phase. The algorithm creating these contracts is defined in Section V of this paper.

Corrupt responder node: A malicious responder j might try to get funds from its neighbors, but will refuse to pay node i. We have designed our pay phase such that the balance transfer or the link weight update occurs first between the requester node i and j, for which i would provide an acknowledgment to j as a proof of link weight update, which j would need to show to its own neighbors. By this method, we prevent any malicious j from diverting the funds to update her own link weight rather than responding to the rebalance request.

Corrupt neighbors: Any neighbor of a rebalance requestor node, on receiving the rebalancing request can act maliciously by just responding to every such request, but not allowing the rebalancing to go through (denial-of-service). In our system, we make use of a timer t_s; any user i doing a rebalance would have a time-out period after which the received responses are not accepted, and if a neighbor does not get back within t_e, the request is considered abandoned. This way we prevent having the requester to wait for a longer time than necessary to do her rebalancing, identify malicious intermediaries, and block them for future rebalance requests.

V. Our Construction

In this section, we describe the three phases of our constructions: the rebalance request phase, the respond phase, and the reserve/payment phase.

A. Rebalance request phase

Algorithm 1 explains the rebalancing request and response that is carried out between two nodes, i and j where node i wants to rebalance the link between i and j. At a high-level the idea is j will contact all its neighbors, find a neighbor k who will increment the (j,k) link by a value val, which will enable j to increment the (i,j) link by val.

First, node i will check the link weight of the (i,j) link (Line 2) stored in a hash table locally by i. Next, node i sends a rebalance request to node j with a timestamp and hopcount (Line 3). Node j calls the $\mathsf{Rspond}(\mathsf{table}_j) \longrightarrow j_{resp}$ function in Line 4; the goal of this function is for j to tell i whether j is willing and able to be part of a rebalancing transaction, and if so, what is the max. amount by which j can rebalance the (i,j) link. The details of the respond algorithm are given in Algorithm 2. All the nodes have their link weights arranged in ascending order in their local hash table so as to be efficient while raising a rebalance request to any of its peers. A node j figures out if any of its neighboring nodes, k, is willing to respond to the rebalance request with a value val_j in the Rspond function. The j node could either return a value val_j or $\perp$ (which would be returned when none of j's neighboring nodes within the hopcount radius are willing to participate in the rebalancing request, or any of the nodes timeout). If the response from node j is "$\perp$", then node i aborts the rebalancing with j (Line 15). In the former case, Node i sends an acknowledgement to j accepting the offered value val (Line 6). Once i sends the acknowledgement to j, j calls the Reserve function (Line 7) to create agreements to the value val between nodes i, j, k.

Reserve returns $CtRes_{ijk}$ which is the signed contract between (i,j) and (j,k) along with signatures σ_{Res_i}, σ_{Res_j}, σ_{Res_k}. Each node involved in the rebalancing store a copy of the contract along with the signature as a proof of the reserve agreement (Line 8). These contracts should be used by the nodes in case of resolving disputes if any of the parties become malicious. This contract is then sent as input to Pay function, which updates the link weights (channel balances) between

(i, j) and (j, k) (Line 9). Node i, j and k keep a copy of the pay contracts, $CtBal_{ij}$, $CtBal_{jk}$ respectively as a proof of channel updates and update their link weights accordingly in their respective local hash tables (Line 11).

Algorithm 1: Rebalance request phase

1 **begin**
2 Node i does $\mathsf{table_i.get} \longrightarrow (j, w(i, j))$, $i, j \in [1..n]$
3 Node i calls $\mathsf{Rreq}(t_s, j, hopcount)$
4 j calls $\mathsf{Rspond}(\mathsf{table}_j) \longrightarrow j_{resp}$
5 **if** j_{resp} == val **then**
6 Node i accepts the value val and sends an acknowledgement to j
7 j calls $\mathsf{Reserve}(i, j, k, val) \longrightarrow CtRes_{ijk}, \sigma_{Res_i}, \sigma_{Res_j}, \sigma_{Res_k}$
8 Nodes i, j and k keep a copy of $\sigma_{Res_j}, \sigma_{Res_i}\&\sigma Res_k, \sigma_{Res_j}$ respectively
9 j calls $\mathsf{Pay}(CtRes_{ijk}, val) \longrightarrow CtBal_{ij}, CtBal_{jk}, \sigma_{Bal_i}, \sigma_{Bal_{j1}}, \sigma_{Bal_{j2}}, \sigma_{Bal_k}$
10 Node i keep a copy of $CtBal_{ij}$, $\sigma_{Bal_{j1}}$
11 Node j retains a copy of $CtBal_{ij}$, $CtBal_{jk}$,$\sigma_{Bal_i}, \sigma_{Bal_k}$
12 Node k retains $CtBal_{jk}$,$\sigma_{Bal_{j2}}$
13 **end**
14 **else if** $j_{resp} = \perp$ **then**
15 Node i rejects j_{resp} and aborts rebalancing with j_{th} node.
16 **end**
17 **end**

B. Respond phase

Algorithm 2 defines the Rspond function that is run by a node j that has received a rebalance request. We refer to node j as the *intermediary* node involved in the rebalancing scheme. This algorithm takes j's local hash table, table_j and returns either a value val_j or a "$\perp$" as the return value. Node j scans her look-up table, does a $\mathsf{table_j.get}$ which contains the weights of j's links in increasing order , and requests each neighboring node connected by a given link for an increase in link weight (Line 4,5). The idea is that, it would be easier for j to request a value val_j from links that have low weights first. When a node k is willing to increase the (j, k) weight (and thus be involved in rebalancing), j responds with a value val_j to i (Line 8).

C. Reserve and pay phase

This phase has two parts: reserve, where rebalance agreements are established between nodes, and the pay part, where the actual channel balances are updated. Algorithm 3 describes the reserve and pay phases. Algorithm 3 defines two different functions Reserve and Pay which are used in algorithm 1 of this section.

The Reserve function takes as input nodes i, j, k, rebalance amount, val, and produces as output a rebalance contract,

Algorithm 2: Respond Phase

1 **Function** Rspond (table_i)
Parties: $i, j, k \in [1 \dots n]$
Result: j_{resp}
2 j sets a variable $Ctr = 0$
3 **while** $Ctr = 0$ **do**
4 j does $\mathsf{table_j.get} \longrightarrow (j, w(j, k))$ in table_j
5 j requests increase in $w(j, k)$, if yes, updates $Ctr = 1$
6 **end**
7 k responds with a value val_j to j
8 j responds with a value val_j to i
9 **end**

$CtRes_{ijk}$. The contract is created by node k and contains parties identities, value, and an expiry time for the contract (Line 2). An expiry time t_e indicates the time period for which parties will hold or reserve the value, val; if the reserve operation is not followed within time t_e by a pay operation, the held values will be released by all nodes.

In Line 3, j verifies the contract and k's signatre, and produces her signature σ_{Res_j}. Then node i verifies the contract $CtRes_{ijk}$ and also signatures σ_{Res_j} and σ_{Resk}, and signs the contract: $Sign_{sk_i}(CtRes_{ijk}) \longrightarrow \sigma_{Res_i}$ (Line 4). The Reserve function returns a contract $CtRes_{ijk}$ and signatures $\sigma_{Res_i}, \sigma_{Res_j}, \sigma_{Res_k}$ signed by all the nodes involved in the rebalancing transaction.

Line 6 of this algorithm defines the Pay function which takes reserve contract $CtRes_{ijk}$ as input and returns pay contracts $CtBal_{ij}$,$CtBal_{jk}$, which are proof of channel updates to be retained by each of the nodes to resolve any later disputes. In Line 7, node j verifies the contract $CtRes_{ijk}$ and updates the (i, j)channel weight by val. Nodes i and j store the new weight in their respective local hash tables (Line 8, 9). We require node j to update the (i, j) link first and node k to update the (j, k) link later, to protect against a malicious j who might not update the (i, j) link, if the (j, k) link gets updated earlier. In Line 10, node i signs and sends an acknowledgement of the (i, j) link update, which will be presented as proof to k, where t_u is the time of update.

In Line 11, k verifies the contract $CtRes_{ijk}$ and updates the (j, k) link weight by val. Following this, nodes j, k update their local hash tables with the new link weights (Line 12, 13). All the three participants i, j, k retain a copy of $CtBal_{ij}$ and $CtBal_{jk}$ and the signatures on the contracts as a proof of the rebalancing.

VI. Analysis

In this section, we provide some preliminary analysis for the protocols that comprise our system. For the purpose of the analysis, we consider three users: node i that initiates the rebalance request, i's neighbor j, which acts as an facilitator, and j's neighbor k, which acts as a responder.

One important measure of the efficiency of any system that uses cryptography is the raw number of cryptographic

Algorithm 3: Reserve and Pay Phase

Function Reserve (i, j, k, val)
  Parties: i,j,$k \in [1 \dots n]$
  Output: $CtRes_{ijk}, \sigma_{Res_i}, \sigma_{Res_j}, \sigma_{Res_k}$
  k creates contract $CtRes_{ijk}$ for an amount val between (i, j, k), where $CtRes_{ijk} = \langle i, j, k, val, te \rangle$, produces $Sign_{sk_k}(CtRes_{ijk}) \longrightarrow \sigma_{Res_k}$
  j verifies σ_{Res_k} and agrees to $CtRes_{ijk}$, produces $Sign_{sk_j}(CtRes_{ijk}) \longrightarrow \sigma_{Res_j}$
  i verifies σ_{Res_k} and σ_{Res_j}, agrees to $CtRes_{ijk}$, produces $Sign_{sk_i}(CtRes_{ijk}) \longrightarrow \sigma_{Res_i}$
end
Function Pay $(CtRes_{ijk})$
  Input : $CtRes_{ijk}$ is the input
  Output: $CtBal_{jk}$, $CtBal_{ij}$,$\sigma_{Bal_i}, \sigma_{Bal_{j1}}, \sigma_{Bal_{j2}} \sigma_{Bal_k}$
  j verifies $CtRes_{ijk}$, increases $w(i, j)_{new} = w(i, j)_{old} + val$
  j does $\mathsf{table_j.put}(w(i, j)_{new})$
  i does $\mathsf{table_i.put}(w(i, j)_{new})$
  i sends an $Sign_{sk_i}(ack) \longrightarrow \sigma_{ack_i}$ to j as a proof of channel update, where $ack = \langle i, j, val, t_u \rangle$
  k verifies $CtRes_{ijk}$ and σ_{ack_i} from j, increases $w(j, k)_{new} = w(j, k)_{old} + val$
  k does $\mathsf{table_k.put}(w(j, k)_{new})$
  j does $\mathsf{table_j.put}(w(j, k)_{new})$
  i, j and k sign $CtBal_{ij}$ and $CtBal_{jk}$ as a proof of channel update, produces, $Sign_{sk_i}(CtBal_{ij}) \longrightarrow \sigma_{Bal_i}$, $Sign_{sk_j}(CtBal_{ij}) \longrightarrow \sigma_{Bal_{j1}}$, $Sign_{sk_j}(CtBal_{jk}) \longrightarrow \sigma_{Bal_{j2}}$, $Sign_{sk_k}(CtBal_{jk}) \longrightarrow \sigma_{Bal_k}$
end

operations that need to be performed per user, per phase by the system. The cryptographic primitives used in our system are digital signatures and their verification. We present the number of signatures and verification operations per user per phase in Table II. The relevant phases are the reserve and pay phase; the respond phase does not use any cryptographic operations and the rebalance request phase calls the reserve and pay phases, but does not have any cryptographic operations other then the ones contained in them. As the table shows, the number of signatures and verifications done per user is minimal.

TABLE II: Cryptographic operations

Phases	Reserve Phase		Pay Phase	
Cryptographic operations	Sign	Verify	Sign	Verify
Initiator (i)	1	2	2	0
Intermediate (j)	1	1	2	1
Responder (k)	1	0	1	2

Next, we present an asymptotic analysis of the running time for each phase of the protocols in our system in Table III. For this, we model the payment network as a graph, $G = (V, E)$. We use the fact that hash table lookup and insert operations can be done in $O(1)$ time. Also, since we are primarily concerned with *computation time* at each node, we disregard network latency and other network delays. The respond phase involves the facilitator node j, who incurs a cost of $O(|V|)$ since in the worst case, j has to request increases from all nodes in its hash table, which in the worst case could be $|V|$. The cost of the reserve and pay phase is $O(1)$ for all nodes involved in the transaction, since every node has to do only a fixed number of signatures and verification operations (which are quantified in Table II), plus some hash table insert operations. Finally, the rebalance request phase calls the respond and reserve/pay phases internally. The cost of the rebalance request phase is dominated by the cost of the respond phase which is $O(1)$. Note that these are worst-case times; on an average, the complexities are expected to be lesser.

TABLE III: Worst case running time per phase for a network graph $G = (V, E)$

Phase	Worst-case time
Respond Phase	$O(\|V\|)$
Reserve and Pay Phase	$O(1)$
Rebalance request phase	$O(\|V\|)$

VII. Conclusion and Future Work

In this paper, we have presented a design to rebalance the link weights in unidirectional, acyclic payment networks. We have done a preliminary analysis of the number of cryptographic operations needed in our design. As a part of future work, we plan to analyze our system in a formal cryptographic adversarial framework, as well as do an extensive thorough experimental evaluation.

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] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Silentwhispers: Enforcing security and privacy in decentralized credit networks," in *24th Annual Network and Distributed System Security Symposium, NDSS*, 2017.

[4] 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.

[5] P. Dandekar, A. Goel, R. Govindan, and I. Post, "Liquidity in credit networks: A little trust goes a long way," in *Proceedings of the 12th ACM conference on Electronic commerce*. ACM, 2011, pp. 147–156.

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

[8] 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.

[9] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.