Low-overhead Routing for Offchain Networks with High Resource Utilization

Xiaoxue Zhang, Shouqian Shi, and Chen Qian
University of California Santa Cruz
{xzhan330, sshi27, cqian12}@ucsc.edu

***Abstract*—Off-chain networks have been designed and utilized to address the scalability challenge and throughput limitation of blockchains. Routing is a core problem. An ideal off-chain networks routing method needs to achieve 1) high scalability that can maintain low per-node memory and communication cost for large networks and 2) high resource utilization of channels. However, none of the existing off-chain routing methods achieve both requirements. In this work, we propose WebFlow, a distributed routing solution for off-chain networks, which only requires each user to maintain localized information and can be used for massive-scale networks with high resource utilization. We make use of two distributed data structures: multi-hop Delaunay triangulation (MDT) originally proposed for wireless networks and our innovation called distributed Voronoi diagram. We propose new protocols to generate a virtual Euclidean space in order to apply MDT to off-chain networks and use the distributed Voronoi diagram to enhance routing privacy. We conduct extensive simulations and prototype implementation to further evaluate WebFlow. The results using real and synthetic off-chain network topologies and transaction traces show that WebFlow can achieve extremely low per-node overhead and a high success rate compared to existing methods.**

***Index Terms*—Blockchain, Off-chain Networks, Routing**

I. Introduction

While blockchains have shown great success as decentralized digital ledgers, *scalability* remains a huge problem with growing numbers of users and transactions [1], [2]. For instance, Bitcoin can only support 10 transactions per second at peak in 2020 [3]. The reason for its low throughput is that every node processes all transactions and the consensus is achieved by Proof of Work (PoW), a time- and resource-consuming process. The recently proposed concept of *off-chain networks*, also known as *payment channel networks* (PCN) [2], [4] provides a high-throughput solution for blockchain-based payment systems. In PCNs, two users can conduct multiple transactions via a bi-directional channel without the need to confirm every transaction on the blockchain. A user can make a transaction with another arbitrary user via a multi-hop path, where any two consecutive users on the path share a channel. Intermediate nodes typically charge a fee for forwarding the transaction. The PCN is a promising solution to increase the scalability of blockchains because most transactions can be achieved in an off-chain manner.

A core problem of PCNs is routing, i.e., finding a path between two arbitrary users. Current routing solutions of PCNs can be classified into two types. 1) Centralized routing that assumes every user knows the entire network topology, including all nodes, channels, and channel balances. Each user runs a centralized algorithm to determine the routing paths [5], [6]. *This approach is not a scalable solution* because each user needs a large memory space to store the network information and every change in channels needs to be broadcast to all users. Besides, some users' confidential information will be exposed. 2) Distributed routing that each user only knows and interacts with a small subset of other users, independent of the entire network size [7], [8]. It is ideal for large-scale PCNs.

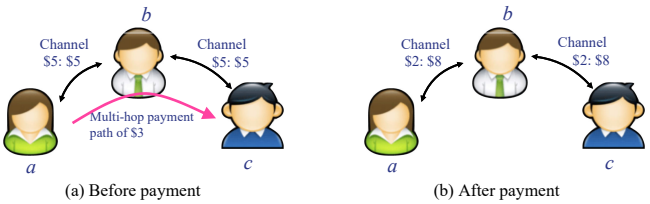


Fig. 1. A multi-hop payment in a PCN.

Existing distributed routing methods for PCNs, however, have a **crucial limitation**: they cannot effectively utilize the channel resource in a PCN. We show an example of a PCN with 7 users and 9 channels as in Fig. 3(a). One typical distributed routing for PCNs is landmark routing [7] as shown in Fig. 3(b). Every transaction (such as the one from c to d) will be sent to the landmark first. The landmark knows the whole network topology and will find the path to the destination d. This approach does not utilize non-tree channels, and channel resources around the landmark could quickly run out. One improvement is embedding-based routing, which is designed to avoid some unnecessary hops in the landmark routing [8]. It learns a vector embedding for each node. Each node relays each transaction to the neighbor whose embedding is closest to the destination's embedding. However, many transactions still need to pass through the landmark and cause similar problems. **Poor channel utilization means fewer transactions can be successfully delivered** in a PCN.

In this paper, we introduce WebFlow, a scalable and distributed routing solution for PCNs with small per-node overhead and high channel resource utilization. WebFlow allows each node to explore the routing paths without relying on certain "hot spots" such as landmarks. Hence, resource utilization is significantly improved. Meanwhile, each node only stores the information of a few neighbors without knowing the global topology. As shown in Fig. 2, WebFlow could provide significantly lower per-node overhead compared to centralized, landmark, and embedding routing while achieving similar

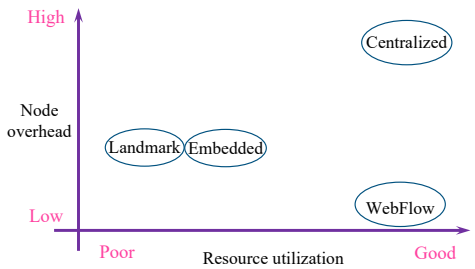


Fig. 2. Trade-off of PCN routing

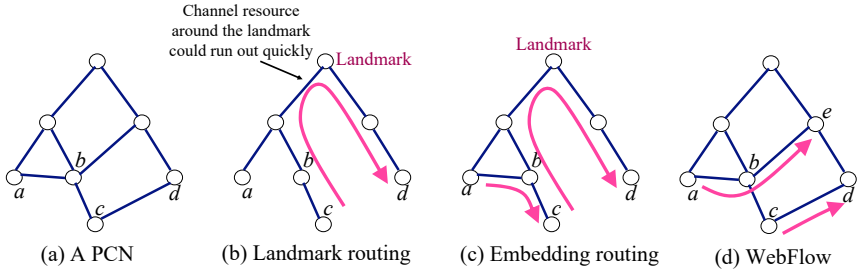


Fig. 3. Example of different routing methods for PCNs.

channel resource utilization as centralized routing.We consider semi-honest attackers. WebFlow is aiming at preventing them from stealing users' financial situations.

WebFlow is a coordinate-based routing protocol for PCNs. It allows every node to calculate a set of Euclidean coordinates and uses the coordinates to perform coordinate-based greedy routing. We design a system that nodes maintain a multi-hop Delaunay triangulation (MDT) [9] based on only the channels with several users to achieve a high success ratio of pathfinding. To further protect the anonymity of senders and recipients, our **important innovation** is to use the property of a distributed Voronoi diagram to achieve the routing tasks. Different from traditional greedy routing, it does not require the coordinate of the destination in a routing message. Instead, it introduces a direction vector to help to determine the path that hides the actual destination.

In summary, this paper makes the following contributions:

- We design WebFlow, a new routing protocol for PCNs with low per-node overhead and high resource utilization.
- We propose an enhanced version of WebFlow to protect user privacy, i.e., the identities of source and destination of a transaction can be hidden.
- We implement WebFlow based on real-world PCN topologies and transactions and build a prototype. The results show the claimed advantages of WebFlow compared to the state-of-the-art protocols.

The rest of this paper is organized as follows. Section II introduces the related work. The system overview and model are presented in Section III. We describe the detail design of the WebFlow protocols in Section IV and Section V. Section VI presents the evaluation results of WebFlow. Section VII summarizes our conclusions.

II. Related work

Routing is a critical problem in a PCN. In the Lightning Network [2], each node locally maintains the network topology and a global routing table. Currently, source routing is utilized such as shortest path [10] and max-flow routing algorithm [11], which has high costs on the user side. Flare [12] is a hybrid routing algorithm for Lightning Network. It requires nodes to proactively gather information about the topology and maintain a routing table, which also triggers extra costs.

Recent works such as Spider [6] and Flash [5] use centralized routing. Spider actively considers the cost of channel imbalance by preferring routes that rebalance channels. But the proposed centralized scheme has high probing overhead. Flash reduces the probing overhead by treating elephant and mice payments differently. It requires each user to maintain a routing table for mice payments, and periodically refreshes it when the local network topology is updated. It applies a huge memory cost for each user.

To reduce the per-node overhead in PCN, distributed routing has been proposed. SilentWhispers [7] utilizes landmark routing, in which all paths need to go through the landmarks. It makes the channels around the landmarks over-used while other channels are under-used, and some paths could be unnecessarily long. SpeedyMurmurs [8] uses embedding-based routing. Computing and updating the coordinates as the topology and channel balances change is a major challenge of this approach. EPA-Route [13] leverages MA ordering [14], [15] to create a static routing table, and dynamically prunes useless paths when probing paths. But it requires that all the nodes own the global network topology. And it cannot deal with the dynamic changes of the payment networks, such as nodes joining in or going offline.

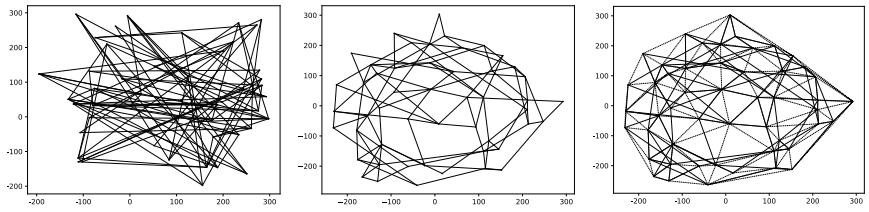
Other overlay network routing methods such as distributed hash tables [16] cannot be used for PCNs because one cannot be forced to build a channel with an arbitrary user. There are many existing Ad-Hoc routing protocols such as DSR, AODV, and GPSR. However, GPSR only works for planar graphs. In DSR and AODV, route discovery is based on flooding, which introduces considerable routing overhead, and can not be applied to the large-scale PCNs. Besides, the initial balance of a payment channel is deposited by the users during the channel setup, and keeps updating during each transaction. So the route maintenance mechanism in both DSR and AODV may not work well in such dynamic networks.

Compared to existing work, WebFlow is the first solution that considers both the scalability of user overhead and channel resource utilization in PCN routing.

III. Background and Overview

A. Payment Channel Networks

In PCNs, two users can conduct multiple transactions via a bi-directional channel. For this channel, only two transactions need to be recorded on the blockchain: opening and shutting down the channel. Each user commits a certain fund at the opening of this channel [17]. Then they can make any number of transactions that update the tentative distribution of the channel funds. These transactions only need to be signed by the two users while *not* be broadcast to the blockchain. For two arbitrary users without a channel, they can make transactions



(a) Network with random coordinates (b) WebFlow coordinates (c) MDT of the set of nodes

Fig. 4. Original PCN graph, the graph after node positioning, and MDT graph.

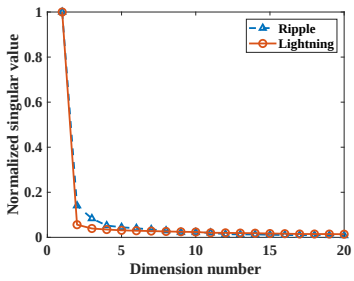


Fig. 5. SVD analysis of PCNs

via a multi-hop path, where any two consecutive users on the path share a channel. For example, in Fig.1(a), user a wants to pay user c without a direct channel. User b has direct channels to both a and c. Hence they can use the multi-hop path $a-b-c$ and adjust the fund distribution on the channels ab and bc accordingly as in Fig.1(b), while b may charge a nominal transaction fee.

B. Network Model

WebFlow is a distributed routing protocol for large-scale PCNs. We model a PCN as a graph $G = (V, E, \Psi)$, where the set V represents the PCN users, the set E of weighted edges represents the payment channels. $\psi_{uv}, \psi_{vu} \in \Psi$ represent the balances of channel uv in two directions. The path of a transaction is accepted only if the amount of this transaction is less than every channel's funds along this path. Every node knows the channels and their balances to its neighbors.

Problem definition. The routing problem of WebFlow is described as follows. Consider a transaction t initiated by *sender* s that should be received by the *recipient* r. WebFlow needs to find a path from s to r, where two consecutive nodes on the path should share a channel and each channel has enough balance to make the payment to the next hop. The success of routing implies that s can make a transaction with r by a sequence of transactions involving other nodes, even if s and r have no direct channel. The atomicity and security of the transaction are guaranteed by Hash Time Locked Contract (HTLC) [10]. We have three objectives: 1) each node should have limited memory and communication overhead, which is independent of the network size; 2) WebFlow should achieve a high success rate of transaction routing; 3) WebFlow should achieve high channel resource utilization.

C. Attacker Model

Our primary attack scenario is the malicious users interested in others' financial situations. They are honest-but-curious users that passively observe the channels related to them and try to infer the source and destination of the transactions. They tend to follow the protocol to do the routing and get profits since misbehavior can be detected by HTLC. We assume that the adversary controls a subset of users in the network, and these users can collude to speculate other users' information. It cannot choose the user to collude arbitrarily, since some users are harder to undermine with higher security. The attackers here aim to undermine the user's privacy and profit from it rather than perform a denial-of-service attack. They may also collude to choose a longer routing path to earn more transaction fees.

Similar to SpeedyMurmurs [8], our goal is to hide values, and achieve anonymity of sender and recipient when making transactions. We use the term *value privacy*, *sender/recipient anonymity* respectively to refer to the following privacy goals.

Value privacy: We say that a PCN can achieve value privacy if it is impossible for any adversary to know the total value of a transaction between two honest users.

Sender/recipient anonymity: We say that a PCN can achieve sender/recipient anonymity if the adversary cannot determine the original sender or the actual recipient of a transaction.

We *formally define a metric to evaluate the sender/recipient anonymity* of a network and its routing algorithm as follows[1]. The anonymity measure follows the anonymity definition that has been used for anonymous routing such as [19]–[21]. Anonymity is the state of being not identifiable within a set of subjects. Since it is impossible that all nodes look equally likely to be the sender or recipient in real-world systems, attackers can assign each node u a probability p_u as being the sender or recipient using knowledge of leaked information from the system. All the nodes in the network are denoted as a finite set V. The anonymity of the system is measured as:

$$\mathbb{A} = \frac{-\sum_{u \in V} p_u log_2(p_u)}{\log_2(|V|)} \quad (1)$$

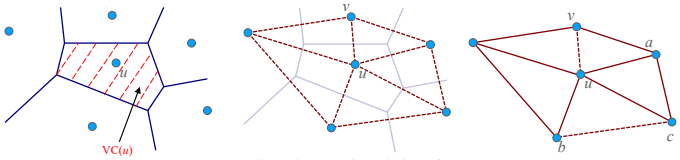
D. Analysis methodology of this work

From our observation of real PCN topologies, they are not regular graphs such as grids or trees. Hence, it is **impossible** to use theoretical formulation to analyze the routing performance. In this paper, we use **extensive simulations with both real and synthetic network topologies** to analyze the routing performance. We also use a prototype implementation to demonstrate that WebFlow can work in practice.

IV. Baseline Design of WebFlow

To achieve distributed and scalable routing, WebFlow utilize the idea of coordinate-based greedy routing that has been widely used for wireless networks [9], [22]. The basic idea is that in wireless networks, each node knows its geographic coordinates as well as its neighbors'. Then without knowing the whole topology, each node can simply forward a packet to a neighbor who is geographically closest to the destination.

[1]Detailed analysis can be found in our full version [18].



(a) Voronoi diagram with a set V of nodes (b) Delaunay triangulation of the set of nodes (dashed lines) (c) MDT of the set of nodes

Fig. 6. Voronoi diagram, Delaunay triangulation (DT), and MDT

One problem is a node could be a local minimum. This problem can be solved in various ways [9], [22]. Another problem is that many users do not want to share their geographic locations in PCNs which could be a threat to their privacy. In addition, the geographic distances of PCNs do not reflect the routing cost, while in wireless networks distances can be an estimate of routing difficulties. Hence, we propose to use **virtual coordinates in a Euclidean space** that reflect the PCN topology features.

Every PCN has multiple webservers as its interface of registration and user interactions. The webservers in WebFlow do **not** participate in routing and only support coordinates computation. Hence it avoids the landmark limitations in landmark-based routing [7], [8]. We assume these webservers are honest.

A. Coordinates computation

For a PCN $G = (V, E, \Psi)$ in a Euclidean space S, each node will be assigned a set of coordinates c^S by the webservers. The goal is to let nodes maintain coordinates such that two neighbors are relatively close in the Euclidean space. For two arbitrary nodes, their network distance (e.g. hopcount) would be proportional to their Euclidean distance. In this way, coordinate-based greedy routing is more likely to succeed.

The webservers randomly select k nodes $T = \{T_1, ..., T_k\}$ called *anchors* where $k = d + 1$ if we use a d-dimensional Euclidean space. [2] For each anchor, the servers recursively visit its neighbors, resulting in a spanning tree. Then the servers have k spanning trees and know the hopcounts between all pairs among the anchors. The servers should find a set of coordinates $c^S_{T_1}$,...,$c^S_{T_k}$, such that the Euclidean distance can reflect the hopcounts with minimal errors. We apply multi-dimensional scaling [24] to obtain the coordinates. Once the coordinates of anchors are determined, each node contacts the webservers to get the coordinates of anchors and their hopcounts to the anchors. Each node can determine its own coordinates by minimizing the overall error between actual hops and computed distances to these anchors. Since each user is responsible to compute its own coordinate, the system can be scaled to large sizes. Actually, the most time-consuming part in coordinate computation is initialization when the webservers build a spanning tree for each anchor and compute the coordinates of anchors. However, even for a real-world PCN topology generated from Ripple network with 1,870 nodes, it only takes 5 seconds to initialize. And it takes less than 10 ms for each user to generate its own coordinate.

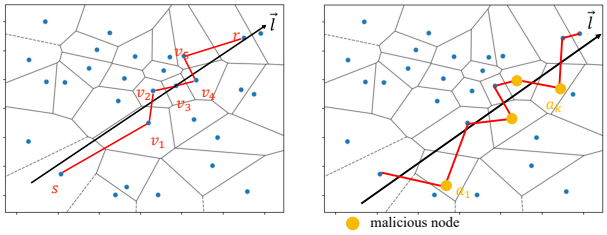
[2]For a d-dimensional Euclidean space, k needs to be at least $d + 1$ [23].

Fig. 4 is an example of a PCN in 2D space. Fig. 4(a) shows the network topology with randomly assigned coordinates. Fig. 4(b) shows the network after the WebFlow coordinate assignment. We find that the WebFlow coordinates show a better correlation between the Euclidean distances and hopcounts.

Note that the webservers do not carry payment traffic. They only work when new users join the system, providing users coordinates of anchors and users' hopcounts to anchors. Users and webservers send messages through a secure communication channel (traditional communication networks). Information leakages between them are beyond the scope of our discussion. The coordinates of users are determined by themselves. So webservers do not know the coordinates of any no-anchor users. After users join in, the routing protocol is executed in a decentralized manner.

Security of Webservers. We initially consider only passive, but still adaptive, corruption of a minority (less than half of the total set) of the webservers, which are thus assumed to be honest-but-curious. We assume that the non-corrupted webservers execute the coordinate computation according to our specifications and do not share private information among each user (i.e., they do not collude). Operations from the webservers in the protocol are confined to the computation of anchors' coordinates before the transaction protocol. The webservers could return wrong coordinates of anchors or user hopcounts to anchors when the user requests. But this would be detected by the user if he sends requests to multiple webservers and gets different values. If users notice that the return values of some webservers are different from most of the webservers they request, they could report this action and the reputation of those webservers will drop. Users tend to choose webservers with higher reputations. Therefore, malicious webservers would lose customers and (possibly) go out of business. We thus argue that it is in the interest of the webservers to follow the protocol in order to maintain the availability of their network.

Choice of dimensionality. One immediate question is how many dimensions of the Euclidean space we shall use to characterize the network topologies of PCNs. We use Principal Component Analysis (PCA) [25] to find an appropriate dimensionality. PCA relies on Singular Value Decomposition (SVD) [26]. The input of SVD is an $n \times n$ matrix M of the hopcounts between all nodes. SVD factors M into the product of three matrices $M = USV^T$, where S is a diagonal matrix with non-negative elements s_i, and $m_{ij} = \sum_{k=1}^{N} s_k u_{ik} v_{jk}$. If singular values $s_1, s_2, ..., s_d$ are much larger than the rest, we may approximate $m_{ij} \approx \sum_{k=1}^{d} s_k u_{ik} v_{jk}$, which means that we may approximate M using d-dimensional Euclidean spaces with low errors. We analyze two real-world PCN topologies: Ripple [27] and Lightning [2], whose details will be presented in Sec. VI. Fig. 5 shows the singular values of the two PCNs. We find the first three singular values of Ripple and the first two of Lightning are significantly larger than the rest. Hence, in WebFlow we use 3D space and show some comparison with



(a) A path by WebFlow-PE

(b) An example of malicious nodes on the path

Fig. 7. An example of WebFlow-PE on the distributed Voronoi diagram

2D and 4D in evaluation.

B. WebFlow Routing

WebFlow is based on a distributed data structure called Multi-hop Delaunay triangulation (MDT) [9]. An important feature of MDT is that, each node only needs to maintain the link and path information to a few nodes, independent of the network size. For a given set of nodes V in a Euclidean space, the Voronoi diagram [28] is a partition of space into cells such that each cell contains one node and the node is closer to any point in the cell than any other nodes. An example in Fig.6(a) shows the Voronoi diagram with 6 nodes. For all points in the Voronoi cell of node u, u is closer to them than any other node in V. A Delaunay triangulation, as shown in Fig.6(b), is a dual graph of the Voronoi diagram, where two nodes in V are connected if their Voronoi cells share an edge. Two nodes u and v are called *DT neighbors* if they are connected in the Delaunay triangulation of V, $DT(V)$. An important proven feature is that greedy routing on the DT edges (i.e., assuming all DT neighbors are connected) always succeeds to find the destination without encountering a local minimum [9].

However, in reality, not every DT edge is connected. As shown in Fig. 6(c), the DT edges uv and bc are not connected. An MDT is a distributed protocol to generate a distributed data structure such that: 1) each node knows its DT neighbors; 2) for a DT neighbor without a direct channel, MDT provides a multi-hop channel path to it. Hence greedy routing with an MDT can guarantee to find a path for a pair of nodes. This feature can be extended to d-dimension for $d \geq 2$.

We denote $MDT(V)$ as a 3-tuple $\{< u, N_u, F_u > | u \in V\}$, where N_u is the set of u's DT neighbors. u determines N_u by calculating a local DT of its direct neighbors. F_u is a soft-state forwarding table that stores the *virtual link* (multi-hop channel path) to the DT neighbors without direct channels. Note that in WebFlow, no node should maintain a global view of the MDT. Each node only maintains local information $< u, N_u, F_u >$ that is independent of the network size. The routing decisions are made locally. In addition, there are no supernodes that handle most payments such as landmark routing. Hence WebFlow is highly scalable and decentralized. **Given the destination coordinates, WebFlow routing using the MDT graph always finds a path to the destination based on the local decisions of the nodes on the path.**

The routing algorithms of WebFlow contain several MDT protocols including the forwarding protocol, join protocol, and maintenance protocol. The forwarding protocol determines how a node should locally decide the next-hop node when routing to a recipient in a correct MDT. The other protocols are used to maintain a correct MDT graph when nodes boot up or get offline dynamically. They are all decentralized algorithms.

Algorithm 1 WebFlow Forwarding Protocol at node u

Input: A payment sent to r with demand ω
Output: Next hop v

if $d(u,r) == 0$ **then**
  return u
//Case 1: Search for direct neighbors
if there exists $v | v \in C_u$ closest to the recipient **then**
  if $\psi_{uv} \geq \omega$ **then**
    return v
else
  for each node v in C_u **do**
    if $d(v,r) < d(u,r)$ and $\psi_{uv} \geq \omega$ **then**
      return v
//Case 2: Search for DT neighbors
if there exists $v | v \in N_u$ closest to the recipient **then**
  Probe each channel of the virtual link from u to v to obtain their capacity ψ_p
  Find the bottleneck capacity ω_{min} = min ψ_p
  if $\omega_{min} \geq \omega$ **then**
    return v
else
  i = 0
  for each node v in N_u **do**
    if $d(v,r) < d(u,r)$ and $i \leq 5$ **then**
      $i++$
      Probe each channel of the virtual link from u to v to obtain their capacity ψ_p
      Find the bottleneck capacity ω_{min} = min ψ_p
      if $\omega_{min} \geq \omega$ **then**
        return v
return ø

1) Forwarding Protocol: Consider a payment t initiated by node s that should be received by node r, and the payment value is ω. For each node u received forwarding request, if u is not the receiver, it first checks if there exists a direct neighbor v closest to the receiver and checks if the channel uv has enough capacity to support the payment. If the channel can support the payment, that is $\psi_{uv} > \omega$, u sends the payment to v. Otherwise, u finds the DT neighbor v' that is closest to r among all u's DT neighbors. Then, u needs to probe the virtual link to check if the underlying multi-hop path of this virtual link has enough capacity to support the payment. If yes, u sends the payment to v' using the virtual link.

If both situations fail due to channel capacity limitation, u traverses all of its direct neighbors to check if there exists a direct neighbor v closer to the receiver with enough channel capacity to support the payment. If such a direct neighbor does not exist, u selects 5 DT neighbors closer to the receiver. Note that u's DT neighbors that are closer to the receiver may be less than 5, in this case, u just selects all these satisfied DT

neighbors. Then u probes the virtual links to these selected DT neighbors until it finds a DT neighbor with a virtual link that can support the payment. If all these selected DT neighbors fail to fulfill the payment, we assume that this payment is failed. Note that if u selects too many DT neighbors to probe, it will introduce large probing overhead and communication costs. If u only probes a small number of DT neighbors, it will lower the success ratio. We analyze the topology generated from Ripple network and find that each user on average has 13 DT neighbors located in different directions. Considering the trade-off between communication cost and success ratio, we choose 5 DT neighbors to probe in our design. The pseudo-code of the forwarding algorithm at an intermediate node is shown in Algorithm 1.

Since large payments could easily use up some specific links [29], if a payment t exceeds a threshold, the sender randomly divides the large payment into several micropayments $t_1, t_2, ...t_k$, and assigns each sub-payment a random unique index. The sender treats these sub-payments as unrelated and independent payments. If the number of sub-payments received by the receiver is less than the number informed by the sender, we assume that this payment t is failed. To better preserve the value privacy, small payments are also assigned a random index. Thus, a malicious node seeing a payment going through it cannot determine whether it is the total value or just part of the value of a sub-payment. Since the index is randomly distributed, the adversary cannot estimate the number of sub-payments or the total value of the payment from the index. To mitigate the case that the adversary might deviate from the protocol to select a longer path, senders will set an upper bound for the total transaction fee for each transaction. If the transaction fee exceeds the pre-determined upper bound, the path is considered to be failed, and the adversary gets no profit.

2) New node joins: When a new node w boots up and wants to join the network, it first discovers its direct neighbors and assigns its coordinate. In order to get its coordinate, w needs to get the coordinates of anchors from the webservers. Then it sends hop-count queries to its direct neighbors and concludes its hop-count to each anchor. For example, assume the hop-count information to an anchor w collects from its direct neighbors is $\{h_1, h_2, ...\}$. w can deduce that its hop-count to this anchor is $min\{h_1, h_2...\} + 1$. With this information, w can determine its own coordinate locally. Then it sends join requests to its neighbors, and tries to find all of its DT neighbors. To begin its search, it must find at least one neighbor working correctly in the system, say v. Node w includes its coordinate in the join request and sends it to v. Now v can begin a greedy routing to get to node c which is closest to w. By the property of DT, the closest node to w in the Euclidean space must be a DT neighbor. So c is definitely w's DT neighbor. Then c sends a $JOIN\_rep$ back to w along the reverse path. After receiving the $JOIN\_rep$, w begins an iterative search to find other DT neighbors. After finding the paths to these DT neighbors, w locally stores the paths as virtual links. w also needs to locally maintain the direct neighbor set C_u and DT neighbor set N_u. Since w boots up to be their new neighbor, they also need to update their node sets C_u and N_u in the memory.

Algorithm 2 WebFlow-PE Forwarding Protocol at node u

Input: The line with a direction $\vec{l}$ and demand ω
Output: Next hop v

```
// Case 1: u is the recipient r
if u = r then
    return u
// Case 2: continue routing
for each edges e_i of u's Voronoi region do
    Find u_i whose Voronoi region sharing edge e_i
    if u_i is not the last hop then
        v = u_i
        Probe each channel of the virtual link from u to v to
        obtain their capacity ψ_p
        Find the bottleneck capacity ω_min = min ψ_p
        if ω_min ≥ ω then
            return v
        else
            return False
```

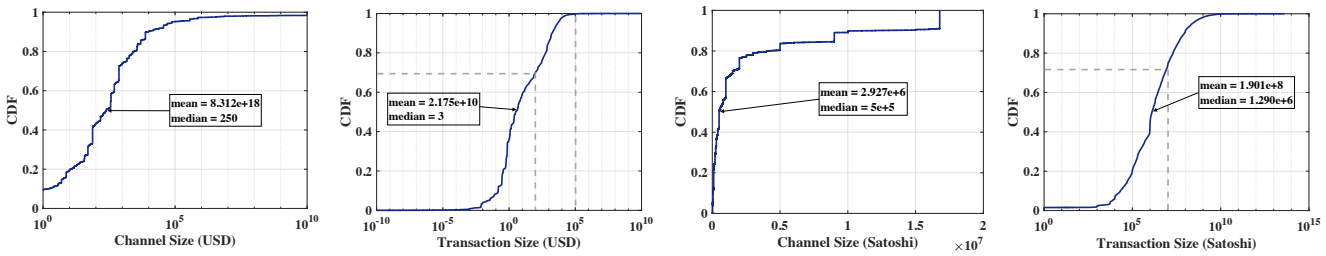
3) Node leaving and other changes: We now consider a maintenance protocol to deal with the situation when a node gets offline, new channels emerge, or some channels shut down. This protocol is designed to fix the structure of MDT. The MDT graph is correct only if every node knows all of its DT neighbors. So in WebFlow, each node u queries some of its neighbors to see if they know mutual neighbors that node u does not know, and then sends neighbor-set requests to them. If u discovers a new neighbor from neighbor-set replies, it will send a neighbor-set request to this new neighbor if they are vertexes in the same simplex in $DT(C_u)$. Every node runs this maintenance protocol locally, and every time when a node finishes running it, it will wait for a time period T_m until running it again. This helps to keep the MDT correct and guarantees that the forwarding protocol works well.

C. Limitations

While MDT-based WebFlow routing protocol provides a high success ratio and short forwarding paths for payment with low memory cost in the payment channel network, it does not achieve all the privacy goals defined in Section III. Since we use Euclidean distance to choose the node closest to the receiver, the coordinate of the receiver is exposed to all the nodes along the path as well as their DT neighbors and direct neighbors. Once the adversary controls some of these nodes, it can speculate what and where the sender and receiver are with some probability, although it is still not sure about the payment value. Even if we apply onion routing in WebFlow as Lightning network did [2], each intermediary still knows the coordinates of the immediately preceding and following nodes. Transactions may still be tracked by malicious intermediaries.

V. WebFlow-PE Design

In this section, we present the routing protocol of WebFlow-PE, a privacy enhancement version in WebFlow. The design of



(a) Channel Size Distribution of Ripple (b) Transaction Size Distribution of Ripple (c) Channel Size Distribution of Lightning (d) Transaction Size Distribution of Lightning

Fig. 8. Transaction dataset and channel size distribution used for real-world evaluations.

WebFlow-PE is consistent with MDT-based WebFlow except the forwarding part. We first provide a high-level overview and then provide a detailed protocol description.

A. WebFlow-PE Overview

Consider the aforementioned problem of the MDT routing in Section IV, we extend the basic design of WebFlow to hide the coordinates of all the nodes along the path and enhance user anonymity based on an innovation called *distributed Voronoi Diagram* [30]. As introduced in Sec. IV, the Voronoi Diagram is the dual graph of the DT. It is very easy for a node running WebFlow to know the boundaries of its Voronoi cell: simply computing the bisectors of the line segments to all its DT neighbors. If every node knows its Voronoi cell boundaries, a distributed Voronoi Diagram is maintained.

Instead of using the destination coordinates, we propose to only use a line with a direction $\vec{l}$ as the routing target. The sender s generates $\vec{l}$ by making it intersect an arbitrary point in the Voronoi cell of s and another arbitrary point in the Voronoi cell of the receiver r, with a direction to the later as shown in Fig. 7(a). The routing algorithm is that, at an intermediate node v, v always sends the payment to its DT neighbor whose Voronoi cell is the next Voronoi cell intersecting with $\vec{l}$ along $\vec{l}$'s direction. To prove that WebFlow-PE guarantees to find a path to r, we prove the following propositions.

Proposition 1. *Suppose the Voronoi cell of v, $VC(v)$, intersects with $\vec{l}$. The next Voronoi cell intersecting with $\vec{l}$ along $\vec{l}$'s direction is the Voronoi cell of v', denoted by $VC(v')$. Then v and v' are DT neighbors.*

Proof. Since $VC(v')$ is the Voronoi cell intersecting with $\vec{l}$ next to $VC(v)$, $VC(v)$ and $VC(v')$ must share a Voronoi edge intersecting with $\vec{l}$. By definition, two nodes sharing a Voronoi edge are DT neighbors. □

Proposition 2. *v knows a path to every DT neighbor of v.*

Proof. This is very easy to prove: 1) if v has a direct channel to its DT neihgbor v', it can send the payment to v' directly; otherwise 2) v' is a multi-hop DT neighbor and v can rely on MDT to get the path to v'. □

Proposition 3. *The routing of WebFlow-PE guarantees to reach the destination r.*

Proof. Let the Voronoi cells intersecting with $\vec{l}$ in a sequence $VC(s), VC(v_1), VC(v_2), ..., VC(v_k), VC(r)....$ Since $\vec{l}$ intersects with $VC(s)$ and $VC(r)$ by definition, $VC(s)$ and $VC(r)$ must exist in the above sequence. WebFlow-PE routing using $\vec{l}$ will visit $VC(s)$, $VC(v_1)$, $VC(v_2)$, ..., until reaching $VC(r)$ according to the routing algorithm. □

How r confirms that it is the receiver of the payment. s and r need to first exchange a common secret for this transaction such as a transaction key k, using classic secure Internet communication such as TLS. In addition to $\vec{l}$, the sender will be a hash value $H(k)$ to the payment routing, where H is a cryptography hash function. When r receives a payment, it will compare $H(k)$ to the hash of its transaction key and confirm that $\vec{l}$ intersects with $VC(s)$ and $VC(r)$. After that, it can keep the payment and stop forwarding.

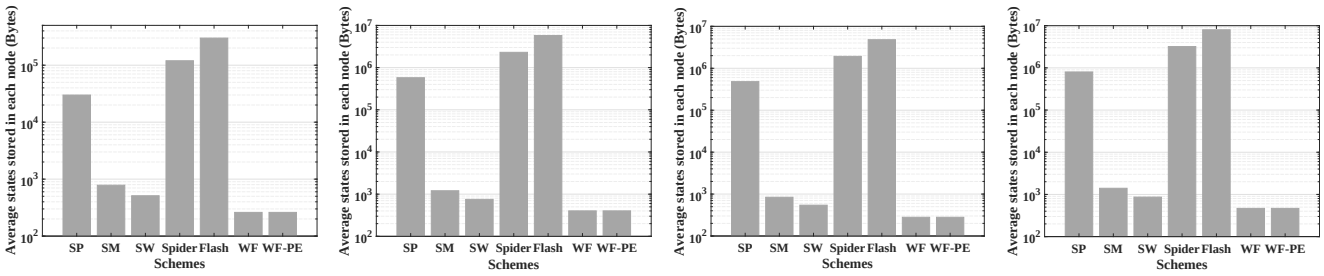
For each node along the path, it does not know the coordinates of the sender or the receiver. The only information it could obtain is $\vec{l}$. Each node can determine which DT neighbor is the next hop according to the direction function. Besides, we also need to consider the capacity of the channel. After a node determines the next hop DT neighbor, it needs to check if the channel or multi-hop path has enough capacity to support the payment. If not, the payment fails. We show the pseudo-code of the forwarding protocol of WebFlow-PE at each node u along the path in Algorithm 2.

Different from MDT-based routing protocol, we first find a path for the payment and then probe the path to see if it has enough capacity to support the payment. Since the path is pre-determined, and it is static routing, it is more likely to fail than MDT-based routing protocol which is dynamic routing that combines the probing and path finding process at the same time. Note that the only difference between MDT-based WebFlow and WebFlow-PE is the forwarding protocol, and all the other designs keep the same. So the users of WebFlow can easily switch between these two routing protocols based on their demands, higher success ratio, or better privacy.

VI. Performance Evaluation

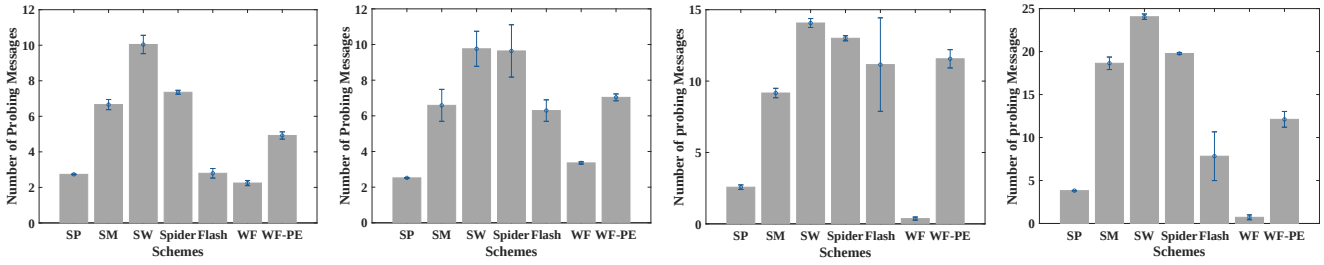
In this section, we evaluate the performance of WebFlow compared to the existing off-chain routing algorithms, using **both simulation and prototype implementation**. The evaluation aims to answer the following research questions:

- How does the WebFlow routing perform with regard to success rate, success volume, and overhead under realistic PCN topologies and traces?



(a) Ripple (b) Lightning (c) Waxman (d) Scale-free

Fig. 9. Storage cost compared with baseline methods.



(a) Ripple (b) Lightning (c) Scale-free (d) Waxman

Fig. 10. Communication cost compared with baseline methods.

- How does network load affect WebFlow's performance?
- How do these results compare to the performance of other approaches?

A. Methodology

We study WebFlow with two real-world PCNs: Ripple [27] and Lightning (LN) [2], as well as synthesis topologies in simulation. For Ripple, we use the data from January 2021 to December 2021, and get the network topology with 1,783 nodes and 18,395 edges in our simulation. For Lightning network, we get the network topology with 3,519 nodes and 47,311 edges on one day in January 2022. Since Lightning network preserves the privacy of link balances, we only get the range of the link balance and evenly assign funds over both directions of a link.

We generate payments by randomly sampling the Ripple transactions for the Ripple topology. Due to the lack of sender-receiver information in the trace of the Lightning network, we randomly sample the transaction volumes and sender-receiver pairs. The distribution of transaction sizes is shown in Figure 8. 70% of the transactions in Ripple is less than 100$, and 70% of the transactions in Lightning are less than 10^7 Satoshi. So we treat payments less than 100$ and 10^7 Satoshi as small payments in Ripple and Lightning respectively. We assume that payments arrive at senders sequentially. Concurrent payments will be considered in future work.

According to the observation of these two real-world topologies, we additionally build two sets of synthesis PCN topologies based on Waxman topology generation [31] and scale-free network model [32]. The link balances are assigned similarly to those of Ripple. The payments are also generated by mapping the Ripple transactions to the simulated topologies.

Baseline methods. We compare both routing methods of WebFlow - MDT-based WebFlow (WF) and WebFlow-PE (WF-PE) - with the following off-chain routing algorithms.

SilentWhispers (SW) [7]: A landmark routing algorithm that nodes always send funds to landmarks, and rely on landmarks to find the path. We set the number of landmarks to 3.

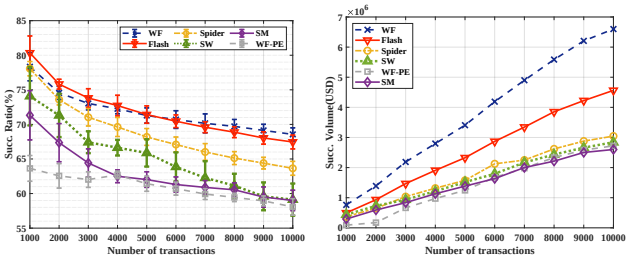
SpeedyMurmurs (SM) [8]: An embedding-based routing algorithm that relies on assigning coordinates to nodes to find shorter paths with reduced overhead. We set the number of landmarks to 3 as [8] suggests.

Spider [6]: An off-chain routing algorithm that considers the dynamics of link balance. It balances paths by using those with maximum available capacity, following a waterfilling heuristic. It uses 4 edge-disjoint paths for each payment.

Flash [5]: An off-chain routing algorithm that differentiates the treatment of elephant payments from mice payments. We set the number of preserved paths for each receiver in mice payment routing to 4, and the number of preserved paths for elephant routing to 20. The elephant-mice threshold is set such that 90 % of payments are mice.

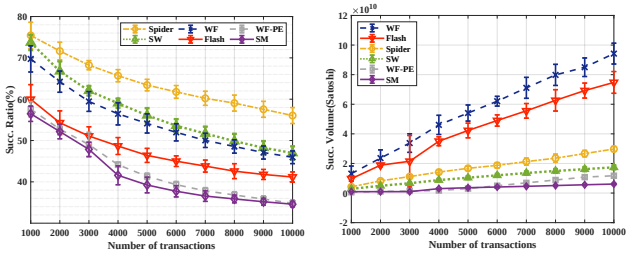
Shortest-Path(SP): Baseline algorithm. SP uses the path with the fewest hops between senders and receivers in routing.

Metrics. We use communication and storage costs as the primary metrics for scalability. Similar to prior work [5], [8], we also use success rate and success volume as evaluation metrics for resource utilization. Besides, we evaluate the anonymity of the system. The success rate is defined as the percentage of successful payments whose demands are met overall generated payments. The success volume describes the total size of all successful payments. Before sending payments, nodes need to probe the usable capacity of the candidate paths. The number of probe messages describes the communication cost, which is the probing overhead. The storage cost is computed by the average number of distinct neighbors a node



(a) Succ. rate (b) Succ. volume

Fig. 11. Performance with varying transaction numbers in Ripple.



(a) Succ. rate (b) Succ. volume

Fig. 12. Performance with varying transaction numbers in Lightning network.

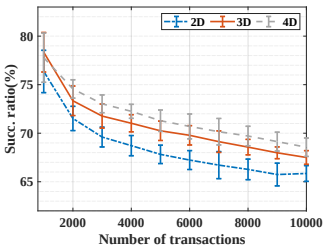


Fig. 13. Routing succ. rate in Ripple

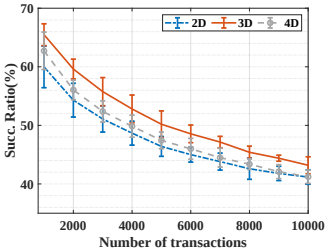


Fig. 14. Routing succ. rate in LN

needs to know (and store) to perform routing, including the coordinates of its neighbors and the information of its related links. We report the average results over 5 runs.

B. Overall Performance and Overhead

Storage cost in each node. We now show the storage efficiency of WebFlow by comparing the average states maintained in each node. In Shortest Path, Spider and Flash, every node needs to locally store the network topology, including all the information of the links. Besides, in Flash, each node needs to maintain a routing table for each payment, and periodically refreshed it when the local network topology updates. In SilentWhispers, the landmarks need to store the network topology as well as the paths to all the nodes. Each node only maintains the paths to the landmarks. For SpeedyMurmurs, the coordinate is assigned according to the landmarks, and the length of the coordinate is depending on the depth of the node in the spanning tree. Since there are always several landmarks in the system, each node has to store several coordinates. Different from these schemes, in WebFlow, nodes only need to maintain the information of their neighbors, including the coordinates and the links to them. Figure 9 shows the average states maintained in each node. For both Ripple and Lightning, both versions of WebFlow cost less than other routing algorithms.

Communication cost. We evaluate communication costs to see if our algorithms can achieve low overhead of routing. The communication cost is computed as the total number of probing messages sent over the network. Since SilentWhispers, SpeedyMurmurs, Shortest Path, and WebFlow-PE are static routing schemes, we consider the number of probing messages the same as their path length. Spider and Flash use multiple paths. They first select several paths and make payments after probing the path. Moreover, Flash only probes a path when it cannot deliver the payment in full. So the number of probing messages along a path is proportional to the number of hops on the path in Spider and Flash. MDT-based WebFlow only does probing if a node needs to reach a multi-hop DT neighbor. In this case, the node will probe the path to its DT neighbor to see if the path has enough capacity to support the payment. Figure 10 shows the comparison results with 1000 transactions. The results here demonstrate that WebFlow indeed efficiently reduces the probing message overhead compared to state-of-the-art in all of the four topologies.

TABLE I
MESSAGE FORMAT IN OUR PROTOTYPE.

Field	Description
TransID	A unique ID of a (partial) payment
Type	Message type and routing scheme
Direction	Routing direction of this message
Capacity	Probed link capacity
Commit	Total funds for this payment

Performance with different network load. We also vary the number of transactions to test the performance of our system with different loads. With the increase in the number of transactions, the success rate of all schemes decreases in both Ripple and Lightning topologies as shown in Figure 11 and 12. This is because as more transactions flow into the network, more links are saturated in one direction, making them cannot be used for future transactions. In most situations, MDT-based WebFlow shows a higher success rate and success volume compared to other methods. The only exception is for the success rate of Lightning, where WebFlow has a lower success rate than Spider.

Choice of Dimensionality. We perform experiments for ripple and lightning networks embedded in 2D, 3D, and 4D virtual spaces and evaluate the performance in terms of success rate. Figure 13 and Figure 14 show the results of routing performance for the two network topologies respectively. For both topologies, 3D outperforms 2D. For 4D, the results are not much better than those of 3D in ripple, and even have a lower success rate in lightning. This observation is consistent with the PCA results in Fig. 5. Besides, in a 4-dimensional space, since nodes have more DT neighbors to maintain and have to keep the coordinates of 4-tuples, both the storage and communication costs will increase compared to 2D and 3D. Hence, we choose 3D in all other experiments.

C. Testbed Evaluation

We conduct a testbed evaluation to further investigate WebFlow's performance. We implement the prototype in Golang with TCP for network communication. The prototype first generates the network topology and assigns coordinates

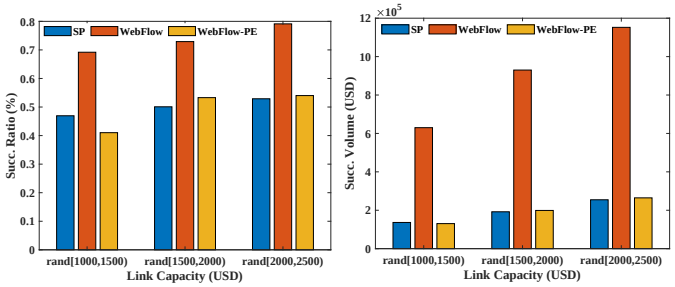


Fig. 15. Testbed results of the 50-node network.

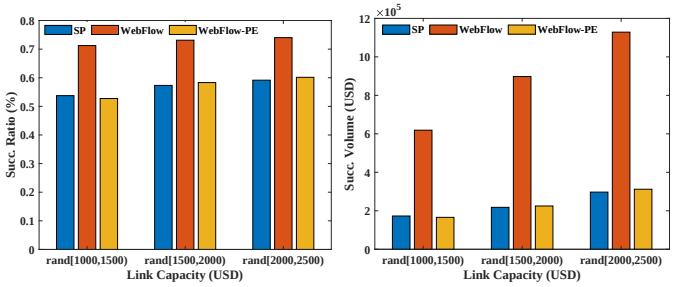


Fig. 16. Testbed results of the 100-node network.

to each node at launch time. Upon seeing a new payment, it runs the routing algorithm and sends it out. we represent each node of the PCN as a single process running in the WebFlow prototype. We build a PCN topology based on Waxman topology generations [31]. The routing algorithm includes three functions: distributed routing, probing, and commit. We describe the details in the following.

Distributed Routing. Each node is responsible for finding the next hop based on the received message. We show the message format of our prototype in Table I. Type field shows the message type, and also includes 1 bit indicating the scheme of routing. The destination field contains the coordinates of the recipient in the MDT-based WebFlow scheme, while shows the direction function in the WebFlow-PE scheme. Upon receiving a ROUTE message, a node can get routing direction from this field and forward it to the next hop according to the Type field. Besides, the nodes need to record TransID, last-hop, and next-hop locally for further commitment. Recipient returns ROUTE_ACK if path found, and ROUTE_NACK if path not found or with insufficient capacity. In the MDT-based WebFlow scheme, the nodes on the reversed path will replace the direction field with the coordinate of its last hop and forward the modified message to it. In the WebFlow-PE scheme, the recipient simply replaces the direction field with a reversed direction function.

Probing. It is used in the MDT-based WebFlow for nodes to collect the ever-changing link balance to determine which link to choose for the next hop. In MDT-based WebFlow, probing only takes place when a node finds that a candidate next hop is a DT neighbor. It then probes the underlying physical path of the virtual link to check if it can support the payment. This node initiates probing by constructing a PROBE message for the virtual link and initiates the Capacity field to the payment value. The intermediate nodes compare their current balances with the Capacity field. If their balances are less than the value in the Capacity field, they replace the Capacity field with their current balances. Otherwise, they do nothing. After receiving the probing message, the node simply compares the value in the Capacity field with the value in the Commit field, and determines whether this virtual link can support this payment.

Commit. It is the step to commit the payment. After finding a path for the payment, the sender sends a COMMIT message in the path. All the intermediate nodes receiving the message search for the next hop according to the TransID stored locally. Then they update their balances according to the COMMIT field and forward the message to the next hop. If an intermediate node finds that its balance is less than the payment amount, it constructs a COMMIT_NACK message sent back to the sender. All the nodes that receive COMMIT_NACK messages will then recover their balances.

Figure 15 and 16 show the performance with different link capacities. The success rate and success volume of MDT-based WebFlow are much higher than Shortest Path in both two topologies with a different number of nodes. This is because we consider link capacities when routing. The results demonstrate the effectiveness of MDT-based WebFlow which selects a good path to improve throughput. However, the performance of WebFlow-PE is similar to Shortest Path. This is because both algorithms first select a path without considering link capacities and then check if the path can satisfy the payment. It is still acceptable since we achieve better anonymity at the cost of lower performance in WebFlow-PE.

VII. Conclusion

In this work, we present the design of a scalable and decentralized routing solution called WebFlow for large and dynamic PCNs. WebFlow includes two protocols: MDT-based WebFlow and WebFlow-PE. The first one provides a high success rate and success volume of payments. The second one, the privacy enhancement version of WebFlow, achieves destination anonymity by using routing with a distributed Voronoi diagram. Both protocols demonstrate low per-node cost and high network resource utilization. The evaluation results using simulations and prototype implementation demonstrate that WebFlow significantly outperforms existing solutions, especially on per-node cost efficiency, while maintaining high resource utilization and success rate.

Acknowledgment

The authors were partially supported by NSF Grants 1750704, 1932447, and 2114113. C. Qian was partially supported by the Army Research Office and was accomplished under Grant Number W911NF-20-1-0253. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. We thank the anonymous reviewers for their comments.

REFERENCES

[1] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer *et al.*, "On scaling decentralized blockchains," in *Proceedings of Springer FC*, 2016.

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

[3] "Transaction rate of bitcoin," http://www.blockchain.com/en/charts/transactions-per-second, 2020.

[4] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of ACM SOSP*, 2017.

[5] P. Wang, H. Xu, X. Jin, and T. Wang, "Flash: efficient dynamic routing for offchain networks," in *Proceedings of ACM CoNEXT*, 2019.

[6] V. Sivaraman, S. B. Venkatakrishnan, K. Ruan, P. Negi, L. Yang, R. Mittal, G. Fanti, and M. Alizadeh, "High throughput cryptocurrency routing in payment channel networks," in *Proceedings of USENIX NSDI*, 2020.

[7] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Silentwhispers: Enforcing security and privacy in decentralized credit networks." in *Proceedings of USENIX NDSS*, 2017.

[8] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," in *Proceedings of USENIX NDSS*, 2017.

[9] S. S. Lam and C. Qian, "Geographic routing in d-dimensional spaces with guaranteed delivery and low stretch," in *Proceedings of ACM SIGMETRICS*, 2011.

[10] "Lightning network daemon," https://github.com/lightningnetwork/lnd.

[11] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian journal of Mathematics*, vol. 8, pp. 399–404, 1956.

[12] P. Prihodko, S. Zhigulin, M. Sahno, A. Ostrovskiy, and O. Osuntokun, "Flare: An approach to routing in lightning network," *White Paper*, p. 144, 2016.

[13] H. Xue, Q. Huang, and Y. Bao, "Epa-route: Routing payment channel network with high success rate and low payment fees," in *Proceedings of IEEE ICDCS*, 2021.

[14] Y. Ohara, S. Imahori, and R. Van Meter, "Mara: Maximum alternative routing algorithm," in *IEEE INFOCOM 2009*. IEEE, 2009, pp. 298–306.

[15] H. Nagamochi and T. Ibaraki, "Computing edge-connectivity in multigraphs and capacitated graphs," *SIAM Journal on Discrete Mathematics*, vol. 5, no. 1, pp. 54–66, 1992.

[16] F. F. E. Dabek, "A distributed hash table," Ph.D. dissertation, Massachusetts Institute of Technology, 2005.

[17] "Raiden network," http://https://raiden.network/, 2020.

[18] X. Zhang, S. Shi, and C. Qian, "Webflow: Scalable and decentralized routing for payment channel networks with high resource utilization," *arXiv preprint arXiv:2109.11665*, 2021.

[19] C. Diaz, S. Seys, J. Claessens, and B. Preneel, "Towards measuring anonymity," in *Proceedings of PET*, 2002.

[20] A. Serjantov and G. Danezis, "Towards an information theoretic metric for anonymity," in *Proceedings of PET*, 2002.

[21] L. Zhuang, F. Zhou, B. Y. Zhao, and A. Rowstron, "Cashmere: Resilient anonymous routing," in *Proceedings of USENIX NSDI*, 2005.

[22] B. Karp and H. Kung, "Greedy Perimeter Stateless Routing for Wireless Networks," in *Proceedings of ACM Mobicom*, 2000.

[23] T. E. Ng and H. Zhang, "Predicting internet network distance with coordinates-based approaches," in *Proceedings of IEEE INFOCOM*, 2002.

[24] I. Borg and P. J. Groenen, *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[25] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[26] M. E. Wall, A. Rechtsteiner, and L. M. Rocha, "Singular value decomposition and principal component analysis," in *A practical approach to microarray data analysis*. Springer, 2003, pp. 91–109.

[27] F. Armknecht, G. O. Karame, A. Mandal, F. Youssef, and E. Zenner, "Ripple: Overview and outlook," in *Proceedings of Springer TRUST*, 2015.

[28] S. Fortune, "Voronoi diagrams and delaunay triangulations," *Computing in Euclidean geometry*, pp. 225–265, 1995.

[29] L. Eckey, S. Faust, K. Hostáková, and S. Roos, "Splitting payments locally while routing interdimensionally." *IACR Cryptol. ePrint Arch.*, 2020.

[30] P. Bose and P. Morin, "Online routing in triangulations," in *Proceedings of Springer ISAAC*, 1999.

[31] B. M. Waxman, "Routing of multipoint connections," *IEEE journal on selected areas in communications*, vol. 6, no. 9, pp. 1617–1622, 1988.

[32] N. Developers, https://networkx.github.io/documentation/networkx-1.9.1, 2014.