UNIVERSITY OF CALIFORNIA
SANTA CRUZ

REFINING THE DESIGN OF BLOCKCHAIN FOR SECURE DISTRIBUTED SYSTEMS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

Xiaoxue Zhang

June 2024

The Dissertation of Xiaoxue Zhang
is approved:

Prof. Chen Qian, Chair

Prof. Alvaro Cardenas

Prof. Ioannis Demertzis

Peter Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Refining the Design of Blockchain for Secure Distributed Systems

by

Xiaoxue Zhang

In our interconnected world, the need for robust distributed systems is undeniable. Such systems must guarantee security and privacy while managing data across multiple users, making them vital for industries like finance, the Internet of Things, and healthcare that handle sensitive information. The challenge lies in building these systems to withstand diverse threats and scale effectively. The advent of blockchain technology, which is a decentralized ledger that collaboratively stores and manages transactions with consensus in a distributed manner, has provided new solutions to these challenges. It offers a way to securely and transparently handle data and transactions with its inherent security features and consensus-driven operations. Despite its promise, blockchain faces scalability limitations that impede its widespread deployment in large-scale systems. Interoperability is another challenge that hinders its application in heterogeneous network environments. It requires the community to fundamentally rethink the current design of blockchains. To tackle these issues, this thesis takes a deep look at reshaping the landscape of the Blockchain and exploring the potential integration of Blockchain with distributed systems in the following themes:

- Refining the design of layer-2 blockchain, off-chain networks, to avoid the imbalanced fund utilization problem restricted by the nature of the per-channel deposit.

- Enabling interoperability in existing off-chain networks to support various cryptocurrencies trading with flexibility.
- Integrating Blockchain with federated learning for decentralized model verification and auditing.

Acknowledgments

First, I would like to gratefully and sincerely thank my advisor, Prof. Chen Qian, for his guidance, patience, and understanding during my PhD studies. This dissertation would not have been possible without the academic freedom and support he has given me during the last five years. His insights and feedback were invaluable, and his encouragement was a constant source of my motivation.

Next, I want to thank the entire Prof. Chen Qian's group, both current and previous members. During my doctorate, I am grateful to have worked with these talented lab mates on multiple topics. I especially thank Shouqian Shi for his effective advice in the design of WebFlow and guidance in quantum networks, Yifan Hua for the great help on the discussion and design of BDFL, and Minmei Wang for the guidance on the IoT topic. I am grateful for the advice from Prof. Song Han in WebFlow and APCN. I also thank Prof. Dejun Yang for his reviews and suggestions on APCN and XHub.

I would also like to thank members of my thesis reading committee and my PhD qualifying exam, Prof. Alvaro Cardenas, Prof. Ioannis Demertzis, and Prof. Hamid Sadjadpour, for their insightful suggestions for my dissertation.

Finally, I want to express my special thanks to my family for their love and support, and to all my friends for making my life colorful. Thank you all for being part of my life!

Chapter 1

Introduction

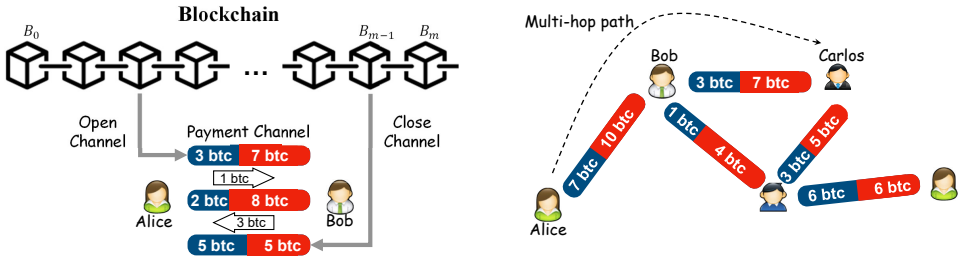
In this chapter, we will introduce the blockchain and its design deficiency. It highlights the design challenges of adopting Blockchain techniques in real-world distributed applications. We will then give an overview of the research presented in this thesis and summarize our contributions.

1.1 Background

Over the past decades, distributed systems have become increasingly crucial. They are designed to achieve reliability, efficiency, and scalability, spreading the data storage and processing to multiple heterogeneous devices. This structure allows for high availability and fault tolerance, as the failure of one node does not affect the entire system. Decentralized ledgers are distributed databases that collaboratively store and manage transactions with consensus in a distributed system. Each transaction is an event that happened in the system and involves one or more users of the system. A

typical transaction when decentralized ledgers are discussed is a payment from one user to another. But transactions have many other more generalized instances, such as a message sent from one user to another, an agreement among multiple users, and data publishing [52]. Blockchain is a promising solution for decentralized digital ledgers. Since Bitcoin was invented in 2008 [94], there have been many other decentralized ledgers emerging based on blockchains, such as Ripple [47], Stellar [46], and Ethereum [45]. While blockchains have shown great success, its scalability on throughput remains a huge problem with growing numbers of users and transactions [32, 102]. For instance, Bitcoin can only support 10 transactions per second at peak in 2020 [7]. In contrast, some widely used centralized payment hubs such as Visa and MasterCard can process more than 65,000 transaction messages per second as of June 30, 2019 [4]. Also, it is not suitable for micropayments owing to the high transaction cost. 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. Whenever a new block arrives, all nodes in the network have to process it and update the state of the blockchain. Hence using blockchains as a global transaction system for massive users is impractical at this moment.

There are some improvements for blockchain throughput, such as Bitcoin-NG [41] and Monoxide [125]. However, their performance is limited by the processing capacity of the nodes and network bandwidth, and thus, cannot be used as large-scale transaction systems. The recently proposed concept of off-chain networks, also known as payment channel networks (PCN) [50, 102], provides a high-throughput solution for



(a) Users set up channels on the blockchain **(b) Users and their channels form an offchain network**

Figure 1.1: An example of users setting up channels on a blockchain and forming an off-chain network.

blockchain-based decentralized ledgers. In off-chain networks, two users can conduct 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 [6]. 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. Each user can establish channels with multiple other users. However, the channel does not always exist between two arbitrary users. In such cases, 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. The off-chain network is a promising solution to increase the scalability of blockchains because most transactions can be achieved in an off-chain manner.

Figure 1.1 shows an example of an off-chain network. As shown in Fig. 1.1(a), two users, Alice and Bob, can set their channels by each putting certain funds into the channel and sending an "Open Channel" transaction to the blockchain. Then,

Alice and Bob can adjust the distribution of their channel funds without updating the blockchain. When they want to close the channel, they will let the blockchain store a "Close Channel" transaction with the signatures from both parties. For a network of users, their channels can form an off-chain network or PCN, as shown in Fig. 1.1(b). If Alice wants to make a payment to Carlos, to whom she does not have a channel, Alice can first move some funds to Bob using their channel and then Bob moves the same funds to Carlos using the channel between Bob and Carlos. These payments do not need to be recorded in the blockchain, and hence, the throughput bottleneck is avoided.

Blockchains can be extended to applications other than financial payments, including machine learning, the Internet of Things, and supply chain management. Machine learning, which requires secure and authentic model data sharing; IoT, needing robust security for vast data generated by interconnected devices; and supply chain management, which demands transparent and tamper-proof tracking, all stand to potentially benefit from blockchain technology.

1.2 Research Challenges and Contributions

While using off-chain networks is a promising solution to the scalability problem of decentralized ledgers, we argue that existing solutions of off-chain networks have several major limitations.

1. High throughput and secure utilization cannot be achieved simultaneously by the nature of per- channel deposit. To achieve high system

throughout, a core problem of an off-chain network is routing, i.e., finding a path between two arbitrary users. Centralized routing always assumes every user knows the entire network topology, including all nodes, channels, and channel balances, which is not a scalable solution since each user needs a large memory space and communication cost to store, receive, and broadcast the network information including every change on any link. Existing distributed routing methods for off-chain networks also have scalability bottlenecks as well as a crucial limitation: they cannot effectively utilize the channel resource in a network. Besides, off-chain networks require a separate deposit for every channel and significant locked-in funds from users [86]. Moreover, funds are not equally distributed among all the channels of one user. A situation might happen when a user cannot support a transaction due to insufficient funds in a required channel, but the node actually has sufficient unused funds in other channels. Redistributing funds among channels immediately is not realistic here, because users need to react with blockchain to set up new channels, which is time-consuming. Such inflexibility of fund utilization results in significant resource under-utilization.

2. Current off-chain network designs do not support secure multi-chain ledgers. Despite the growing number of different blockchains, cryptocurrencies continue to operate in complete isolation from one another. Hence, an off-chain network needs to support users from multiple blockchains, forming multi-chain ledgers. This requirement is also called "interoperability". Users in different blockchains may rely on the relay hubs to perform token exchanges and forward their transactions. One important security requirement is that we have to make sure every user, including

relay hubs and clients who followed the protocol, will not lose funds when malicious participants exist.

3. Integration of blockchains with various real-world applications. Blockchain, as a general-purpose architecture for decentralized computing, has far broader applications than just cryptocurrencies. However, currently, the dominant application of blockchain is still cryptocurrency, which is far below what the blockchain has promised. In order to tailor the blockchain to satisfy the needs of real-world applications, there are several significant issues to be considered. Firstly, identifying the specific problem that blockchain is intended to solve, such as decentralization, security, or accountability. Secondly, the type of blockchain to be used, public, private, or consortium, based on the requirement of the application. Thirdly, what information is to be stored on the blockchain. Finally, how to achieve security and privacy.

Taking the aforementioned issues into consideration, the major contributions of this dissertation are outlined in the following contents.

Scalability. We propose WebFlow, a scalable and 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 further propose Aggregated Payment Channel Networking (APCN), a novel design of payment channel networks with shared funding that could improve the success ratio of multi-hop payments and avoid locked-in funds in channels as well.

Interoperability. We propose XHub, a cross-chain off-chain network that allows two arbitrary users in different blockchains to make transactions via a multi-hop

path, without the need to track other blockchains.

Integration. We propose BDFL, a Blockchain-based Decentralized Federated Learning, that leverages blockchain's inherent decentralization, transparency, and security. It eliminates the need for a centralized aggregator, thereby enhancing privacy and robustness in Federated Learning.

1.3 Dissertation Organization

The rest of this dissertation is organized as follows. In Chapter 2, we present WebFlow, a scalable and distributed routing solution for off-chain networks. In Chapter 3, we provide APCN, a novel design of payment channel networks with shared funding. In Chapter 4, we propose XHub, a cross-chain payment channel network. In Chapter 5, we give an example of the integration of blockchain and real-world application, a Blockchain-based Decentralized Federated Learning. Chapter 6 provides the overall summary and conclusions of the dissertation.

Chapter 2

WebFlow

2.1 Introduction

While blockchains have shown great success as decentralized digital ledgers, *scalability* remains a huge problem with growing numbers of users and transactions [32, 102]. The recently proposed concept of *off-chain networks*, also known as *payment channel networks* (PCN) [50, 102, 140] 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.

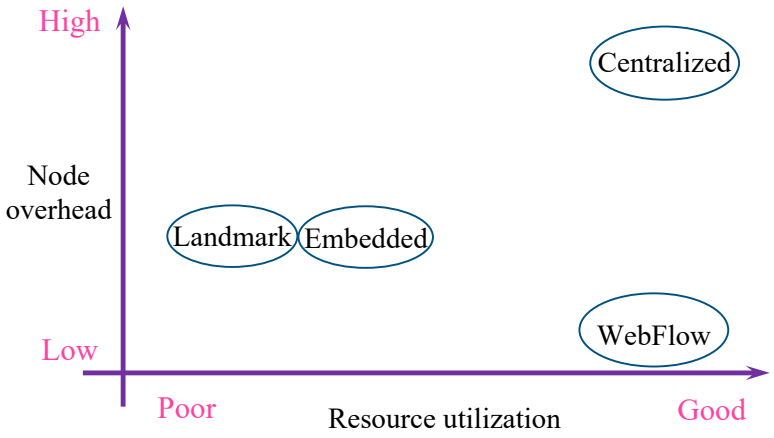


Figure 2.1: Trade-off of PCN routing

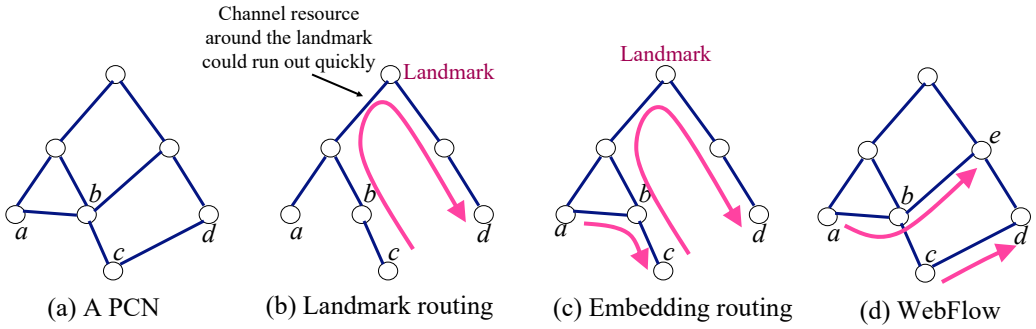


Figure 2.2: Example of different routing methods for PCNs.

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 [118,126]. *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 [84, 110]. It is ideal for large-scale PCNs.

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. 2.2(a). One typical distributed routing for PCNs is landmark routing [84] as shown in Fig. 2.2(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 [110]. 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. Hence, it is possible to utilize some non-tree links in a subtree, as shown in Fig. 2.2(c). 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 work, 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. 2.1, WebFlow could provide significantly lower

per-node overhead compared to centralized, landmark, and embedding routing while achieving similar 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) [73] 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 work 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 2.2 introduces the related work. The system overview and model are presented in Section 2.3. We describe the detail design of the WebFlow protocols in Section 2.4 and Section 2.5. Section 2.6 presents the evaluation results of WebFlow. Section 2.7 summarizes our conclusions.

2.2 Related work

PCNs provide a high-throughput solution for blockchains [50]. In PCNs, channels are not always existing between two arbitrary users, and two users can make transactions via a multi-hop path. Hence, routing becomes a challenging problem in PCNs. In the Lightning Network [102], each node locally maintains the network topology and a global routing table. In current implementations, source routing is utilized such as shortest path and max-flow routing algorithm. Max-flow routing always achieves a high success ratio of transactions. However, it has three practical problems:1) high per-node storage cost; 2) high per-transaction computation cost; 3) every update of any channel will be broadcast to all nodes. Hence it leads to scalability problems.

Recent work such as Spider [118] and Flash [126] use centralized routing. Spider actively accounts for the cost of channel imbalance by preferring routes that rebalance channels, and it proposes a centralized offline routing algorithm to maximize the success volume of payments. But the 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 [84] utilizes landmark routing. It performs a periodic breadth-first search to find the shortest path from the landmarks to users. All paths need to go through the landmarks, which makes the channels around the landmarks over-used while other channels under-used, and some paths could be unnecessarily long. SpeedyMurmurs [110] 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 [134] uses decentralized design. It leverages MA ordering [93, 98] 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 (DHTs) cannot be used for PCNs because one cannot be forced to build a channel with an arbitrary user. In DHT, a node picks its neighbors according to a certain structure and mapping rules. But these neighbors might not be its direct neighbors in the PCN, and the paths to these neighbors might be long, resulting in a high routing stretch. Kademlia [87] enforces a structured topology. However, in reality, as users can set up and shuttle down channels anytime, the topology is unstructured and keeps changing.

There are many existing Ad-Hoc routing protocol such as DSR, AODV and

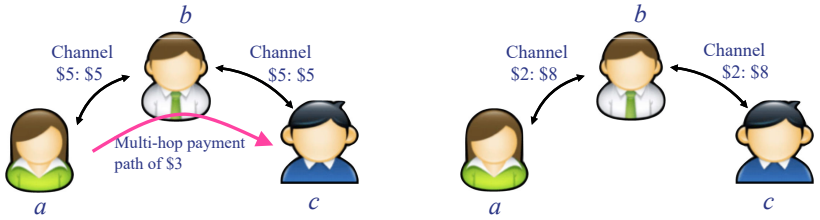


Figure 2.3: A multi-hop payment in a PCN.

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 is kept 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.

2.3 Background and Overview

2.3.1 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 [6, 141]. 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 via a multi-hop path, where any two consecutive users on the path share a channel. For example, in Fig.2.3(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.2.3(b), while b may charge a nominal transaction fee.

2.3.2 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) [2]. 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.

2.3.3 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 [110], 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

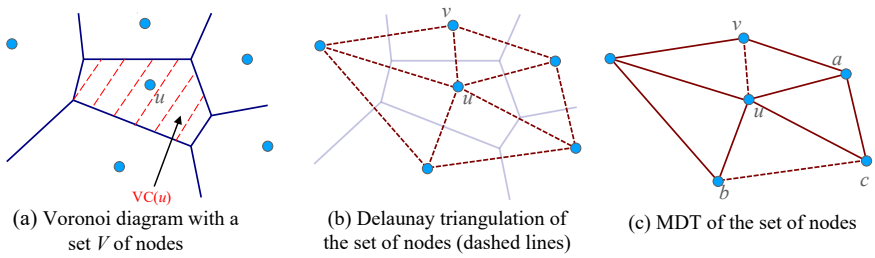
anonymity if the adversary cannot determine the original sender or the actual recipient of a transaction.

We *formally define a metric to evaluate the sender/recipient anonymity* of a network and its routing algorithm as follows. The anonymity measure follows the anonymity definition that has been used for anonymous routing such as [35, 114, 145]. 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:

$$\mathbb{A} = \frac{-\sum_{u \in V} p_u log_2(p_u)}{\log_2(|V|)} \tag{2.1}$$

2.3.4 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.



(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

Figure 2.4: Voronoi diagram, Delaunay triangulation (DT), and MDT

2.4 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 [67, 73]. 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. One problem is a node could be a local minimum. This problem can be solved in various ways [67, 73]. 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.

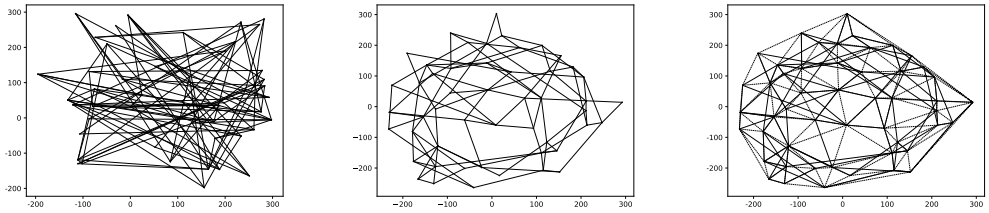
Every PCN has multiple webservers as its interface of registration and user interactions. The webservers in WebFlow do **not** participate in routing and only support coordinates computation. Hence it avoids the landmark limitations in landmark-based routing [84, 110]. We assume these webservers are honest.

2.4.1 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 web-servers. 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. [1] 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 [20] 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 web-servers 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

[1] For a d-dimensional Euclidean space, k needs to be at least $d + 1$ [95].



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

Figure 2.5: Original PCN graph, the graph after node positioning, and MDT graph.

each user to generate its own coordinate.

Fig. 2.5 is an example of a PCN in 2D space. Fig. 2.5(a) shows the network topology with randomly assigned coordinates. Fig. 2.5(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) [131] to find an appropriate dimensionality. PCA relies on Singular Value Decomposition (SVD) [124]. 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 [13] and Lightning [102], whose details will be presented in Sec. 2.6. Fig. 2.6 shows the singular values of the two PCNs. We find the first three singular

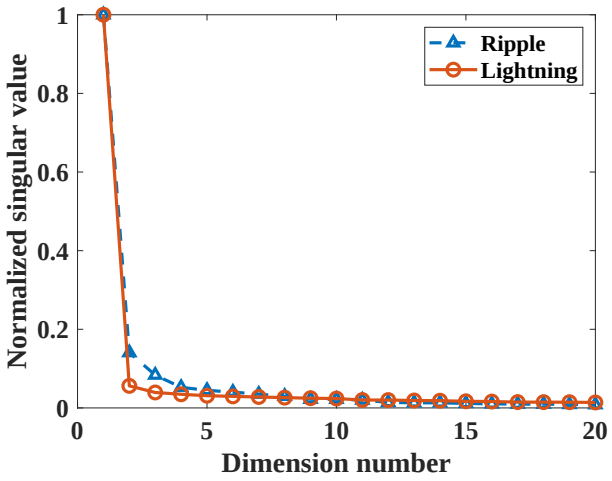
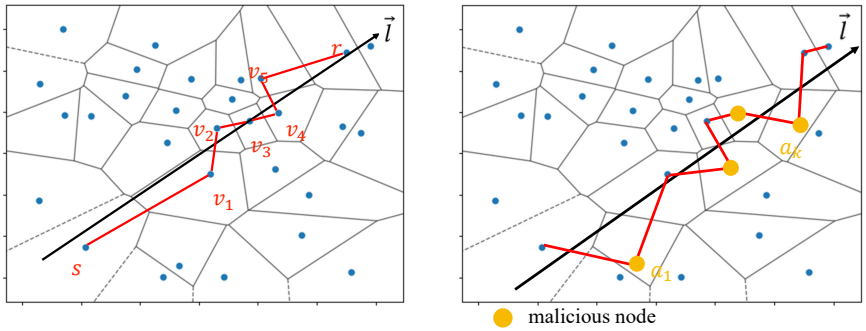


Figure 2.6: SVD analysis of PCNs.



(a) A path by WebFlow-PE

(b) An example of malicious nodes on the path

Figure 2.7: An example of WebFlow-PE on the distributed Voronoi diagram

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 2D and 4D in evaluation.

2.4.2 WebFlow Routing

WebFlow is based on a distributed data structure called Multi-hop Delaunay triangulation (MDT) [73]. 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 [44] 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.2.4(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.2.4(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 [73].

However, in reality, not every DT edge is connected. As shown in Fig. 2.4(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 informa-

tion $< u, N_u, F_u >$ 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.

2.4.2.1 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.

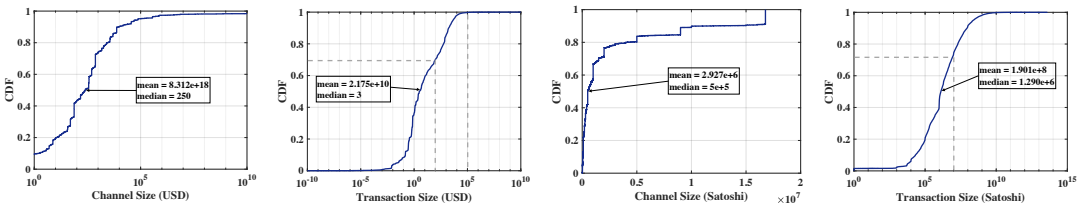
If both situations fail due to channel capacity limitation, u traverses all of its direct neighbors to check if there exists a direct neighbor v closer to the receiver with enough channel capacity to support the payment. If such a direct neighbor does not exist, u selects 5 DT neighbors closer to the 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.

Since large payments could easily use up some specific links [40], if a payment t exceeds a threshold, the sender randomly divides the large payment into several micro-payments $t_1, t_2, ...t_k$, 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.

2.4.2.2 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



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

Figure 2.8: Transaction dataset and channel size distribution used for real-world evaluations.

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.

2.4.2.3 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.

Algorithm 1 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
```

2.4.3 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 2.3. 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 [102], each intermediary still knows the coordinates of the immediately preceding and following nodes. Transactions may still be tracked by malicious intermediaries.

2.5 WebFlow-PE Design

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

2.5.1 WebFlow-PE Overview

Consider the aforementioned problem of the MDT routing in Section 2.4, 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* [21]. As introduced in Sec. 2.4, 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. 2.7(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 1.

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.

2.6 Performance Evaluation

We evaluate the performance of WebFlow comparing 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 ratio, success volume, and overhead under realistic PCN topologies and traces?
- How do network load affect WebFlow's performance?
- How do these results compare to the performance of other approaches?

2.6.1 Methodology

We study WebFlow with two real-world PCNs: Ripple [13] and Lightning [102], as well as synthesis topologies in simulation. For Ripple, we use the data from January 2013 to November 2016, and get the network topology with 1,870 nodes and 17,416 edges in our simulation. For Lightning network, we get the network topology with 2,511 nodes and 36,016 edges on one day of December 2018 [126]. 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 2.8. 70% of the transactions in Ripple is less than 100$, and 70% of the transactions in Lightning is less than 10^7 Satoshi. So we treat payments less than 100$ 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 the 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 [128] and scale-free network model [34]. The link balances are assigned similar to those of Ripple. The payments are also generated by mapping the Ripple transactions to the simulated topologies.

Benchmarks. 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) [84]: 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) [110]: 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 [110] suggests.

Spider [118]: The off-chain routing algorithm which 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 [126]: The off-chain routing algorithm which 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

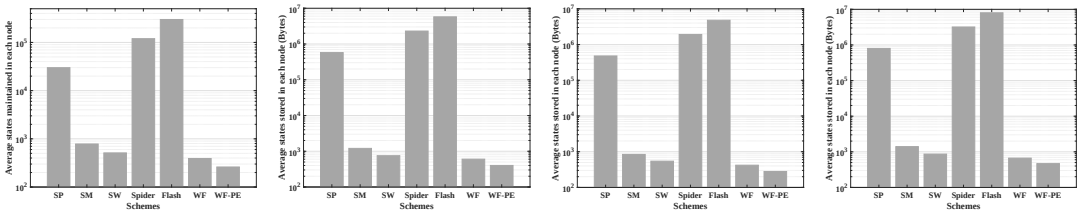
are mice.

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 [110,126], we also use success ratio and success volume as evaluation metrics for resource utilization. Besides, we evaluate the anonymity of the system. The success ratio 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 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.

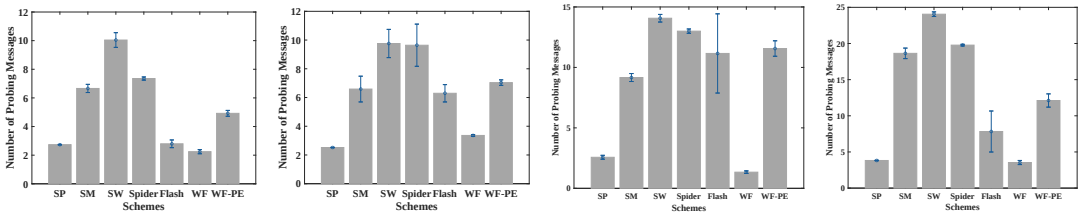
2.6.2 Overall Performance and Overhead

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



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

Figure 2.9: Storage cost compared with baseline methods.

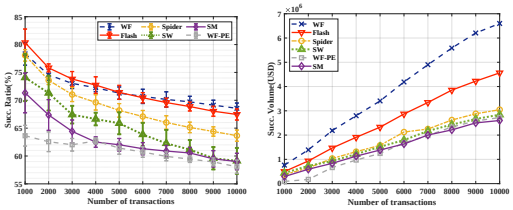


(a) Ripple (b) Lightning (c) Scale-free (d) Waxman

Figure 2.10: Communication cost compared with baseline methods.

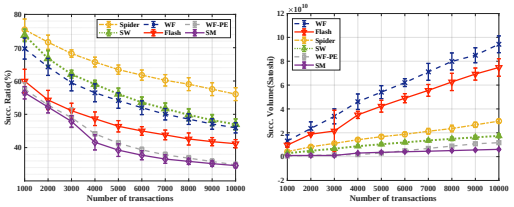
as the paths to all the nodes. Each node only maintains the paths to the landmarks. For SpeedyMurmurs, the coordinate is assigned according to the landmarks, and the length of the coordinate is depending on the depth of the node in the spanning tree. Since there are always several landmarks in the system, 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 2.9 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 cost 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, Speedy-



(a) Succ. ratio (b) Succ. volume

Figure 2.11: Performance with varying transaction numbers in Ripple.



(a) Succ. ratio (b) Succ. volume

Figure 2.12: Performance with varying transaction numbers in Lightning network.

Murmurs, 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 of 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 2.10 shows the comparison results with 1000 transactions. The results here demonstrate that WebFlow indeed efficiently reduce the probing message overhead compared to state of the art in all of the four topologies.

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 of the number of transactions, the success ratio of all schemes decreases in both Ripple and Lightning topologies as shown in Figure 2.11 and 2.12. This is because as more transactions flowing into the network, more links are saturated in one direction,

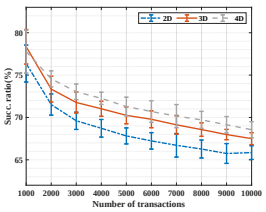


Figure 2.13: Routing succ. ratio in Ripple

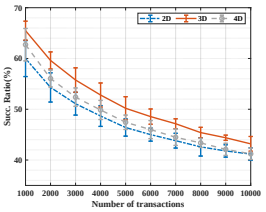


Figure 2.14: Routing succ. ratio in Lightning

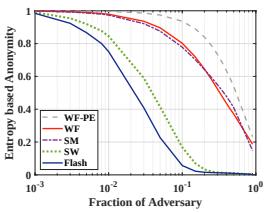


Figure 2.15: Anonymity of Ripple

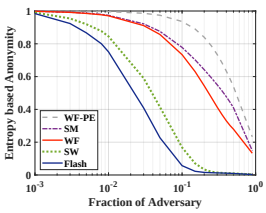


Figure 2.16: Anonymity of Lightning

making them cannot be used for future transactions. In most situations, MDT-based WebFlow shows a higher success ratio and success volume compared to other methods. The only exception is for the success ratio of Lightning, where WebFlow has a lower success ratio 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 ratio. Figure 2.13 and Figure 2.14 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 has lower success rati in lightning. This observation is consistent with the PCA results in Fig. 2.6. 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.

Anonymity. We now demonstrate the comparison of anonymity among WebFlow and Benchmarks. WebFlow-PE shows better anonymity measure as shown in Fig.2.15

Table 2.1: 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

and 2.16. In SilentWhispers and Flash, as long as the attacker is standing on the path, it will know exactly who the sender or recipient is. In MDT-based WebFlow, since the node needs to compute the distance between its neighbors and recipient to find the neighbor closest to the recipient, all the nodes along the path will know the information about the recipient. In SpeedyMurmurs, although it improves privacy by using anonymous return addresses, attackers can still infer that some nodes may be the sender or recipient with higher probability from the knowledge of tree constructions. In WebFlow-PE, nodes can only know the routing direction, instead of the coordinate of the recipient, which reduces privacy leakage. This comparing result does follow our analysis.

2.6.3 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 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 [128]. The

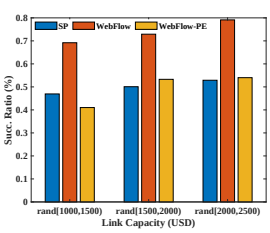


Figure 2.17: Testbed results of the 50-node network.

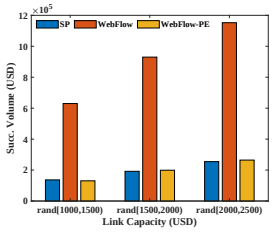


Figure 2.18: Testbed results of the 100-node network.

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 2.1. 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 chose 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 determine 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 sending back to the sender. All the nodes that receive COMMIT_NACK messages will then recover their balances.

Figure 2.17 and 2.18 show the performance with different link capacities. The success ratio 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.

2.7 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.

Chapter 3

APCN

3.1 Introduction

To address the well-known scalability issue of Blockchain, payment channel networks (PCNs) [102] emerge as a leading concept to provide a high-throughput solution for blockchains. The PCN is a promising solution to achieve the scalability of blockchains because most transactions can be achieved in an off-chain manner. However, such "layer-2" blockchain solutions have their own problems: PCNs require a separate deposit for every channel and significant locked-in funds from users [86]. Besides, funds are not equally distributed among all the channels of one user. A situation might happen that a user cannot support a transaction due to insufficient funds in a required channel, but in fact, the node has sufficient unused funds in other channels as in Fig. 3.1. Redistributing funds among channels immediately is not realistic here, because users need to react with blockchain to set up new channels which is time-consuming. Such

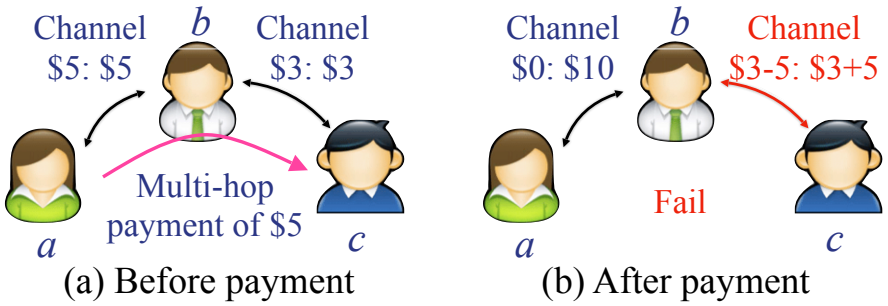


Figure 3.1: A multi-hop payment in a PCN.

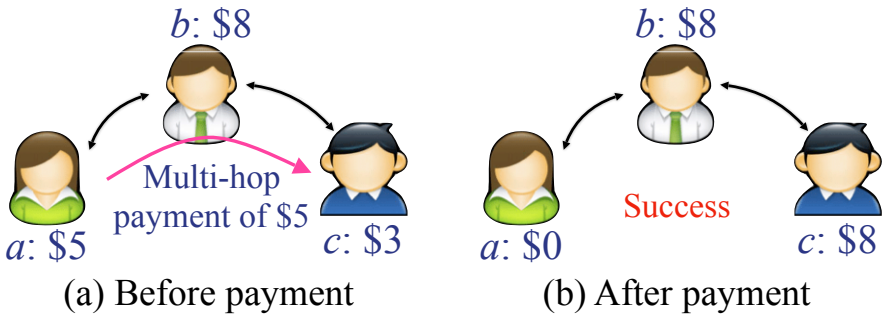


Figure 3.2: A multi-hop payment in an APCN.

inflexibility of fund utilization results in significant resource under-utilization in PCNs. Many recent studies focus on using routing protocols to improve resource utilization in PCNs, such as Spider [118] and Flash [126]. However, our evaluations show that **routing cannot fully solve the problem of imbalanced fund utilization problem across different channels.** The key reason is that per-channel funds also limit routing path selections.

In this work, we introduce Aggregated Payment Channel Network (APCN), a system that enables sharing and freely allocating funding among all payment channels of a single user. In APCN, funds are maintained in a *per-user* basis instead of *per-channel*, which provides higher *flexibility* of fund utilization and hence much higher

payment success rate (from 70% to $>$ 95% in our evaluation). When users perform multi-hop payments, those intermediate nodes only deliver the payments to the next-hop node, instead of adjusting funds in the channels as in the PCNs. So intermediate nodes actually act as relay hops which is more similar to packet-switching networks compared to existing PCNs. Fig. 3.2 shows an example of multi-hop payment in APCN. A multi-hop payment is successful as long as: 1) a path exists between the sender and receiver; 2) the sender has enough funds to pay the receiver. Unlike PCNs, there is no requirement that every channel on the path must have that amount of lock-in funds.

However, there are multiple challenges in designing APCN. **1)** How to prevent users from double-spending. Since funds are not maintained in separate channels, the user cannot determine whether the funds sent to her have been paid to others until she makes the settlement on the main chain. **2)** How to make settlements when shutting down channels or users going offline. In PCNs, payments only change the distribution of the channel's funds, and the total balance of the channel always keeps the same. When closing a channel, two users only need to broadcast a blockchain transaction with the final balance. However, in APCN, the funds are not kept in a single channel, and it is difficult to trace payments in the network. In order to address these two challenges, we design protocols based on the widely available trusted execution environment (TEE) for controlling funds, balances and payments. TEE is a hardware security feature in modern CPUs [81] that ensures the confidentiality and integrity of code and data. **3)** We further assume not every user of APCN has a TEE device. Hence how users can rely on other TEE devices and trust the execution remains another challenge. **4)** We consider the

congestion control problem in APCN: if too many payments go through a certain node, the transaction processing rate on this node should be slower than the transaction arrival rate which causes congestion. Such a node will become the bottleneck of the whole network. To prevent this situation, we design a routing protocol with congestion control in APCN that each channel locally keeps a congestion factor, and nodes would consider the congestion factors of channels to select the next hop.

We conduct both *prototype implementation* and *large-scale simulations* for APCN, based on real-world PCN topologies and transactions. The results show that even the most advanced PCN routing protocols cannot achieve 75% transaction success rate – a transaction is successful if there is a routing path with sufficient funds – while APCN always achieves over 95% transaction success. We show APCN is also cost-efficient.

The rest of this paper is organized as follows. The system overview and model are presented in Section 3.2. We describe an overview in Section 3.3 and the detail design of the APCN and routing protocol in Section 3.4. Section 3.6 presents the evaluation results of APCN. Section 3.7 describes the related work. Section 3.8 concludes this work.

3.2 Overview

3.2.1 Network Model

APCN is a payment channel network in which the funds are maintained in a per-user basis instead of per-channel. In APCN, each user is called a *node*. The bi-directional payment channel shared by two nodes is called a *physical channel* or *direct link*, and these two nodes are called *direct neighbors*. Each node maintains some funds to make transactions with others or help to relay transactions. We model an APCN as a graph $G = (V, E, \Psi)$, where E is the set of links, V is the set of nodes with a weight function w, and $\psi_u \in \Psi$ is the funds of user u in the network. Each node is assigned a congestion rate which can reflect the time it will take on average to process a transaction going through it. This value is periodically updated according to the number of transactions going through it in the last time slot. Furthermore, a path p is a sequence of links $e_1...e_k$ with $e_i = (v_i, v_{i+1})$ for $1 \leq i \leq k-1$. The path of a transaction is accepted only if the amount of this transaction is less than the fund of the sender, ψ_1.

Problem definition. The problem of making successful payments in APCN is described as follows. Consider a transaction t initiated by *sender* s that should be received by the *recipient* r. APCN needs to find a path from s to r, where two consecutive nodes on the path should share a physical link (payment channel) to transfer the payment to the next-hop. The success of the payment implies that s can make a transaction with r by a sequence of transactions involving other intermediate nodes, even if s and r have no trusted channel.

APCN should make use of a *routing protocol* that finds an end-to-end path from the sender to the recipient in the network graph. In fact, APCN is able to apply any existing routing protocol of PCNs and make corresponding adjustments to allow them to work in APCN. In our implementation, we use the virtual coordinates based greedy routing introduced in a recent work WebFlow [142] and extend it for APCN.

3.2.2 Trusted execution environment (TEE)

The requirement for synchronous blockchain access in existing payment networks comes from the fact that their protocols use the blockchain as a root-of-trust: parties executing the payment protocol monitor the blockchain to discover when other parties deviate from the protocol, and react appropriately. In traditional PCNs, users can easily verify transactions by checking their channel states and balances. This mechanism also prevents the *double spending* problem since a single fund cannot be used in two different channels. A single fund cannot be paid to the same receiver twice either, since both the receiver and sender keep a view of channel state. They could detect misbehaving parties when a dispute happens. However, in APCN, the channel is stateless. We should prevent the situation where a malicious node tries to spend a fund twice to two different receivers.

In order to ensure the faithful execution of the payment protocol in APCN, we make use of trusted execution environments (TEEs) [66]. TEEs are encrypted and integrity-protected memory regions, which are isolated by the CPU hardware from the rest of the software stack. Multiple TEE implementations are commercially available,

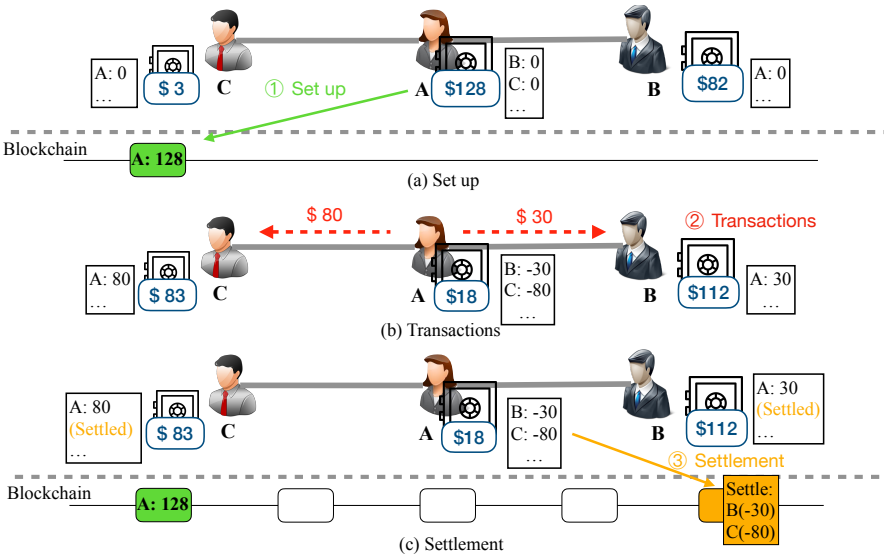


Figure 3.3: APCN overview: APCN nodes operate TEEs to store and manage funds. Users construct payment channels between nodes to exchange funds directly, and execute multi-hop payments along concatenated payment channels.

including Intel SGX [64], ARM TrustZone [59] and AMD SEV [66], with several others currently underway, such as KeyStone Enclave [74], Multizone [112] and OP-TEE [80]. Intel CPUs from the Skylake generation onwards support SGX [61], a set of new instructions that permit applications to create TEEs called SGX enclaves. TEEs ensure faithful execution of software and the owners cannot make changes on either the data or software in TEEs.

APCN constructs a peer-to-peer payment network in which each node comprises: (i) an API for users to interact with the payment network; (ii) an interface through which to read and write blockchain transactions; and (iii) a TEE-protected program called *Ledger* that securely holds and manages users' funds. Ledgers ensure the faithful execution of the payment protocol. They are responsible for managing pay-

ment channels, executing payment transactions, and controlling access to funds. They communicate via secure channels established by two neighboring nodes to update user funds.

Fig. 3.3 shows an example of APCN. For a user A, to join in the APCN at initialization, it needs to construct a set-up message and send it to the blockchain. This message should include a transaction that A makes a deposit of \$128 to the blockchain. After this message being confirmed in the blockchain, A can open channels with other users and make or relay transactions in the APCN. Assume A opens channels with user B and C respectively. When A wants to make a payment of \$30 to B, the TEE of A, denoted as TEE_A, will record this transaction in the local ledger as $B : -\$30$, and update A's remaining funds to \$98. The TEE of B, denoted as TEE_B, will also record this transaction in its local ledger as $A : \$30$, and update B's remaining funds to \$112. The next time when A wants to make another payment of \$80 to user C, TEE_A and TEE_C will update their local ledgers to $C : -\$80$ and $A : \$80$ respectively, and update A and C's remaining funds to \$18 and \$83 as well. When A wants to go offline and make a settlement, it first needs to retrieve the encrypted ledger of the latest version from TEE_A, and then send it to the blockchain. It keeps monitoring the blockchain until its ledger is confirmed. In this process, TEE_A denies all the transactions to or through it. After the ledger showing up in the block, A sends messages to all its neighbors. The neighbors' TEEs who receive the messages will update their local ledgers to mark the transaction records with A as confirmed.

3.2.3 Attacker Model

We assume users and webservers can exchange messages through a traditional secure communication channel such as TLS. Information leakages among them are beyond the scope of our discussion. We assume the attackers can gain complete physical access to a node in which the funds are stored and complete control of its network connections. They may drop, modify and replay messages. An attacker may also delay or prevent the node it controls from accessing the blockchain for an unbounded amount of time. However, they cannot make changes to the TEEs on the controlled nodes. The widely applied TEE implementation SGX is known to be vulnerable to attacks such as controlled-channel attacks, and there have been some countermeasures to them [96]. To prevent information leakage from access patterns, existing oblivious RAM library can be adopted [77]. There are also existing timing and memory-access side-channel resistant libraries for sensitive data [81]. Shih et al. [117] presented a modified LLVM compiler dubbed T-SGX, which is effective against all known controlled-channel attacks. Lee et al. [75] proposed ZigZagger as a defence against their own branch shadowing attack. To defeat enclave specific attacks such as ROP attacks, Seo et al. [113] activated ASLR inside SGX enclaves to make exploitation more difficult. BYOTee [12] put forward a method to build multiple equally secure enclaves by utilizing commodity FPGA devices. Microcode patch could also help, but it can only be changed by the manufacturer of the CPU, which is out of scope of this paper. We apply side-channel resistant libraries and T-SGX in our implementation. We consider the user security of their funds in a fully

distributed PCN. Users may be malicious and attempt to steal funds and deviate from the payment protocol, if it benefits them.

3.2.4 Requirements

Security: The main security requirement of APCN is that it should enable transactions to be executed between users safely and correctly. For safety, we consider two situations. The first is online users making transactions. Since funds are not kept in a single channel, we should ensure that APCN could prevent double spending. If a malicious node tries to spend a fund twice to two different receivers, the receivers should be able to detect it and reject the transaction. The second situation we consider is the settlement. When a user goes offline, all the transactions related to this user should be settled and written to the blockchain. If other users want to go offline later, we need to guarantee that the same transaction will not be written to the blockchain twice.

Performance: The main performance goal of ACPN is a high transaction success rate, which is determined by many factors including available funds, routing protocols, and congestion control to handle concurrent requests.

3.3 Design Overview of APCN

We provide an overview of transaction executions in APCN. Table 3.1 shows the API that APCN provides to users. It supports 1) creating deposits, 2) operating payment channels, and 3) settlement. APCN generates unique identifiers for each deposit and channel, e.g., when a deposit is created (new_deposit), a unique identifier is

Table 3.1: APCN API

APCN APIs	Inputs	Outputs	API Description
Deposits:			
`new_deposit`	t, k	d_i	Create a new fund deposit with ID d_i using a transaction t and the TEE's public key k
`new_pay_channel`	k	c_i	Create a new payment channel with ID c_i with a given TEE identified by k
Payments:			
`Update_ledger`	v, c_i, L	L	Pay an amount v to the other user in a payment channel c_i and update the Ledger
Routing:			
`routing`	v, n_d	n_j	Determine the next hop node n_j of a transaction to destination n_d
Settlement:			
`close_channel`	c_i, L	-	Shut down a payment channel c_i by updating the ledger. Mark the status of c_i as Inactive
`settle_deposit`	v, d_i	t	Refund a deposit d_i by generating and returning a transaction t

returned as a handle to be used in subsequent API calls. TEEs of each user are identified through unique public keys.

TEE service providers. Users generate public/private key pairs for their wallet addresses, which are cryptocurrency addresses owned exclusively by a user's TEE. They are generated securely inside each TEE, and their private keys are stored in TEE memory. The owner of the TEE cannot see the private key. Users can send funds to these addresses in the form of fund deposits. Then deposits can be used in any payment channels of the users. Note that **not all users are equipped with TEEs on their devices**, while some machines with TEEs are willing to provide their TEEs to others. These machines can serve as TEE service providers. Those users without a TEE-enabled node of their own can use a remote TEE service provider to manage their funds.

Users must verify the integrity of TEE before trusting them. APCN uses the remote attestation support of TEEs for verification [62]. A TEE (i) measures the enclave code, (ii) cryptographically signs the measurement and the user's public key, and (iii) provides the signed measurement and public key to the remote user [81]. The remote user then verifies the attestation, i.e., the remote user ensures that the attestation is correctly signed by the Trusted hardware and that the measurement corresponds to

a known TEE implementation. Users can thus verify that a specific service provider, identified by its public key, is running the protocol correctly in the TEE hardware. And remote TEE providers have the same abilities as a local TEE. To deal with the situation that the machine with remote TEE going offline and avoid having to trust a single remote TEE service provider, APCN constructs committees with multiple remote service providers.

Service provider Committee. Committees are groups of TEE service providers that jointly manage fund deposits, ledgers and transactions. They are used to prevent single point failure when a user does not have a local TEE and has to rely on a TEE service provider to manage their funds. For each deposit owned by a service provider committee, a minimum number of members are required to sign transactions before that deposit can be spent, thus tolerating a fixed percentage of TEE failures to some degree. For this, APCN used multi-signature support of the blockchain: each fund deposit is paid to a m-out-of-n wallet address, where m TEE signatures are required to spend the deposit. The n committee members are responsible to manage the user deposit [81].

3.4 Payment Protocol

This section describes the design of APCN protocols.

3.4.1 Deposits allocation

In order to create a deposit for a user, a transaction indicating making deposits related to the user needs to be recorded on the blockchain. To construct a new deposit d, users invoke `new_deposit`, and present a deposit transaction `t` and the public key of the user's TEE. The TEE then verifies that `t` sends funds to the correct address using its public key `k`. The TEE then constructs a new deposit d, forwards `t` to the blockchain, and returns d's unique identifier signed by the TEE to the user.

Although the deposit is maintained by each user, we still need payment channels for transactions among users. Since the channels in APCN are stateless without funds in them, it is not necessary to associate a determined number of deposit with a certain channel. To create payment channels between users without a blockchain interaction, participants call `new_pay_channel` and provide the public key of the TEE with which to create the channel. The two TEEs then establish a secure communication channel using authenticated Diffie-Hellman for key provisioning and remote attestation. Using the secure channel, the TEEs assign a unique channel identifier to the channel c and return the channel identifier.

3.4.2 Using payment channels

To execute a payment t along a channel, the sender u calls `update_channel`, which specifies the amount ω to send and the channel identifier c_i. The sender's TEE first ensures that the sender has sufficient funds, $d_u > \omega$, before decrementing the sender's balance and incrementing the recipient v's balance locally. It then forwards the

payment to the recipient's TEE to update balances. If the payment is not received by the recipient in a pre-determined time slot, e.g., due to a network failure, the sender's TEE rolls back the payment to prevent balance inconsistencies. If the payment is received by the recipient successfully, the sender's TEE needs to update the remaining deposit to be $d_u - \omega$, and the recipient's TEE needs to update the deposit to $d_v + \omega$. They also need to update the ledgers of the users respectively, which is $v : -\omega$ in the sender's ledger, and $u : +\omega$ in the recipient's ledger.

3.4.3 Congestion control

In PCNs like the Lightning and Raiden networks, most users by default pick the shortest path from the sender to the destination. However, it leads to the congestion problem [118]. Consider an example PCN shown in Fig 3.4. Suppose many users on the left side of a (in Cluster A) try to make transactions with users on the right side of b (in Cluster B) at the same time. Based on many routing protocols, when transaction requests from cluster A reach node a, a always forward those transactions to node b which has shorter paths to the receivers in cluster B. This leads to congestion on channel $a - b$, while channels $a - u$ and $b - u$ are under-utilized. And thus, all the transactions between clusters A and B would suffer from extra processing latency of channel $a - b$.

To address this problem, we introduce a congestion factor l_c for each channel, which shows the current processing latency for an incoming transaction in the channel. However, it is impossible to minimize the processing latency as well as maximize the success volume of the whole system as a linear programming problem, because we cannot

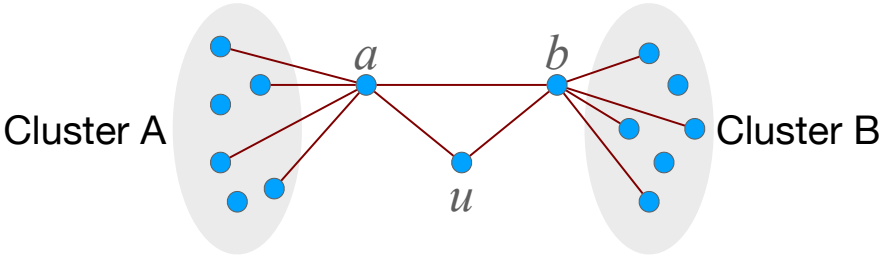


Figure 3.4: Example illustrating the importance of congestion control in APCN.

probe and compare all the possible paths for every transaction in advance. Instead, we apply a heuristic mechanism to optimize the end-to-end latency - Latency Awareness (LA) forwarding, which avoids congested links.

In LA forwarding, a node u chooses a neighbor x as the next hop to receiver r such that it minimizes the heuristic function $h(u) = l(u,x) + \widetilde{l}(x,r)$. $l(u,x)$ is computed as how many transactions are currently using the channel $u-x$, and $\widetilde{l}(x,r)$ denotes the estimated routing latency from x to r from locally computing the distance between the virtual positions of x and r. The first question is how to assign the congestion factor l_c for each channel. Assume the processing latency of a transaction at an idle channel c is Δ, and the channel can process one transaction at a time. If multiple transactions want to use the same channel, they will be put in a queue and l_c is adjusted according to the number of transactions in the queue. For example, the congestion factor of an idle channel c is $l_c = \Delta$. If there are 2 unfinished transactions in the channel c, the congestion factor becomes $l_c = 3\Delta$. The second question is how to estimate the remaining routing latency $\widetilde{l}$ from a neighbor node x to the receiver r. Note that we assign each node a virtual coordinate that reflects the network topology features. The node pair with small hopcounts in the network also shows a short distance in

the Euclidean space. So we use the distance d_{xr} between the virtual positions of x and r as the estimated hopcounts between them. We estimate $\widetilde{l}(x,r)$ such that it is proportional to the estimated hopcounts between x and r. For simplicity, we assume all the channels between x and r are idle. So the heuristic function at node u is computed as: $h(u) = (n(u,x)+1)\Delta + d(x,r)\Delta$, where $n(u,x)$ is the ongoing transactions in the channel ux.

However, simply assuming that all the channels between x and r are idle is not accurate since node u does not have any information on these channels. And selecting the next hop x according to the heuristic function $h(u)$ instead of choosing the next hop that is closest to the receiver r may lead to a larger routing stretch, and thus may introduce extra routing latency. There exists a trade-off between WebFlow which has lower routing stretch, and LA forwarding which has lower estimated routing latency. So we combine WebFlow and LA forwarding together. For each transaction arrived at node u, it has the probability p to apply WebFlow to be forwarded to the neighbor closest to the receiver. Otherwise, it runs the LA forwarding protocol. We will further evaluate and find the optimal p value in evaluation.

3.4.4 Deposits settlement

In PCNs, if a user wants to shut down a channel, he needs to have a transaction claiming the final state of the channel recorded on the blockchain. However, in APCN, shutting down a channel would not require any operation on the blockchain as we mentioned in Sec 3.4.2. Only if a user wants to go offline, does he need to settle his

deposits and have his final deposits on the blockchain. The channel record in a ledger has four statuses: **Pending, Complete, Inactive, and Settled**. `Pending` is the status that there exist one or more ongoing transactions related to this channel. `Complete` is that all the transactions going through the channel is complete and confirmed by the sender and recipient. `Inactive` is the status that the user has shut down the channel and this channel does not exist anymore. `Settled` is the status that the neighbor that the user shares the channel with is offline and has settled all his channels and deposits.

To shut down a channel of the user u, it invokes `close_channel` and includes the channel id and the user id of its neighbor v whom it shares this channel c with as inputs. The TEE of u first checks the status of channel c in its ledger. If the status is `Inactive`, it means that the channel c has been closed before and does not exist currently. So `close_channel` will return 'FALSE'. If the status is `Settled`, it indicates that v is offline and has settled all the related channels. So the function will fail to shut down the channel and return 'FALSE'. If the status is `Pending`, it means that there exist one or more ongoing transactions related to this channel, including u sending payments via the channel c, other users sending payments to u via the channel c, and c served as intermediate hop of passing by transactions. In this case, the TEE of u will hold the `close_channel` request until the status of c becomes `Complete`. It is to prevent the situation that a transaction has probed and determined the path, but some channels of this path break down before the transaction completes. In the whole process, the TEE of u will reject any other transactions via the channel c. If the status of channel c becomes `Complete`, u's TEE can directly shut down the channel by changing the status

to `Inactive` and inform v's TEE to change the status of channel c to `Inactive` in v's ledger as well. To prevent the case that a malicious u tries to close the channel using a stale state to benefit itself, when it invokes `close_channel`, v will have a bounded reaction time to invalidate the action by providing the latest state with the timestamp. If v approves or fails to respond within the time slot, u will continue closing the channel.

Consider user u wants to go offline and settle its deposit on the Blockchain. The first thing it needs to do is to settle all its channels by invoking `close_channel`. After the status of all the channel records in its ledger becomes `Complete`, the next step of u is to call `settle_deposit` and settle its deposits on chain. To do this, u need to obtain the latest ledger signed by its TEE from the TEE, and directly send its signed ledger to the Blockchain. u needs to keep monitoring the Blockchain until its ledger is verified, packed into a block, and added to the Blockchain. Then, u constructs a proof that its ledger has been added to Blockchain and sends the proof to all its neighbors. If some neighbors do not agree on the channel states, they could provide the correct signed ledger to Blockchain to dispute. If the proof is correct, the neighbor nodes will mark the channel $u - v$ as `Settled` in their own ledgers.

3.4.5 Transaction data format

Transactions. All transactions among users are conducted via channels by TEE service providers. Each transaction τ includes the address of the transaction recipient τ_{to}, the transaction amount τ_a, the address of the last hop user τ_f, and a monotonically increasing transaction index τ_i. We note that τ_f here is not necessarily

Algorithm 2 APCN payment protocol executed by each node and TEE service provider.

```
1: def new_deposit_withTEE(t, k):
2:   verify_tx(t, k)
3:   d ← create_new_deposit(t)
4:   deposits[d_i] ← d
5:   write_to_blockchain(t)
6:   create_Ledger(d_i)
7:   return d_i
8: def new_deposit_withoutTEE(t,
     k_1 ...k_m):
9:   verify_tx(t, k_1...k_m)
10:  d ← create_new_deposit(t)
11:  deposits[d_i] ← d
12:  write_to_blockchain(t)
13:  create_Ledger(d_i)
14:  return d_i
15: def approve_channel(k):
16:  apprv ← ask_approve_remote(k)
17:  return apprv
18: def new_pay_channel(k):
19:  c ← create_channel_with(k)
20:  channels[c_i] ← c
21:  add_Ledger(c_i)
22:  add_Ledger(c_N)
23:  return c_i
24: def pay_channel(v, c_i):
25:  c ← channels[c_i]
26:  assert(c.my_bal ≥ v)
27:  Update_Ledger(-v, c_i)
28:  Update_Ledger(v, c_N)
29: def create_Ledger(d_i):
30:  d ← deposits[d_i]
31:  a ← d
32:  return L
33: def add_Ledger(c_i, L):
34:  c ← channels[c_i]
35:  c.a ← 0
36:  s ← s_c
37:  return L
38: def Update_Ledger(v, c_i, L):
39:  c ← channels[c_i]
40:  c.my_bal ← c.my_bal + v
41:  c_a ← c_a + v
42:  s ← s_p
43:  return L
44: def close_channel(c_i, L):
45:  c ← channels[c_i]
46:  Close_Ledger(c_i,L)
47:  Close_Ledger(c_N,L)
48: def Close_Ledger(c_i, L):
49:  // Collect all entries of channel
     c
50:  c ← channels[c_i]
51:  if s == s.c then
52:      s ← s.i
53:      return TRUE
54:  else
55:      wait for time Δ
56:      if s == s.c then
57:          s ← s.i
58:          return TRUE
59:  return FALSE
60: def settle_deposit(d_i):
61:  for all channels c in U's ledger
     do
62:      close_channel(c_i)
63:  t ← construct_tx(d_i, k)
64:  return t
```

the sender. For multi-hop transactions, τ_f records the last hop where the transaction comes from. For each intermediate user received a transaction, it needs to replace the τ_f field to be the address of it, and relay the transaction to the next hop.

Ledger state. The ledger in the TEE maintains state that contains the remaining deposit amount of the user, and several entries as shown in Table 3.2. Each entry denotes a channel of the user u, and consist of the following items: the channel c built by u and its neighbor, the neighbor c_N the user u shares the channel c with, the overall amount u sent to the neighbor c_N via channel c, and the state s of this channel c as introduced in Sec 3.4. The amount of the channel can be negative, which is the amount user u owes c_N.

Table 3.2: Ledger state

Field	Symbol	Description
`Channel`	c	The channel id of this entry
`Neighbor`	c_N	Neighbor's address the user build channel c with
`Amount`	c_a	The overall transaction amount of the user
`State`	s	the state of transactions going on in the channel
	$\hookrightarrow s_p$	Pending, ongoing transaction in the channel
	$\hookrightarrow s_c$	Complete, all transactions in the channel complete
	$\hookrightarrow s_i$	Inactive, user has closed the channel
	$\hookrightarrow s_s$	Settled, user's neighbor has settled the channel
`Version`	ω	The version number of the latest transaction

3.4.6 Users with and without TEE

Here we describe two categories of users separately, the user with local TEE, and the user without local TEE. Algorithm 2 shows the protocol executed by each node and TEE service provider. To construct a new deposit d, users with local TEE invoke `new_deposit_withTEE` (Alg. 2, line 1) and present a deposit transaction t and the TEE public key that t sends funds to. TEE verifies that t sends funds to the correct address using its public key k, and then constructs a new deposit d, forwards t to the blockchain, and returns d's unique identifier to the requester (line 7), signed by the corresponding TEE. For users without local TEE, they have to use more than one remote TEE service provider to prevent malicious attackers. To construct a new deposit d, users without local TEE invoke `new_deposit_withoutTEE` (line 8), and present a deposit transaction t and the list of TEE service providers' public keys forming the committee that t sends funds to. The service providers then verify that t sends funds to a k-out-of-m multi-signature address using the committee members' public keys, $k_1...k_m$, and notify the committee of the new t. The user then constructs a new deposit d, forwards t to the

blockchain, and returns d's unique identifier to the requester (line 14), signed by all committee members.

Payment channels do not hold any funds, and can be set up or close at any time. Creating a payment channel c is to add an entry in the ledger of user u and v. Before the channel c can be set up, it must be approved by the the remote party (e.g., v if u requests channel creation approval) using `approve_channel` (line 15). Approval contacts the remote user via its TEE and queries if the user is online to build a channel c.

After approval, to create payment channels between users without blockchain interaction, participants call `new_pay_channel` and provide the public key of the TEE with which to create the channel (line 18). The TEEs of two users then establish a secure communication channel using authenticated Diffie-Hellman for key provisioning and remote attestation. Using the secure channel, the TEEs assign a unique channel identifier c_i to the channel c, initialize both participant's balances to 0, and return the channel identifier (line 24). Then the two users u and v need to create the corresponding entry of the channel in their ledgers using `add_ledger` (line 33). When u creates the channel c, its TEE initializes the amount of entry c in the ledger to be 0, and the channel state to be s_c. Only after the ledger created successfully, can the channel c be used by user u and v for future transactions. If one of them wants to close this channel, she needs to call `close_channel` (line 44) to close the corresponding entry of the channel in both u and v's ledgers. At any time, users may settle the deposit using `settle_deposit` (line 60) by calling `close_channel` for all channels.

3.4.7 TEE operations

In this section, we describe three functions associated with ledger: Ledger creation, Ledger update and Ledger close. The construction consists of the instructions for two users, Alice and Bob, and their ledgers on their TEEs.

Ledger creation: We start with describing a procedure in which Alice and Bob register in the APCN system with the initial balance, a_A and a_B. As mentioned in Sec 3.4.6, after their TEEs verifying the correctness of their deposit transactions t_A and t_B, respectively, their TEEs need to construct new deposits d_A and d_B, forward their deposit transactions to the blockchain, and initialize a ledger with the deposit d_A and d_B, with the amount being a_A and a_B. The current version of the ledgers is empty ones with no entry.

When Alice and Bob agree to open a channel c in APCN, their TEEs negotiate and assign a unique channel identifier c_i to the channel c. Then TEEs need to create the corresponding entry of the channel in their ledgers, whose format should follow Table 3.2. For the new ledger entry in Alice's TEE, the Deposit field continues to be a_A, since no transaction happens at this time. The Channel field is c_i as the return value of the function `new_pay_channel` in Alg. 2. The Neighbor field c_N is set to be Bob's address. The Amount field is initialized as 0, since no transaction happens and the overall transaction amount Alice sends to Bob is 0. The State field is s_c, which means the channel is active and there is no pending transaction in the channel, so the channel is ready to serve future transactions.

Ledger update: When Alice and Bob want to make a new transaction when there is an ongoing transaction in channel c, we use a standard technique (see, e.g, Sec. 3.3 in [102]) for updating the entry for a payment channel in the ledger that is based on counters called "version numbers" $\omega \in N$. Note that the transaction here includes the direct transaction between Alice and Bob, and the multi-path transaction going through Alice and Bob. We do not distinguish between these two situations. Initially, ω is set to 0, and it is incremented after each transaction via channel c. Suppose Alice initiates the first transaction τ of amount τ_a in channel c. If Bob agrees on this transaction, TEE_A and TEE_B both need to update the corresponding entry in their ledgers. On Alice's side, there is only one entry of channel c in its ledger, and the current status of the entry is s_c with version number 0. So Alice will update its ledger by updating this entry. The Amount field is set to be $c_a - \tau_a$. The State field is changed to s_p until the transaction is complete. Also, the version number ω is incremented by 1. In Bob's ledger, its TEE updates the Amount field to be $c_a + \tau_a$, the State field to be s_p, and the version number to be 1 in the entry for channel c.

Ledger close: If one of the parties, say Alice, wants to close the channel c, she first needs to negotiate with Bob. After approval by Bob, both of them needs to close the entry of channel c in their ledgers. Again, their TEEs need to check their ledgers, collect all entries of channel c, and merge all those with status s_c. After this, if there is only one entry of channel c and its state is s_c, TEEs can directly close the channel by setting the State field of the entry to be Inactive s_i. If there exists some entries of channel c with status s_p, TEEs wait time Δ for those transactions to complete. After

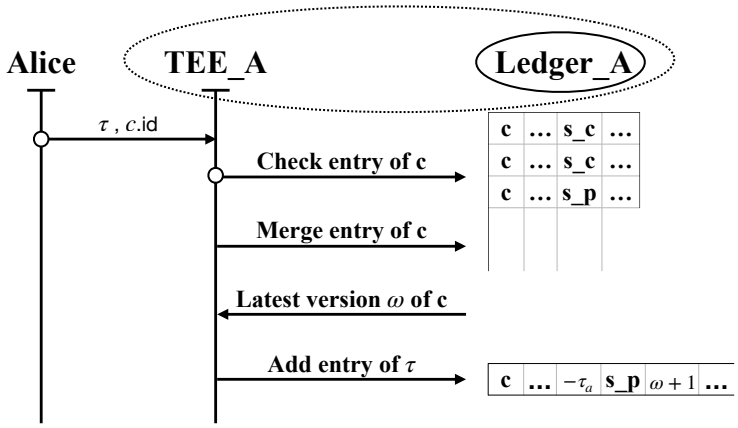


Figure 3.5: Illustration of ledger update protocol.

the waiting time Δ, TEEs merge those entries with status s_c and update the State field to be Inactive s_i. Those entries whose status are still s_p will be abandoned.

3.4.8 TEE committees

We provide TEE committees to prevent malicious TEE service providers for users without TEEs. For a new TEE service provider who wants to join the system, it has to perform remote attestation with a group of TEE committee to verify that it has the correct code and works correctly. It also has to pay certain amount of participation fee to be included in this committee. Every time when the committee performs a transaction correctly, all the members will receive incentive from the user.

When creating channels or sending a payment, a user should get approval from the committee and update its ledger. In order to achieve agreement and consistency of ledger state among all committee members, APCN uses Committee chains introduced in TeeChain [81]. The chain replication offers strong consistency without requiring all

committee members to communicate directly. The committee members form a chain, with the primary at the head, and the last backup at the tail. The user first sends the update request to the primary in the committee. The primary will check if the user has sufficient funds and propagates the update down the chain. Each committee member does the same check, forwards the update to its backup, and waits for an acknowledgment before updating the ledger. When the primary receives an acknowledgment, the entire chain has updated. If any committee member fails or refuses to update to the latest agreed upon ledgers, the replication chain is broken, freezing all nodes at the current ledger state. And this member will lose all its participation fees and incentive in the committee.

3.5 Protocol Security Analysis

APCN protects the funds of all users in the PCN: despite what others may do, funds cannot be stolen or double spent. At any time during the payment protocol execution, each user should be able to perform a finite set of actions that eventually results in them receiving their perceived balance on the underlying blockchain.

We now prove that APCN achieves funds security using the Universal Composability (UC) framework [25] similar to prior work [39, 81]. The UC framework includes parties executing the protocol in the real world, ideal functionalities performed by idealized third parties, and a set of adversaries $\mathcal{A}$. A protocol is said to be UC secure if the real-world execution of the protocol cannot be distinguished from the idealized protocol

execution by the environment.

We model committees as a single TEE executing the protocol. Under UC, we consider a real world, in which users run the APCN protocol, π_{APCN}, as described in Sec 3.3, and an ideal world, in which users interact with an ideal functionality, F_{APCN}, implemented by a trusted third party. Attackers behavior is introduced in the ideal world by a simulator $\mathcal{S}$ with appropriate attacker abilities as described in Sec 3.2.3. To prove that APCN achieves fund security, we show that (i) the real and ideal worlds are indistinguishable to an external observer ε. This implies that any attack violating fund security in the real world is also possible in the ideal one; and (ii) F_{APCN} achieves fund security in the ideal world. This proves that π_{APCN} also achieves fund security. We'll show that the simulator $\mathcal{S}$ in the ideal-world translates every adversary $\mathcal{A}$ in the real-world into a simulated attacker, which is indistinguishable to the environment.

We prove indistinguishability between the real and ideal worlds through a series of five *hybrid steps*, starting at the real world H_0, and ending in the ideal world H_5. In each step, a key element is changed and indistinguishability is proven. As commonly done [16], in H_0, the desired behavior of TEEs and the blockchain are replaced by two ideal functionalities, F_{TEE} and F_B respectively. F_{TEE} is an ideal functionality that models a TEE. It abstracts an enclave as a third party trusted for execution, confidentiality and authenticity, with respect to any user that is part of the system. F_B is an ideal functionality that represents the blockchain. H_1 behaves the same as H_0 except that $\mathcal{S}$ simulates F_{TEE}. When the adversary $\mathcal{A}$ wants to communicate with its F_{TEE}, $\mathcal{S}$ faithfully emulates F_{TEE}'s behavior and records $\mathcal{A}$'s messages. As $\mathcal{S}$ simulates the

real-world protocol perfectly, the environment ε cannot distinguish between H_0 and H_1. In H_2, $\mathcal{S}$ simulates F_B. When the adversary $\mathcal{A}$ wants to interact with the blockchain, $\mathcal{S}$ emulates F_B's behavior for $\mathcal{A}$, and no environment can distinguish between H_1 and H_2. H_3 behaves the same as H_2 except that if $\mathcal{A}$ invoked its F_{TEE} with an incorrect call, $\mathcal{S}$ aborts and drops incorrectly signed messages to F_{TEE}. Otherwise, $\mathcal{S}$ delivers the message to the honest party in the protocol. H_2 and H_3 are indistinguishable, or else ε and $\mathcal{A}$ can be leveraged to construct an adversary that succeeds in a signature forgery. In H_4, the only difference is that incorrectly signed messages to F_B are dropped by $\mathcal{S}$. H_4 is indistinguishable from H_3 for the same reasons as the last step. H_5 is the ideal world execution, that calls of $\mathcal{S}$ to F_{APCN} are mapped from the calls in the simulated real-world. In H_4, $\mathcal{S}$ can faithfully interact with F_{APCN}, while faithfully emulating $\mathcal{A}$'s view of the real-world. $\mathcal{S}$ can then output to ε exactly $\mathcal{A}$'s output in the real-world. So it is equivalence between π_{APCN} and F_{TEE} to ε.

Since for any environment the ideal-world and the real-world executions are indistinguishable, funds security that holds in the ideal-world will also hold in the real-world. We now discuss why the ideal functionality F_{APCN} satisfies the security requirements from Sec. 3.2.4.

Correctness on channel update. For users sharing a channel with their own TEEs, the correct channel activities are achieved by the ideal functionality notifying the users of whether the channel has successfully been created or updated. For users without TEEs, chain replication in the TEE committee offers strong consistency among all TEEs, which will finally achieve consensus on channel state and notify users.

Guaranteed channel closing with latest state. A channel $u - v$ can be closed by either u or v with latest state. If u sends a channel closing request to the ideal functionality F_{APCN}, it will inform v with a message. If it does not receive any dispute or response from v within time Δ, it will close the channel after the channel finishing all the ongoing transactions or reaching time bound. If v provides a dispute with the correct signed ledger and latest timestamp, F_{APCN} will accept this channel state to close the channel, and also for future settlement.

Guaranteed no double spending. Consider a user u, it calls the ideal functionality F_{APCN} to make a transaction to v. F_{APCN} always guarantees that u has enough funds to pay v, and updates funds after each transaction. It makes sure that u cannot use the same amount of money to pay others twice.

3.6 Performance Evaluation

We present the evaluation results based on prototype implementation and simulations.

3.6.1 Methodology

We implement the APCN prototype using Intel SGX SDK in C++. The prototype is mainly used to evaluate the real latency to generate ledgers, links, and transactions. Note that multiple TEE implementations are commercially available, including ARM TrustZone and AMD SEV. They can also be applied.

The simulations use two real PCN topologies: Ripple [13] and Lightning [102],

as well as synthesis topologies. 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, we get the network topology with 3,519 nodes and 47,311 edges on one day in January 2022. The node balance in APCN is assigned as the sum of the channel balances of a node. We build two sets of synthetic PCN topologies based on the Waxman model [128] and the scale-free network model [34]. The node balances are assigned similar to those of Ripple. The payments are also generated by mapping the Ripple transactions to the synthetic topologies.

In order to defend side-channel attacks, we use timing and memory-access side-channel resistant libraries, AES-NI based AES-GCM [60, 63]. To further enhance the security of APCN, we apply T-SGX [117], a countermeasure for controlled-channel attacks.

Comparisons. To evaluate the performance of APCN, we compare APCN with WebFlow [142], SpeedyMurmurs (SM) [110], Spider [118], Perun [38], and shortest paths (SP).

Metrics. We use average processing latency and the number of hopcounts to evaluate the congestion control mechanism in APCN. The processing latency of payment is calculated as the sum of per-hop delay along the path which is related to the channel condition. Similar to prior work [110, 126], we also use success rate as evaluation metric for resource utilization, defined as the percentage of successful payments whose demands are met overall generated payments. We report the average results over 10 runs, each of which includes hundreds of communication pairs.

Table 3.3: Channel performance

Operation and Latency(ms)	APCN	APCN w/ T-SGX
Single Local TEE:		
`new_payment_channel`	2,310	6,183
`close_channel`	2,205	5,830
`makepayments`	105	291
Remote TEE:		
`new_payment_channel`	4,317	25,294
`close_channel`	2,984	12,523
`makepayments`	427	1,015

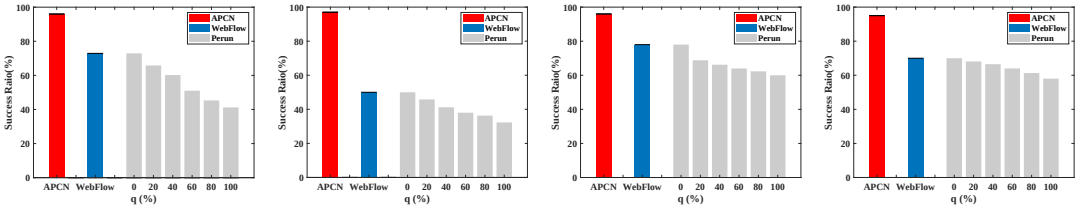
3.6.2 Evaluation Results

Performance of payment channels. We conduct a testbed evaluation with the prototype. In the experiments, we construct a payment channel between two users with local TEEs. So the users only use a single local TEE to manage their ledgers and transactions instead of the TEE service providers committee. We execute several transactions between them. Table 4.1 shows the performance of different actions of APCN, and the latency when applying T-SGX to improve system security. Each channel creation takes 2.3 secs on average. It is much faster than channel creation in Lightning Network, which is approximately 60 mins, as a transaction must be placed onto the blockchain and confirmation takes 6 Bitcoin blocks. Channel creation in APCN only requires the corresponding TEE to perform remote attestation and add an entry in its ledger, without the participants of the blockchain. Even though remote attestation requires participation of the Intel attestation service, it will not become the bottleneck when the system scales up. The reason is that each user only has limited number of channels with its neighbors, and channel creation is not a frequent action. As long as the channel is there, users can perform unlimited number of transactions via the channel.

To close a channel, TEE has to wait until the channel status in the ledger becomes 'Complete'. The waiting time can vary a lot, so we only evaluate the time to close a channel whose status is Complete. In APCN, closing a channel only requires status change in the ledger and takes 2.2 secs on average, which is much less than the time to close a channel in Lightning Network which requires a transaction in blockchain. For the payments processing latency, we only consider the time of an idle channel processing one payment. It is 105 ms on average. We use this time as Δ in our congestion control mechanism in evaluation.

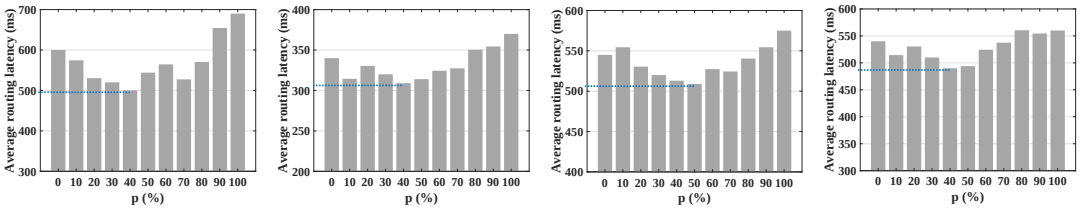
We then consider the case of non-SGX users. we construct a payment channel between two users, one is equipped with SGX, one is not and uses a TEE service provider committee at size of 3. Creation of such a payment channel takes 4.3 secs, as the non-SGX user must verify the integrity of TEEs of the committee. Closing channel and processing payment also take more times, 2.9 secs and 427 ms respectively, since each TEE service provider in the committee needs to verify and sign each update of the user's ledger. When applying T-SGX to APCN, the processing latency increased within 5 times in all the cases, which is tolerable for better security.

Comparison with other PCNs. We use simulations to compare the performance of APCN, WebFlow, and Perun – a virtual payment channel system. For virtual payment channels, we analyze the historical transaction dataset in different network topologies. The virtual channels are built according to the transaction frequency of user pairs. We tends to build virtual channel for user pairs making transactions with higher frequency. We set the proportion of virtual channels to be q, and vary the q



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

Figure 3.6: The success ratio comparison of APCN, PCN and virtual payment channel network with varying proportion of virtual channels.



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

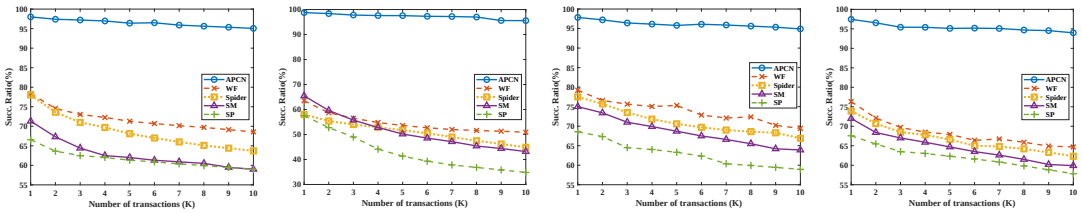
Figure 3.7: The average routing latency with varying p values.

value from 0% to 100% to test the performance. Here, we use 5,000 transactions in each run. Figure 3.6 shows the success rates of transactions in APCN, WebFlow, and Perun with varying q value. When q is 0, Perun has no virtual channel, and it becomes the same scheme as WebFlow. All users need to execute the routing protocol to probe the payment channels to send or relay transactions. So it has the same performance as WebFlow. With more payment pairs setting up virtual channel, the overall success rate decreases a lot. The reason is that, with more virtual channels in the networks, more funds are locked in the virtual channels, and those funds cannot be used in other channels. Although virtual payment channels provide a very fast way to stream payments, it actually affect the overall success rate.

Efficiency of congestion control mechanism. We first consider the influ-

ence of parameters in our congestion control mechanism. With congestion control, the intermediate node would send the payment to the direct neighbor closest to the receiver with a probability p in APCN. To find the optimal p for our system, we vary the p value from 0% to 100%, and see how the choice impacts the performance, i.e. average processing latency. Here, we use 5,000 transactions in each run. Figure 3.7 shows the average processing latency with varying p values. It is understandable that both settings: $p = 0\%$ and 100% result in relatively high average processing delay. Because when p equals to 0%, it becomes the same routing protocol as WebFlow without congestion control. Even if the algorithm always chooses the next hop that is closest to the receiver and more likely to have lower routing stretch, the next hop itself may introduce large processing delay, and thus lead to higher overall processing latency along the path. On the contrary, when p equals to 100%, our heuristic routing algorithm at intermediate nodes estimates the remaining processing latency proportional to the distance from the node to the recipient. However, this estimation is not accurate reflecting the processing latency, since hop-delay is not a stable metric and changing over time. Observed from the evaluation result, when p equals to 40%, the congestion control mechanism could achieve a better performance. So we set p value to 40% in the following experiments.

Performance with different networks. We evaluate APCN with four PCN topologies and a varied number of transactions. As shown in Fig. 3.8, by increasing of the number of transactions, the success rate of all schemes except APCN decreases significantly in all topologies. The reason is that, for other schemes under traditional payment channel networks, as more transactions flowing into the network, more channels



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

Figure 3.8: The success ratio with varying transaction numbers under different network topologies.

are saturated in one direction, making them cannot be used for future transactions. However, in APCN, the success ratio almost keeps over 95% and does not have obvious changes, while success ratio of other schemes are always below 80%.

3.7 Related Work

PCNs provide a high-throughput solution for blockchains [50]. Lightning Network [102] uses max-flow routing algorithms to find paths. Flash [126] also uses modified max-flow routing algorithms but treats elephant and mice payments differently. SilentWhispers [84], SpeedyMurmurs [110], and have been proposed to improve routing scalability.

In order to improve the fund utilization and avoid channel imbalance, Spider [118] develops a multi-path congestion control algorithm. It is a centralized offline routing algorithm and still has a high probing overhead. REVIVE [68] enables users to securely rebalance their channels, according to the preferences of the channel owners. Sprites [90] supports partial withdrawals and deposits, during which the channel can continue to operate without interruption, but requires smart contracts. Teechain [81]

supports dynamic deposits with treasuries by TEEs, in order to prevent parties from stealing the fund. Different from them, APCN enables shared deposits among all payment channels of each user and allows funds to be used with high flexibility.

3.8 Conclusion

We present APCN, a novel design for PCNs that enables shared funds among all the payment channels of a node. This design provides high fund-allocation flexibility and hence significantly increases transaction success rates. To prevent users from misbehavior, we use TEEs to control funds, balances, and payments. We also design a routing protocol in APCN that takes congestion control into account. We build a prototype of APCN with Intel SGX and evaluate the performance with both prototype experiments and simulations with real PCN data. Results show that APCN achieves evidently higher success rates of multi-hop payments with lower average hops and latency, compared to existing PCNs.

Chapter 4

XHub: A Cross-chain Payment Channel Network

4.1 Introduction

Blockchain is a promising solution for decentralized digital ledgers. Since Bitcoin was invented in 2008 [94], there have been many other payment systems emerging based on blockchains, such as Ripple [13], Stellar [46], and Ethereum [45]. The total number of cryptocurrencies in the world has soared to more than 20,200 in circulation currently [8]. However, despite the growing ecosystem, cryptocurrencies continue to operate in complete isolation from one another. *Interoperability*, i.e., allowing cryptocurrencies to be transferred across multiple blockchains, is currently one of the bottlenecks preventing the mass adoption of blockchain technology.

One solution of interoperability is to use sidechains [14]. The mainchain main-

tains a ledger and connects to the sidechain via a communication protocol that facilitates asset transfer between the mainchain and the sidechain [14]. However, it does not allow payments across sidechains. Another solution is to use a blockchain of blockchains where there is another level of blockchain recording and monitoring information and communication between different blockchains [71, 132]. However, introducing another blockchain leads to high difficulty in managing the blockchain and has high latency. Cross-chain bridges have been built in practice [5] to enable users to move funds from one blockchain to another. However, it requires users to have wallets and participate in both blockchains. In practice, users who are in different blockchains and want to make transactions might not want to participate in both blockchains. Currently, there are several commercial centralized exchange systems for executing fund transfers and exchanges across blockchains, such as Coinbase [29]. They work like banks, and users completely rely on them for token exchanges. Thus, these services break the trustless and decentralized property of blockchains.

On the other hand, *scalability* on the throughput of blockchains remains another huge problem with growing numbers of users and transactions [32, 102]. For instance, Bitcoin can only support 10 transactions per second at peak in 2022 [7]. Payment channel networks (PCNs), a type of peer-to-peer network, have been proposed to provide a high-throughput solution for blockchain [50, 140]. In a PCN, each user maintains payment channels to a few other users they trust. A transaction between two arbitrary users can be achieved by a multi-hop path of payment channels. Hence only opening and closing a payment channel need to be confirmed by the blockchain while

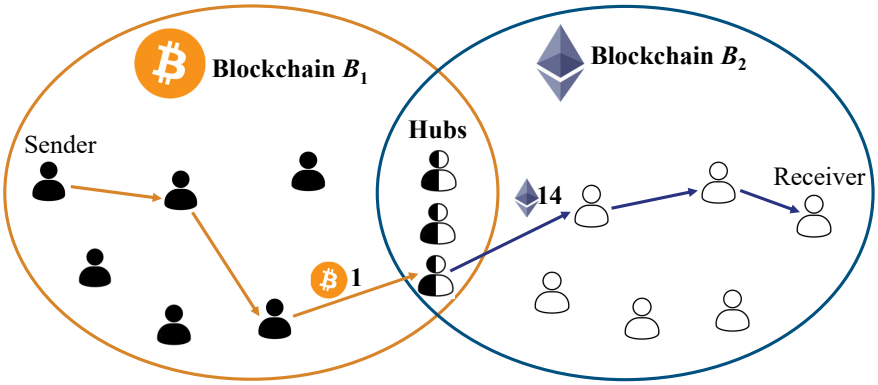


Figure 4.1: An example of the cross-chain transaction.

most transactions do not, which significantly reduces the blockchain load.

One intuitive idea to achieve both interoperability and scalability is to build a cross-chain PCN that allows two arbitrary users in different blockchains to make transactions via a multi-hop path. One might immediately think of Internet routing that may cross multiple domains. Similar to the gateway routers on the Internet, there could be some users who act as hubs that have wallets in two blockchains. A user in one blockchain can make a payment to a hub and the hub forwards the payment to another user in a different blockchain, as shown in Fig. 4.1. In this way, most users only need to hold tokens in one blockchain and still have the freedom to make transactions with any user in other blockchains. This design significantly broadens the user space of blockchain-based applications, compared to existing solutions that rely on one blockchain or require every user to have tokens of multiple blockchains.

However, there are multiple challenges in designing the *decentralized network architecture* of a cross-chain PCN. First, how to manage multi-currency wallets of the hubs such that malicious hubs cannot steal funds from other users. Since the relay hubs

are in both blockchains while the clients only have access to one of the blockchains, clients have no idea if the relay hub performs correctly in the other blockchain. The design of Hashed Time-Lock Contracts (HTLCs) [56, 102] and the recently proposed payment channel hubs [38,55,106,120] enable atomic operations, which means payment will either complete or the sender can get its funds back. However, they cannot solve another challenge: malicious hubs use low token exchange rates to attract users but fail all payment requests. In addition, as a decentralized network, how to make trustworthy hub information available and accessible to other users is another challenge. There is no solution in the literature that can address all the above challenges for a cross-chain PCN.

In this work, we present the first network architecture and the corresponding protocols of a cross-chain PCN, called XHub. XHub addresses the above challenges by achieving service availability, transaction atomicity, and auditability. In XHub, users who correctly follow the protocols will succeed in making payments or get profits from doing the services. In addition, trustworthy information about hubs will be managed in a decentralized manner and available to all users. **To our knowledge, no prior solution can achieve all these properties.** XHub does not propose new cryptographic methods. Instead, it includes a novel design that combines existing security protocols including multi-signature (multisig) wallets [31], Byzantine agreements with blockchains [50], simplified payment verification protocol [94], and anonymous atomic locks (A^2L) [120].

We conduct *prototype implementation* of machines that exchange messages and

run on two real blockchains, Bitcoin testnet3 [1] and Ethereum Sepolia testnet [9], as well as *large-scale simulations* based on real-world PCN topologies and transactions, Ripple [13] and Lightning [102]. The results show that the latency of cross-chain transactions is below 1 minute, and even if there are malicious hubs, dispute management takes no more than 3 minutes. The evaluation shows that XHubs could achieve a significantly higher success rate compared to the version without hub management protocols.

In summary, this work makes the following contributions:

- We propose, XHub, the first decentralized network architecture of a cross-chain PCN.
- We design a series of protocols, including the auditor communication protocol, hub registration protocol, transaction protocol, and hub management protocol to achieve the security properties.
- We use both prototype implementation and large-scale simulations to demonstrate the effectiveness of XHub.
- This work is an important step towards the big picture of a decentralized transaction system that connects a wide scope of users in different blockchains.

The rest of this paper is organized as follows. The network and security models are presented in Section 4.2. We describe the detailed design of XHub in Section 4.3. Section 4.4 provides the security analysis of XHub. Section 4.5 presents the evaluation results. Section 4.6 describes the related work. Section 4.7 concludes this work.

4.2 Network and Security Models

4.2.1 Network Model

The design of XHub considers the case of two blockchains, B_1 and B_2. For a system with more than two blockchains, we assume XHub is built between every pair of blockchains and leave the design of a network of networks to future work. Every user has one or more *wallets*, and each wallet includes tokens (funds) belonging to one blockchain. We use Ψ_1 and Ψ_2 to denote the names of the tokens in B_1 and B_2, respectively. There are three types of users:

Clients. Clients are users who want to make transactions, even if they are in different blockchains. Typically, clients are users with only one wallet and maintain tokens in one blockchain. They cannot directly talk to or transact with users in other blockchains.

Hubs. Users with wallets in both B_1 and B_2 can register as *hubs* that act as relays to forward payments between clients in B_1 and B_2, and get profits by charging transaction fees. For clients in different blockchains to make transactions, they both need to find a bi-directional payment channel or a path to a selected relay hub first. Each hub has an *exchange rate* for tokens. This is public knowledge on blockchains and can be periodically updated by hubs. Each hub is associated with a *reputation*, which is a score to measure its past behaviors. A hub needs to deposit collateral during registration. If the hub fails to provide the correct relay, XHub guarantees that all clients will not lose funds and clients can dispute the failed transactions to lower the

reputations of misbehaving hubs.

Auditors. The system also has a committee of *auditors*. The committee provides trustworthy management of hub information, including the exchange rates, reputations, and collateral. Each auditor is a user of both blockchains and can profit by correctly performing the committee services. Any user of both blockchains can register as an auditor as long as they put enough collateral in a multisig wallet maintained by the system. If an auditor performs maliciously or remains unresponsive for some time, they will lose all their collateral and be expelled from the system.

For each blockchain B_i, all users and channels form a *payment channel network* (PCN), modeled as a graph $G = (H, V_i, E_i)$, where H is the set of hubs, V_i is the set of clients in blockchain B_i, and E_i is the set of bi-directional payment channels between users. In a PCN, two users can make transactions if they share a bi-directional channel by committing a certain fund to open the channel, or find a multi-hop path of channels between them. Existing work assumes every client needs to open channels with all hubs the client will use, which leads to significant locked-in funds from hubs [86, 120] and significantly limits scalability. Hence XHub allows a client to connect a hub via a multi-hop path within the PCN and send or receive funds through the path. There are extensive studies on how to route a payment within the same blockchain [84, 110, 118, 142]. Hence, we consider the research of routing within a blockchain to be out of the scope of this paper and use a decentralized solution [142] to route payments between a pair of users in the same blockchain.

4.2.2 Security Model and Assumptions

We assume users can exchange messages through a traditional secure communication channel such as TLS. Information leakages among them are beyond the scope of our discussion. We assume the attackers can gain complete control of some but not all relay hubs. For each compromised hub, including controlling the stored funds and network communication, they may drop, modify and replay messages. An attacker may also delay or prevent a hub it controls from accessing the blockchains for an unbounded amount of time. All users, including clients and hubs, are rational, selfish, and potentially malicious, i.e., they may be malicious and attempt to steal funds and deviate from the payment protocol, if it benefits them. Hubs may intentionally fail ongoing token exchanges, keep funds from senders without exchanging and forwarding them to the receiver, or overcharge transaction fees. Malicious clients may collude to keep sending cross-chain transactions through a certain hub in one direction (e.g., always from B_1 to B_2). This attack will exhaust one type of token of the hub and make it fail to serve as a hub.

Following Byzantine fault-tolerant settings, we assume the proportion of adversaries is less than 33% of the total number of consensus participants of both blockchains and the committee of auditors. The delay Δ of posting consensus information to a blockchain depends on the block generation speed. The block generation speeds vary a lot among different blockchains and might change over time.

4.2.3 Design Objectives

XHub achieves the following design objectives.

Availability and auditability: If hubs, clients, and auditors follow the XHub protocols, they will succeed in making payments or get profits from doing the services. Auditability provides resilience to *denial-of-service* (DoS) and *counterfeiting* attacks. Although malicious hubs cannot steal funds from clients due to atomicity, they still can attract clients to select them as relays and fail to forward payments. Those failures will be detected by clients and clients can dispute them to the auditors. The auditors will decrease the reputation of any misbehaving hub based on the dispute results. Clients can find the latest reputations of all hubs and avoid selecting those with low reputations. Auditability also guarantees detection and penalty of counterfeiting, i.e., a client, hub, or auditor reporting incorrect information.

Atomicity: Atomicity ensures that in a cross-chain transaction, all the payments along the path will succeed together or all fail. It guarantees that honest users will not get any loss even if there exist malicious parties.

Unlinkability: Unlinkability ensures that if there are multiple cross-chain transactions happening through one hub, the hub cannot determine the sender-receiver pairs better than a random guess.

Performance: The main performance goal of cross-chain payment hubs includes low latency for cross-chain transactions, high scalability, and high success rate.

4.3 Protocol Design of XHub

This section presents the design of the protocols of XHub.

4.3.1 Design Overview

XHub is a network architecture that supports clients to select preferred hubs for cross-chain transactions while preserving security properties such as atomicity. The key idea is to maintain public-available and trustworthy information about the hubs, including their reputation scores, and exchange rates, with the help of a committee of decentralized auditors. XHub includes the following protocols.

1) Auditor communication protocol supports the functions of a committee of auditors, which register and manages hub information including their reputations and exchange rates, responds to queries of this information, and handles disputes from victim clients. A client or hub needs to broadcast to the whole committee of auditors or let an arbitrary auditor broadcast to other auditors.

2) Hub registration protocol allows a user of two blockchains to register as a hub.

3) Transaction protocol allows two clients in different blockchains to make a transaction via a hub. It includes three components: hub selection, intra-blockchain routing, and cross-chain transaction.

4) Hub management protocol allows auditors, hubs, and clients to manage trustworthy information of hubs, including their reputations and exchange rates.

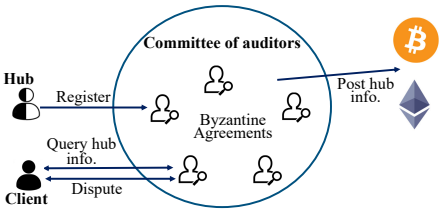


Figure 4.2: Auditor communication protocol.

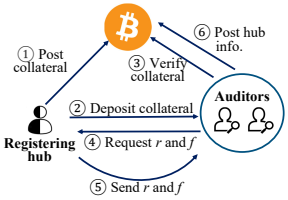


Figure 4.3: Hub registration protocol.

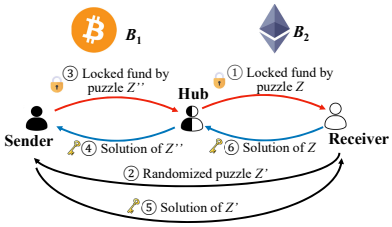


Figure 4.4: Cross-chain transaction by A^{2}L [120].

Misbehaving hubs will be penalized by lowering their reputations.

4.3.2 Auditor communication protocol

The committee of auditors manages the exchange rates, reputations, and collateral of all hubs together. All the auditors jointly maintain a multi-signature (multisig) wallet in each blockchain using the multisig protocol proposed in Bitcoin [31]. It performs a write operation to a blockchain when a threshold number of signatures are successfully collected from the auditors. In XHub, the threshold is set to be 2/3 of all auditors. Hence at least 2/3 of the auditors are required to sign each verification or update message for the information of a hub before sending the message to the blockchain. A user of both blockchains may register as an auditor to get profits when they correctly provide signatures. When a user registers as an auditor, they need to put collateral in the multisig wallet. Only the auditors who correctly sign the messages can get profit [50, 139], which is from the fees of hub registrations, exchange rate updates, reputation updates, and disputes.

Fig. 4.2 shows the communication processes related to the auditors. The auditor communication protocol achieves *Byzantine agreements*, because we do not assume

that all auditors are honest and available all the time. In XHub, we follow the design of Byzantine agreement protocol in Algorand [50], to achieve Byzantine fault tolerance among the auditors. This solution can tolerate up to 1/3 faulty auditors.

In addition, auditors also communicate with hubs, clients, and blockchains in other protocols of XHub including the hub registration and hub management protocols. Those processes will be detailed in later sections.

4.3.3 Hub registration protocol

Any user with wallets in both blockchains can register as a hub and get profit by forwarding payments. To register as a hub, a user deposits collateral to the multisig wallets of auditors in both two blockchains. The value of the collateral is pre-determined by the system and sufficient to cover dispute fees until its reputation reaches a very low value.

Figure 4.3 shows the hub registration protocol, which includes the following steps.

1) The registering user h first sends two transactions of putting collateral $\texttt{col}_h$ to the multisig wallets of the auditors in both blockchains B_1 and B_2 respectively.

2) h keeps monitoring the two blockchains until the deposit transactions are posted. It generates two corresponding proofs of inclusion and sends them to an arbitrary auditor. The proof is constructed by signing the transaction id and the transaction data with h's private key.

3) Upon receiving the proof, each auditor searches both blockchains to verify

if the deposit transactions are in the blockchains. XHub does not make each auditor download the whole blockchains to search for the transactions, because it costs a large amount of storage, computation, bandwidth, as well as long delays [24]. We use the *simplified payment verification* (SPV) protocol [94]. Instead of downloading the whole blockchain, auditors download only the header of each block, which contains the root of a Merkle tree [89] of transactions. To verify the correct inclusion of a deposit transaction, it is sufficient to provide the Merkle tree path from the root to the leaf containing the transaction of the corresponding block.

4) Once an auditor successfully verifies the two deposit transactions, it will request the registering hub for its exchange rate r and transaction fee f.

5) Upon receiving r and f, the auditor will broadcast to the committee a signed *New_Hub* message, including the hub's address, r, f, and a default reputation R_h.

6) When the auditor obtains signatures of the message from at least 2/3 of the auditors, it sends the *New_Hub* message with these signatures to the two blockchains respectively. The inclusion of the *New_Hub* message in the blockchains indicates the successful registration of the hub.

4.3.4 Transaction protocol

The transaction protocol supports that a sender in B_1 spends funds in an amount of ψ_1 and a receiver in B_2 receives funds in an amount of ψ_2. Note $\psi_2 = r_{12}(\psi_1 - f)$, where r_{12} is the exchange rate from B_1 to B_2 and f is the transaction fee charged by the hub.

Hub selection. When two clients want to make a cross-chain transaction, they need to first select a hub as the relay. There are three factors to consider: the exchange rate, transaction fee, and reputation score for each hub, and select the one that is considered ideal. The sender prefers a high exchange rate to pay fewer funds, a low transaction fee, and a high reputation – note an unsuccessful payment by a malicious hub does not make the sender's funds be stolen, but causes extra time to dispute and select another hub. Each client can self-define a 3-tuple (α, β, γ) to calculate a score $(\alpha R + \beta r_h - \gamma f_h)$ for each hub h and select the one with the highest score. After selecting the relay hub, the sender needs to confirm with the receiver to guarantee this hub is correctly registered in the other blockchain with the same reputation, exchange rate, and transaction fee information. If not, the sender needs to select another hub.

Intra-blockchain routing. After selecting the hub, the sender needs to make the payment of the corresponding amount of funds to the hub first, via a direct link or multi-hop path. We use a decentralized routing protocol [142] to find a payment path between two users in the same blockchain. Similarly, the hub uses the same routing protocol to find a path to the receiver in the other blockchain.

Cross-chain transaction. One important security requirement when a hub forwards payments between two blockchains is atomicity. Hubs are not necessarily honest and, in particular, they might attempt to steal money from clients, such as withholding funds from the sender without relaying them to the receiver, or overcharging in conversion fees than what they are allowed to. Atomicity guarantees that either a transaction of the correct amount is successful or payment funds go back to the

original sender. In addition, a further requirement is unlinkability, which ensures that if there are multiple cross-chain transactions happening through one hub, the hub cannot determine the sender-receiver pairs better than a random guess [51, 85]. The cross-chain transaction step of XHub is developed by a recently proposed solution called anonymous atomic locks (A^2L) [120] that achieves both atomicity and unlinkability at a single hub. A^2L cannot achieve availability or auditability: a malicious hub can keep failing payments without penalty.

We briefly present the protocol of a cross-chain transaction as shown in Fig. 4.4.

1) Suppose a sender A wants to send a cross-chain payment to a receiver B. The selected hub h first creates a fresh cryptographic puzzle Z and its corresponding solution. The hub then sends a locked fund $\texttt{Lock}(h, B, \psi_2, Z, T_2)$ with this puzzle to B, indicating that B can receive the payment in the amount of ψ_2 from the hub only if he provides the correct solution to Z within time T_2.

2) B randomizes Z into a new puzzle Z' using a randomness *rand* and sends Z' to A. A^2L applies a *homomorphic cryptographic scheme* to guarantee that the solution of Z can be obtained using the solution of Z' and *rand*, but one cannot link Z with Z'.

3) A re-randomizes Z' to a new puzzle Z'' with a randomness $rand'$ and sends a locked fund $\texttt{Lock}(h, A, \psi_1, Z'', T_1)$ to h, indicating that h can get the funds if it provides the solution of Z''.

4) h can solve Z'' using a universal trapdoor *tp* but cannot link Z'' with Z. h then provides the solution of Z'' and get the payment from A.

5) A computes the solution of Z' from the solution of Z'' and $rand'$ and sends

the solution to B.

6) B computes the solution of Z from the solution of Z' and $rand$ and sends the solution to h. Then B gets the payment from h.

In the end, B_2 receives funds in an amount of ψ_2. If any of the three parties fails to perform the correct operations, the whole transaction will fail but no one loses funds. In addition, the party that causes the failure will be detected by others and reported to auditors.

In the above protocol, the communication between A and h and B and h can be direct Internet packet exchanges, but the payments from h to B in 1) and from A to h in 3) could take one or more hops in the PCN, based on intra-blockchain routing. We further implement payment forwarding at every hop using A^2L for unlinkability within a blockchain.

In practice, observations show that the exchange rate between the two cryptocurrencies may be susceptible to strong fluctuations. Hence XHub locks the exchange rate once a transaction is set up until the transaction completes.

4.3.5 Hub management protocol

To achieve the availability of hub services and auditability of hub behaviors, the reputations of hubs should be correctly managed in XHub and accessible to clients.

For all new hubs that join XHub, they are assigned the same initial reputation $\hat{r}$. The reputation of a hub is updated for each time epoch. If the misbehavior of a hub is disputed by a client and verified by auditors in an epoch, the reputation of this hub

will decrease by 50%. If a hub keeps behaving correctly for 10 epochs, they can request the auditors to raise its reputation by 5% by paying a transaction fee. Note that there is always a latency to post the new reputation to the blockchains. Hence, clients may contact an arbitrary auditor for the latest reputation in the current epoch. Later clients can verify this in blockchains. If a hub does not respond to a transaction request from the beginning, it is not considered misbehavior because the hub can be offline. The clients can easily choose another hub. However, a hub cannot fail on Step 4) of the cross-chain transaction protocol, i.e., providing a correct solution of Z''. Besides, since auditors can be arbitrary users with enough collateral in both blockchains, an auditor could be offline or malicious. In order to mitigate the potential influence on client requests, clients can always make multiple queries to several auditors. In this way, they can detect the malicious hub that provided the wrong hub information. Once such misbehavior is identified, clients can submit disputes to other auditors for compensation and expel the malicious hub from the committee. Furthermore, if clients notice an auditor who remains unresponsive, they can also dispute to remove it from the committee.

Dispute handling. When a hub fails to provide a correct solution of Z'', intentionally or unintentionally, this event is guaranteed to be detected by the sender and receiver, based on the transaction protocol. Although the sender does not lose funds due to the atomicity of the protocol, the clients can dispute this event to decrease the reputation of the hub. In order to incentivize the clients and auditors to do so, if auditors successfully verify a misbehavior and update the reputation to the blockchains, this hub's collateral will be used to compensate the users and auditors.

We first discuss the case of the hub h providing a wrong solution of Z''. When a sender A detects that a hub h fails to perform the transaction protocol, e.g., h sends a wrong solution of Z'', A can send a dispute message $D_A = \{msg_{k_h}||Z''||col_A\}$ to an arbitrary auditor U, where msg_{k_h} is the signed message from h including the wrong solution of Z'' and col_A is A's collateral. If the dispute launched by A is incorrect, A will get punished by losing her collateral. This prevents clients from abusing the dispute process. Upon receiving the dispute message D_A, the auditor will broadcast D_A to other auditors and then verify if the solution from h is correct. If the solution is incorrect, the auditor signs a *dispute_success* message and broadcasts it to all auditors. When 2/3 of the auditors sign *dispute_success* messages, a reputation update can be posted to the blockchains.

The second case is that the hub h does not send the solution of Z'' to A. A will first try extra x attempts of requesting the solution of Z'', where x is a random integer in $[1, 5]$. If there is still no response, A sends a dispute message to an auditor. After receiving A's dispute, an auditor will also send Z'' to h and ask for the solution. Note the transaction protocol achieves unlinkability, hence h cannot tell whether a request is from A or an auditor. If h intentionally declines to send the solution of Z'', even with a very small probability, this misbehavior will eventually be detected by the auditor. The only way h can avoid being detected is by always responding with the correct solution of Z''. When 2/3 of the auditors detect that h declines to provide the solution, a reputation update can be posted to the blockchains.

Exchange rate management. Each hub can set its own exchange rate r. It

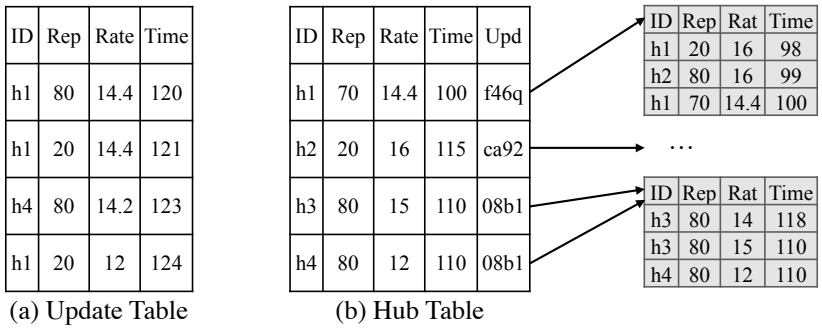


Figure 4.5: Example of the update table and hub information table on an auditor.

can simply set r to the market rate or use the exchange rate to balance its two tokens in both blockchains. Token balancing is necessary because if one token is depleted, the hub cannot relay any transactions between two blockchains. Assume that r_{12} is the rate such that the sender pays the hub 1 Ψ_1 token and the hub will pay the receiver r_{12} Ψ_2 tokens. We have $r_{21} = \frac{1}{r_{12}}$. For example, if a hub has more Ψ_2 tokens than Ψ_1 tokens, the exchange rate r_{12} can be higher than the market price to encourage clients to make payments in Ψ_1 tokens, while the exchange rate r_{21} can be lower than the market price.

Hub information management. In every epoch, all active hubs can update their exchange rate by sending the new one to auditors. Auditors maintain a local table that stores the exchange rate and reputation information of all hubs that will also be posted to the blockchains. Auditors might update the hub information every epoch with the current timestamp and their signatures. The updated hub table needs to be signed by at least 2/3 of the auditors to be put in the blockchains.

Using blockchain to maintain hub information raises two performance problems. 1) An update of hub information takes a considerable amount of time to be

available on a blockchain, due to its long processing time. Hence, the hub information on the blockchains might not be the latest. 2) The hub information, including hub reputation and exchange rates might change frequently, resulting in both a large amount of update requests and a large size of blockchain data to be posted.

To resolve the above problems, we propose a hybrid solution that combines the strengths of trustworthy information on blockchains and large storage capacity on auditors. Auditors locally maintain a hub table storing all hub information and an update table recording all the hub updates in the current epoch. At the end of each epoch, auditors send the digest of these two tables to the blockchains. Users query hub information directly from auditors and verify the correctness by checking the blockchains after the digest is processed by the blockchain and posted. In this way, users can access the latest hub information while still benefiting from the security and trust provided by blockchains. Furthermore, the combination of blockchains and auditors ensures that hub information can be updated and maintained in a timely and efficient manner, reducing the risk of failed transactions due to outdated or incorrect hub information. We provide an example of the hub table and update table in Fig. 4.5. The Hub table maintains the latest hub information at the end of the previous epoch. For each hub, the *Upd* entry indicates the time of the last update and includes a link to the corresponding update table in the corresponding epoch. The update table records every hub update in the current epoch. The update message needs to be signed by at least 2/3 of the auditors in order to be confirmed in blockchains.

User query. A user can query an arbitrary auditor for the hub information.

The auditor sends the current update table and hub table with the timestamp and its signature. The user searches the blockchain to check if this version of the tables has been confirmed. If this version exists in the blockchains, the user computes the table digest to verify its correctness. If this version does not exist, the user can ask the auditor for the most recent version of the tables that is available on the blockchains. If that version passes the digest checking, the user can temporarily trust the information from the auditor and wait for some time to check its correctness later. In the future, if the information is proved to be incorrect by the blockchains, the user can dispute the misbehavior of the auditor and the auditor will be removed by the committee.

4.3.6 Multiple Blockchains

XHub can be extended to the scenario of more than two blockchains, as long as there exist hubs and auditors between any two of them. For hubs that have wallets for multiple blockchains and are willing to work between all of them, they must register in each blockchain and set up exchange rates and transaction fees for every pair of blockchains. Hub reputation in each blockchain pair is independently managed by auditor committees responsible for that pair. Auditors who can serve between multiple blockchains, similarly, will participate in one auditor committee for each blockchain pair. And they have to maintain one hub information table for each pair.

4.4 Protocol Security Analysis

4.4.1 Availability

The availability of XHub refers to the property that any party who fails to follow the protocols will be detected by others and (eventually) be excluded from the system. Hence users can receive the services of XHub. First, we show that:

Proposition 4. *A registering hub that follows the registration protocol to lock collateral is guaranteed to be posted to the public as a valid hub even if there exist malicious auditors.*

This proposition holds based on the property of the multisig wallets. Suppose t_{lock} denotes the transaction of a registering hub locking funds to a multisig wallet of the auditors. t_{lock} is guaranteed to be confirmed if more than 2/3 of auditors are honest.The multisig protocol settles a transaction to a blockchain when signatures are successfully collected from more than 2/3 of the auditors [31]. Hence if more than 2/3 of auditors are honest, t_{lock} is guaranteed to be confirmed on the blockchain. When the transactions of both blockchains are confirmed, the hub is successfully registered.

Proposition 5. *All the clients have access to the information of a hub that is successfully registered.*

Clients always query hub information from several auditors. If they notice the hub information from some auditors is not consistent, they will follow the one with a correct digest in the blockchains. Since we assume 2/3 of the auditors are honest and

trusted, the digest of the correct hub information will receive signatures from at least 2/3 of the auditors and be successfully posted to the blockchain, based on Byzantine agreement protocol [50]. And clients can always get the latest hub information if they query a sufficient number of auditors.

Denial-of-Service (DoS) attacks. There are two main types of DoS attacks. 1) A hub accepts a payment request but fails to complete the transaction protocol by either providing a wrong solution of Z'' or declining to provide the solution. For a wrong solution, the sender may submit a dispute message to an auditor. If the solution is incorrect, the auditor signs a *dispute_success* message and broadcasts it to all auditors. When 2/3 of the auditors sign *dispute_success* messages, a reputation update can be posted to the blockchains. For a hub that declines to provide the solution, the dispute protocol allows an auditor to anonymously request the solution. Hence hub either always provides the solution or it will be detected by an auditor. 2) A malicious auditor intentionally refuses to provide the hub information. In this case, clients can simply query another auditor. Moreover, at the end of each time slot, auditors send the digest of both the update and hub tables to the blockchains. Even if an adversary refuses to verify and sign, the system can still rely on the remaining honest auditors to sign the digest and make sure it can be confirmed in the blockchain. If an adversary tries to perform a Sybil attack to submit false tables to the blockchain, it would need to register a large number of auditors to approve this message, which requires it to lock up a large amount of collateral to be effective, making this attack expensive and irrational.

Counterfeiting. When a client request hub information from auditors, a

malicious auditor might send a false hub table, creating counterfeit T'_h. However, at the end of the epoch, auditors will work together to put the hub table digest to the blockchains which require verification and signatures from more than 2/3 of auditors. If the malicious auditor tends to put a counterfeit T'_h in the blockchains, it has to compromise more than 2/3 of auditors in the system, which is impractical in the real world. The correct table digest $D(T_h)$ will ultimately be verified and confirmed in the blockchains. Auditability ensures that any client with read access to the blockchains can detect the misbehavior of the malicious auditor by comparing the table digest $D(T_h)$ retrieved from the blockchains with the $D(T'_h)$ computed using T'_h got from the auditor. If two digests do not match, clients can submit proof showing the auditor manipulates false tables.

Stale table. During the middle of an epoch, the update table remains incomplete. When clients query the update table, the current version of the update table is T'_u. But the adversary might send a stale version T^s_u to its own benefit. For example, as shown in Fig.4.5, at time 122, the current update table T'_{upd} shows that hub h_1 has a reputation of 20. But the malicious auditor could collude with the hub h_1 and send a stale update table T^s_u at time 120 when h_1 had a reputation of 80. After a while, the clients retrieve the digest of the update table $D(T_u)$ from the blockchains and verify the correctness of T^s_u by checking its inclusion in the T_u. Since T^s_u is not fabricated, the adversary can still pass this inclusion test, and clients remain unaware of the stale table T^s_u. To prevent this attack, the system requires that all parties include a timestamp when querying or transmitting hub information. When a client queries the hub informa-

tion at time t_1, they construct a query with timestamp $Q(t_1)$ and send it to an auditor. The auditor constructs the response by including the latest update table T'_u at t_1, the received request $Q(t_1)$, and its signature. Consequently, the client can confirm that the response corresponds to their request at time t_1 and cannot deny the timestamp. At the end of the epoch, when the auditor sends the digest of the update and hub table to the blockchains or helps to verify the table digest, it will also send a current version of the update table T_u to the client. At the client side, after verifying the correctness of T_u by checking the $D(T_u)$ from the blockchains, the client compares its locally stored T'_u with T_u. If the subset of T_u that all the entries satisfy $t <= t_1$, denoted as $T_{u_{t_1}}$, is larger than the table T'_u, T'_u is detected to be a stale table.

4.4.2 Atomicity and unlinkability

Atomicity guarantees that honest parties will make transactions successfully or get all their money back, which ensures balance security for the involved parties. The atomicity and unlinkability properties of XHub relies on the security of the randomized puzzle scheme, as proved in A2L [120]. The hub can only provide the solution of Z'' in order to receive the funds and Z'' will be used by A to generate a solution of Z', which will be used by B to unlock the funds from h. The clients can only steal the funds if they can generate a correct solution to the randomized puzzle, without paying h. However, this breaks the discrete logarithm (DLOG) problem, which is believed to be a hard problem [120]. In addition, the unlinkability is achieved by the fact that the adversary cannot break the indistinguishability of the adaptor signature scheme [120].

And we skip the detailed proof due to space limit.

We now show that the Cross-chain payment hub achieves funds security using the Universal Composability (UC) framework [25] similar to prior work [39, 81] which is based on a system of interactive Turing machines (ITMs). The UC framework includes parties executing the protocol in the real world, ideal functionalities performed by idealized third parties, and a set of adversaries $\mathcal{A}$. A protocol is said to be UC-secure if the real-world execution of the protocol cannot be distinguished from the idealized protocol execution by the environment. The UC framework includes an environment ε, which represents the external world. The environment chooses the inputs given to each ITMs in the system and observes the outputs. The framework also includes honest parties who follow the protocol, and a set of adversary $\mathcal{A}$ who try to break the security of the system. Besides real-world functionalities, the framework also includes ideal functionalities, which act as idealized third parties, and implement some target specifications. They exhibit the desired properties of the protocol. We define the ideal world functionality F_{atomic} for the transaction protocol. The clients and relay hubs interact with F_{atomic} implemented by a trusted third party to perform the cross-chain transactions. F_{atomic} manages a list P to keep track of the cryptographic puzzles and timelocks, and another list K to keep track of the valid key to the puzzles. Atomicity for a cross-chain transaction means that a puzzle can only be solved if there is a corresponding execution of the solution for that puzzle. This is enforced by F_{atomic} because it keeps track of the puzzles in the list P, and checks whether the puzzle matches one of the existing entries in the list P that has already been solved. Since the puzzles can

only be solved by a PuzzleSolver function inside F_{atomic} which is trusted, this ensures that PuzzleSolver must be called before checking the validity of the puzzle solution in order for it to succeed.

4.5 Experimental Evaluation

We evaluate the performance and security of XHub using both simulations and prototype implementation.

4.5.1 Methodology

We implement the system prototype of XHub on two blockchains, Bitcoin testnet3 [1] and Ethereum Sepolia testnet [9]. The testnets are real blockchains but the tokens do not have any value. They are used for software testing and research purposes. Both Bitcoin and Ethereum use ECDSA with the secp256k1 Koblitz curve [69, 107], proving native support for the corresponding cryptographic operations. The prototype is mainly used for evaluating the real latency to set up a hub, make transactions, and dispute management. The transaction protocol of XHub is built based on the RELIC library [11] for the cryptographic operations and on the PARI library [121] for the arithmetic operations in class groups.

Our large-scale simulations use two real PCN topology and transaction datasets: Ripple [13] and Lightning [102]. We treat the Ripple data as transactions sent by clients in Ripple and sent to clients in the Lightning network. And Lightning data as transactions from clients in Lightning to those in Ripple. For Ripple, we use the data from

January 2021 to December 2021 and get the network with 1,783 users in our simulation. For Lightning, we get the network with 3,519 nodes on one day in January 2022. We assume 500 users are both in Ripple and Lightning networks, and they can volunteer to be relay hubs. We generate payments from Ripple by randomly sampling the Ripple transactions for the sender-receiver pair in Ripple and Lightning respectively. Due to the lack of user information in the Lightning network, we randomly sample the transaction volumes and sender-receiver pairs for transactions from Lightning. We generate cross-chain transactions by randomly selecting transactions from the above two groups of payments.

Metrics. We use average processing latency to evaluate the performance of the prototype system. The processing latency of payment is calculated as the total delay from hub selection to fulfilling a transaction. Similar to prior work [110, 126], we also use the transaction success rate as an evaluation metric for resource utilization, defined as the percentage of successful payments whose demands are met overall generated payments. Note a transaction may fail due to limited funds on payment channels. We report the average results over 10 runs, each of which includes hundreds of communication pairs.

4.5.2 Results of cross-chain transactions in real systems

We conduct a testbed evaluation with the XHub prototype of 7 machines including 2 clients, 1 hub, and 4 auditors, running on two real blockchains. In the experiments, we construct a payment path from a client in the Bitcoin testnet to a

user in the Ethereum testnet via a hub. We execute several transactions between them. We also have a group of 4 auditors that are both registered in Bitcoin and Ethereum testnets. Table 4.1 and 4.2 show the performance and cost of different operations of XHub respectively. Hub registrations take approximately 75 minutes with 1, 3, or 4 auditors, and hub information updates take 60 minutes with 1, 3, or 4 auditors. The reason why both operations take a relatively long time to complete is that they require writing operations to real blockchains. For example, each hub registration incurs two transactions in the blockchains: one is to lock collateral to the blockchain and the other is to post the new hub information to the blockchain by the auditors. **The time waiting for confirmation on the blockchains contributes to more than 99.9% of the latency while the XHub protocol execution time is negligible compared to it. Also, both operations can be executed in parallel to save latency.** For example, multiple hubs can register at the same time and the whole process still takes around 75 minutes. The time of a new hub proving the locked collateral to auditors and auditors verifying this information only takes around 105 ms and 729 ms respectively. Hence the number of auditors does not play an important part in the processing latency. Hub information update is to update the reputation and exchange rate information on the blockchains. This information is kept in one table, and this table will be updated every epoch by the auditors. As long as one relay hub has information changes in an epoch, the auditors will post the digest of the new table to the blockchains, which takes around 60 mins. The latency again is dominated by the transaction processing time in the blockchains. The number of auditors does not affect it.

Table 4.1: Cross-chain payment hub performance of XHub

Operation and Latency	1 Auditor	3 Auditors	4 Auditors
On chain:			
Hub registrations	75 mins	75 mins	75 mins
Hub info updates	60 mins	60 mins	60 mins
Off-chain:			
Cross-chain transaction	592 ms	661 ms	748 ms
Dispute management	1,903 ms	2,439 ms	2,987 ms

Table 4.2: Cost of XHub operations on the two blockchains

	ETH	BTCTEST
Hub registration	0.000032	0.000057
Reputation update	0.000129	0.000158
Exchange fee update	0.000138	0.000174
Dispute	0.000031	0.000092

Off-chain operations of XHub that do not involve blockchain transactions have much less latency. The time to make a cross-chain transaction is the time to perform the atomic swap protocol. Before initializing the transaction, two clients need to negotiate and determine the hub with the help of auditors. So it is the network latency that leads to the processing time difference with the varying number of auditors. For dispute management, users need to first send a dispute message to all the auditors, then the auditors check deposits and verify the malicious behaviors, and make the penalty. It requires the participation of all the auditors in every step, and the penalty can be executed only if more than 2/3 of the auditors approve. Even if the system has 4 auditors, the dispute management time is still less than 3 minutes.

4.5.3 Results of exchange rate management

We use simulations to demonstrate that the dynamic exchange rate in XHub helps to improve the transaction success rate. A hub can use dynamic rates to encourage clients to make payments in one of the tokens to achieve token balancing, while fixed rates may cause one type of token to be exhausted. We compared it with the version of fixed exchange rates, in which all hubs have the same standard exchange rates, constant over time. We assume there is no malicious hub in the system. Thus, each hub has an equal likelihood to be chosen to conduct cross-chain transactions by sender-receiver pairs. Fig. 4.6 shows that XHub with dynamic exchange rates always achieves higher transaction success rates compared to that using the fixed rate, by varying the numbers of transactions and relay hubs. In Fig. 4.6(a), we set the number of hubs to 200 and vary the number of transactions, and in Fig. 4.6(b) we set the number of transactions to 1,000 and vary the number of hubs. With a fixed exchange rate, a hub might become imbalanced in two blockchain tokens when the transactions across it are higher in one direction than the other. Eventually, the hub runs out of one token and cannot support further payments in this direction. On the contrary, in XHub, with adaptive exchange rates, hubs set a good rate to attract the transactions which can make their tokens balance. Hubs are less likely to run out of their funds, and thus, can serve more transactions.

4.5.4 Results of reputation management

We evaluate the XHub with different proportions and different behaviors of malicious hubs, and compare the results with and without our reputation mechanism. In this experiment, we use 200 relay hubs and 1,000 transactions in total. We consider the following types of malicious behavior of hubs: 1) hubs provide rational exchange rates, but fail all transactions going through them deliberately; 2) hubs provide rational exchange rates, but fail transactions in one direction; 3) hubs provide rational exchange rates, but fail small transactions below a threshold which provide less profit; 4) hubs provide extremely low exchange rates to attract more transactions, but fail all those transactions; 5) hubs provide extremely low exchange rates to attract more transactions going through them, but fail some of them according to their own interests. Fig. 4.7(a) shows the performance of XHub varying with the percentage of malicious hubs, ranging from 0% to 50%, with and without reputation management, under different malicious attacks. The figure shows the results of Attacks 1 to 5 without reputation management and XHub under Attack 4 (the one that causes the lowest success rate with no reputation management). Without the reputation mechanism, the success rate of XHub decreases a lot with the growing percentage of malicious hubs. When the percentage of malicious hubs is below 15%, the performance of XHub with reputation management is similar to that with no malicious hub. When the percentage of malicious hubs achieves 50% (unlikely to happen in practice), the success rate of XHub is still above 60%, while the success rate without reputation management is only 30% for

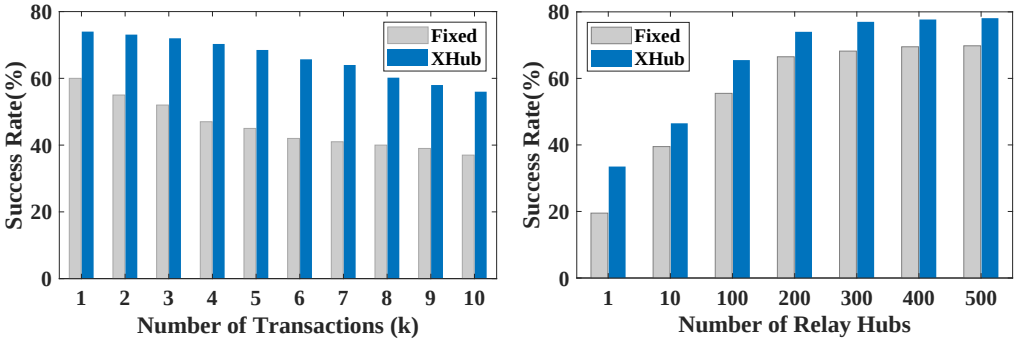
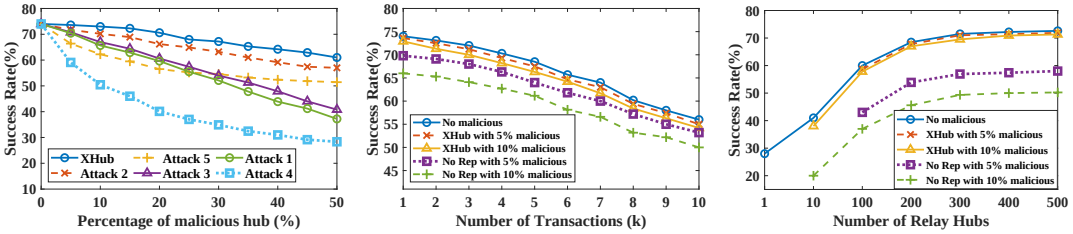


Figure 4.6: The success rates with fixed and adaptive exchange rates.



(a) Varying the percentage of malicious hubs

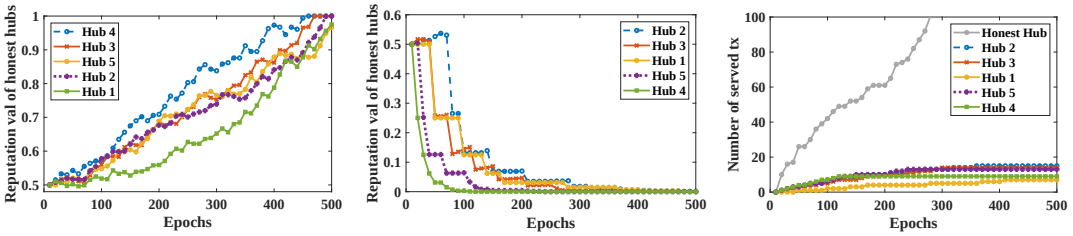
(b) Varying the number of transactions

(c) Varying the number of hubs

Figure 4.7: The success rate with the varying number of transactions and relay hubs with and without reputation mechanism.

Attack 4. In Fig. 4.7(b) and Fig. 4.7(c), we provide the transaction success rates by varying the numbers of transactions and hubs. Specifically, we focus on cases where 5% and 10% of the hubs are malicious and perform Attack 4. We believe this focus is reflective of more realistic systems where attackers tend to fail as many transactions as possible, and it is uncommon to encounter a high percentage of malicious hubs. With reputation management, the success rate of 5% and 10% malicious hubs are both close to that of no malicious hub. However, without reputation management, the success rate is significantly lower.

We also monitor the change of reputation scores for both honest and malicious



(a) Reputation changes of 5 honest hubs

(b) Reputation changes of 5 malicious hubs

(c) Number of served transactions

Figure 4.8: Reputation changes and the number of served transactions of honest and malicious hubs.

hubs. The initial reputation of each hub is set to be 0.5. The reputation value will be dynamically updated according to the hub behaviors. The percentage of malicious hubs in this set of experiments is set to 5%. Fig. 4.8(a) shows the reputation changes for 5 randomly chosen honest hubs and Fig. 4.8(b) shows that of 5 randomly chosen malicious hubs. Hub 1 to 5 in Fig. 4.8(b) denotes the corresponding malicious behavior from the aforementioned 5 types. We find that honest hubs always gradually achieve the maximum reputation value after 500 epochs, even if they might not be able to fulfill some cross-chain transactions due to their fund limits. On the other hand, the reputation value of malicious hubs will shortly decrease to 0, and they will be excluded from the system. Hubs 4 and 5 experience a rapid decrease in reputation as they attract a larger volume of transactions, and thus will be detected immediately and excluded from the system. While transactions do not frequently route through Hub 1, once it is selected by any transaction, Hub 1 will be detected and get a low reputation. For Hubs 2 and 3, even though they do not misbehave all the time, they can still be detected and penalized with low reputations. Fig. 4.8(c) shows the number of total transactions served by these

5 malicious hubs and one randomly selected honest hub. After a short duration, those malicious hubs receive fewer cross-chain transaction requests and eventually cannot serve any transaction, while the honest hub can keep serving cross-chain transactions.

4.6 Related Work

Blockchain interoperability, i.e., how to enable multiple parties to exchange tokens across multiple blockchains has been an important problem that attracts increasing attention. Centralized exchange systems are widely used such as Coinbase [29]. However, these services require trust and, therefore, undermine the decentralized nature of the blockchains. Atomic cross-chain swaps (ACCS) is a mechanism to perform a trustless cross-chain transfer based on hashed timelocks [56, 102]. Although ACCS enables trustless exchanges, it relies on all parties monitoring the blockchain throughout the exchange to ensure security. Moreover, ACCS is vulnerable to packet and transaction memory-pool sniffing, allowing an adversary to exploit blockchain race conditions to steal funds. Many decentralized exchanges remove the need to trust centralized intermediaries for blockchain transfers through the use of ACCS [72, 99, 127]. However, they only enable the exchange of cryptocurrency assets within a single blockchain [14]. Interledger [122] is a protocol that supports multi-hop payments where each link represents a payment channel defined in a different cryptocurrency. It also relies on the HTLC contract, aiming to ensure payment atomicity across different hops. However, the HTLC contract breaks the unlinkability property and has privacy issues. XCLAIM [138]

defines the notion of cryptocurrency-backed assets for blockchains and builds a secure system to construct cryptocurrency-backed assets without trusted intermediaries. It enables one cryptocurrency one-to-one backed by other cryptocurrencies. It suffers from the scalability problem that it cannot support a large number of users back up and construct different cryptocurrency-backed assets. It also leads to large lock-in funds if a user wants to participate in many different blockchains, requiring one backed asset for each individual blockchain. zkBridge [133] designs a trustless cross-chain bridge to move users' funds from one blockchain to another. The similar problem as XCLAIM also exists, large lock-in funds for multiple blockchains. Moreover, cross-chain bridges always require users to have wallets in both blockchains and monitor them. However, the overall complexity of managing funds across multiple blockchains can be overwhelming for some users. Different from previous works, XHub is the first to develop the network architecture of flexible cross-chain payments, which considers the problem of hub selection and management and service availability.

4.7 Conclusion

Extending the concept of PCNs to support multi-hop paths across multiple blockchains and resolve both interoperability and throughput scalability is an attractive idea. XHub is the first cross-chain PCN architecture to achieve service availability, transaction atomicity, and auditability. We design a series of protocols, including the auditor communication protocol, hub registration protocol, transaction protocol, and

hub management protocol. Both prototype implementation and large-scale simulations show that XHub has a small latency for cross-chain payments and can achieve a significantly higher success rate compared to the version without hub management protocols. We expect XHub would be an important step for a decentralized transaction system that connects a wide scope of users in different blockchains

Chapter 5

Secure Decentralized Learning with Blockchain

5.1 Introduction

Federated Learning (FL) [88] is a distributed machine learning (ML) paradigm that allows training ML models across numerous distributed devices, such as mobile and IoT devices. Those edge devices hold their data locally and collaboratively perform training tasks without directly sharing training data among them to ensure privacy. The trained ML models are then aggregated on a central server, called the aggregator. The aggregator first distributes a global model to clients. Each client trains the model locally using its own data and generates a model update, which is then sent back to the aggregator. The aggregator aggregates these updates to update the global model, and distribute it to clients for further training. FL preserves data privacy by enabling

decentralized model training [83, 129], saving communication costs by avoiding moving raw data, and reducing computational costs by leveraging the computing resource of each device. However, the existence of the centralized aggregator makes FL vulnerable to a single point of failure [78]. Once the centralized aggregator is compromised, the whole FL system will fail. Also, the aggregator that frequently exchanges models with clients can become the bottleneck of the system.

The recently proposed concept of Decentralized federated learning (DFL) [54, 79, 82, 119] provides a solution for aforementioned problems by removing the involvement of the central server. In a DFL system, instead of communicating with a central aggregator, clients directly exchange model updates with a subset of other clients, also known as their "neighbors", using P2P communication. Clients keep exchanging model updates until their local models converge to a model that reflects the features of data from all clients. Thus, DFL improves the limitations of having a single point of failure, trust dependencies, and bottlenecks on the server side in the traditional FL. However, DFL still has some challenges such as malicious clients, low-quality models, and the lack of incentives, which undermines the reliability of the whole system. Given the large number of participants in the DFL system, it is unrealistic to simply assume all the clients are honest and follow the protocols to do the training correctly. Therefore, there may exist malicious clients sharing false model updates about their local training results. Also, some clients with low-quality models might also affect the performance of their neighbors with high-quality models, and these errors may be further propagated in the whole network. Besides, how to motivate data owners to participate in the system

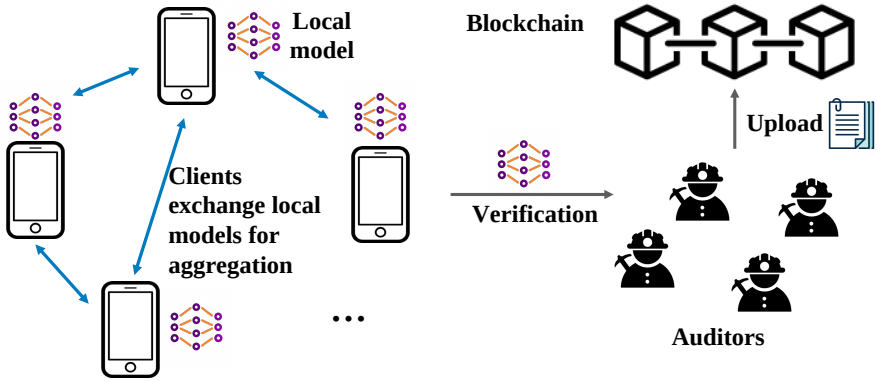


Figure 5.1: Blockchain-based Decentralized Federated Learning: clients form a peer-to-peer overlay network to exchange models, and auditors are responsible for model verification.

and continuously contribute their data to the FL model remains a challenge.

Blockchain, as a distributed ledger technology built on a peer-to-peer network, provides a possible solution for the security and incentive issue in the DFL system [115], as shown in Fig. 5.1. It provides trust by allowing all participants to verify transactions submitted to the blockchain with its underlying provable cryptography and consensus protocol. Every participant could verify each model update before it can be aggregated and confirmed in the blockchain. However, directly storing all model updates on the blockchain is not feasible due to the significant costs incurred by data storage and computation. And pushing large model data to the blockchain has the problem of heavy latency, limited block size, and transaction size. Thus, we introduce an auditor committee and the reputation mechanism for model verification. Auditors are responsible for managing the clients' reputations according to the quality of their model updates. Instead of models or gradients, auditors will only put the clients' reputations on the blockchain and update periodically.

In this work, we introduce BDFL, a Blockchain-based Decentralized Federated Learning system with an incentive mechanism and reputation model. We introduce a set of auditors for model verification, and honest clients can receive incentives while clients providing malicious models will be punished. In our reputation model, clients' reputation scores are assigned by auditors according to their model verification results, which will be updated periodically. Clients with higher reputations will have a higher probability for their model updates to be accepted by neighbors, and thus gain more profits from the system.

In summary, this work makes the following contributions:

- We design BDFL, the first blockchain-based fully decentralized federated learning system for model verification with high learning accuracy and system robustness.
- We design and implement the BDFL protocol suite. We introduce an incentive mechanism to encourage clients to participate in the model exchange, and a reputation model to evaluate the trustworthiness of each client to avoid malicious model updates from attackers.
- We evaluate BDFL using experiments on real ML datasets. We find that BDFL achieves a high model accuracy and fast convergence. It also has strong resilience to client dynamic and malicious model updates.

The rest of this paper is organized as follows. The system overview and model are presented in Section 5.2. We describe a protocol overview in Section 5.3 and the detailed design of the BDFL in Section 5.4. Section 5.5 presents the evaluation results

of our protocol. Section 5.6 describes the related work. Section 5.7 concludes this work.

5.2 Overview

5.2.1 Network Model

We consider a decentralized learning system with a large number of clients, in which clients can join or leave anytime. Those clients, such as edge devices, are willing to train models using their own data locally, and exchange model updates with their neighbors to get some profits. There are also some auditors in the system who are responsible for model verification. They are a group of nodes that have read and write access to the blockchain, similar to the miners in common blockchain systems. They work as a committee to verify the model updates using public validation data in the system, and generate new blocks of reputation information. Honest auditors will receive rewards for correct verification, and dishonest ones can get punishment if detected.

Our network model is divided into two parts, the decentralized federated learning network, and the blockchain network. We model the BDFL network as an undirected graph $G = (V, E)$, where V is the set of clients, and each link $e = (u, v) \in E$ represents that client u and v are neighbors, and can directly exchange local ML models. We assume clients have equal roles in the BDFL network and similar numbers of neighbors. Clients have read access to the blockchain, and can communicate with auditors for model verification. In the blockchain network, the clients' and auditors' identities and model verification information are recorded in the blockchain in the form of trans-

actions. To encourage the participation of more users, auditors successfully performing verification and clients honestly providing model updates will receive incentives, which are guaranteed by smart contracts in the blockchain. Malicious participants can also be identified by the blockchain to protect the quality of the overall model.

5.2.2 Blockchain Model and Assumptions

The blockchain in the BDFL system should support smart contracts, which are responsible for managing client reputation and ensuring auditors behave appropriately. Both clients and auditors need to register on the blockchain first to join the BDFL system. Since we use the blockchain as the underlying root of trust, if it is compromised by an adversary, the correct functionality of the BDFL system cannot be guaranteed. Therefore, we assume that the proportion of consensus participants corrupted by an adversary for the blockchain is bounded by a threshold to ensure safety and liveness for the underlying blockchains. Following Byzantine fault-tolerant settings, we assume the proportion of adversaries is less than 33% of the total number of consensus participants [49].

5.2.3 Attacker Model

We assume the attackers can potentially gain physical access to some clients in which the data and model are stored, and complete control of their network connections. They may want to destroy the global model by performing poisoning attacks. They will train models using false data and exchanging adversarial updates with neighbors [17,48].

They may also delay or prevent the client they control from accessing the blockchain for an unbounded amount of time. They are curious about clients' private information, and can perform information leakage attacks by observing the model updates, and then recover details about clients' training data [57, 111]. To prevent such attacks during model update transmission, clients can send differentially-private updates to mask their gradient [10, 37]. We assume most of the auditors will correctly follow the protocol, and part of them may be corrupted in a Sybil attack. But attackers cannot control more than 1/3 of total auditors according to Byzantine fault tolerance. We assume that all clients in the BDFL system could securely conduct initialization, in which they can correctly obtain the first version of the global model.Auditors have relatively equivalent computation resource [137], and rational public validation data to perform model update verification.

5.2.4 Requirements

Security: The main security requirement of the BDFL system is that it should enable model update exchanges between clients safely and correctly. We consider the security of both clients and auditors. Honest clients who provide the correct model updates will be acknowledged with profits and gain a better reputation, while clients with malicious model updates will ultimately be detected, and has a dramatic drop in their reputation. Whenever a client receives a model update from their neighbor, they can verify the correctness of the update with the help of auditors and the blockchain. If the model update successfully passes the verification, the client will accept it to further

aggregate with their own model locally. If the model is considered to be malicious, the client will reject this update. As for the auditors, if an auditor claims an incorrect model update could pass verification, this behavior will certainly be detected within the blockchain, as it requires the approval of more than 2/3 of the auditors for the verification result to be confirmed in the blockchain. The malicious auditor will eventually lose all their collateral as ensured by the smart contract.

Auditability: Any clients with read access to blockchain are able to get the latest reputation of all clients. Clients can also audit the model updates from their neighbors with the help of auditors and the blockchain.

Privacy: The BDFL system should be able to keep client training data private by preventing information leakage attacks. The auditors received masked model updates from clients can successfully verify the correctness of the model, but cannot learn any information on the clients' training data.

Robustness: In the BDFL system, the local models on the honest clients should eventually converge to a model that reflects the features of data from all clients with high accuracy. The system should keep robust under attacks, which means, even if there exist attackers, the system should still achieve equivalent model accuracy. Moreover, the DFL network should be resilient to client dynamics such as client joins, leaves, and failures.

5.3 Design Overview

In a fully decentralized overlay network for DFL, the BDFL protocol suite provides two sets of protocols for clients: 1) a DFL network *Topology Maintenance Protocol* to build the overlay network and recover it from churn; 2) a *Model Exchange Protocol* which includes model verification to achieve fast model convergence for heterogeneous clients and asynchronous communication. Table 5.1 shows the API that BDFL provides to clients. BDFL generates unique identifiers and an initialized reputation for each client, e.g., when a client u joins the DFL network and registers in the BDFL system for model exchange, a unique identifier u_{id} is returned as a handle to be used in reputation management and subsequent API calls.

The BDFL protocols work with any overlay topology and we apply a recently proposed overlay topology as a case to study BDFL [58], which is based on near-random regular graphs (RRGs) [135].

In P2P model exchanges, a client with low-quality local models might pollute its neighbors with high-quality models. This could lead to further propagation of these errors throughout the overlay network. Thus, every time when a client receives a model update from its neighbor, the client will first self-evaluate the confidence of this model. If the reputation of this neighbor is too low, the client can directly reject this model update. If the client feels the model is unreliable, the Model Exchange Protocol allows them to request model verification from auditors. Auditors then use an anonymized public validation dataset to do the model verification [116]. If the computed accuracy

by the auditors drops a lot compared to its previous model accuracy, this model update is considered to fail the verification. Auditors will announce the verification result to the corresponding client, and reduce the reputation of the client who provided this model update. Otherwise, the client performs the model aggregation locally after receiving the correct verification result from auditors.

DFL Topology. In BDFL, each client is identified by a set of *virtual* coordinates C, which is an L-dimensional vector $< x_1, x_2, ..., x_L >$. Each element x_l is a random real number computed as $H(IP_x|i)$ where H is a publicly known hash function and IP_x is x's IP address. We create L virtual ring spaces [135] such that each client in the i-th ring space is virtually positioned based on its coordinate x_i. In each virtual ring space, every client has two adjacent clients based on their coordinates, forming overlay neighbors for model exchanges. Each client can have a maximum of $2L$ neighbors, with L serving as a trade-off parameter between communication and convergence. A larger L leads to more model exchanges but also increases the communication cost.

Auditors. Auditors are groups of nodes that have read and write access to the blockchain. They work jointly with the blockchain for client registration, model verification, and reputation management. The system leverages a public smart contract (aSC) to maintain an auditor list and ensure the correct behavior of the auditors. They are required to lock some collateral to be registered with this smart contract, i.e., aSC can verify the auditor's digital signature and knows the auditor's public key. We assume the majority of the auditors are reliable. They are willing to follow the protocol to get profits, and punish malicious auditors for misbehavior. For each client's model

verification request and reputation update, a minimum number of auditors are required to sign the result before packing the update into the blockchains, thus tolerating a fixed percentage of auditors' failures to some degree. A common way to address such concerns is to use Byzantine-fault tolerant protocols [86]. For example, the auditors could use a BFT consensus such as [26] to stay up to date with all the coming requests from users. Such a solution can tolerate up to 1/3 faulty auditors. Thus, in BDFL, the smart contract defines that at least 2/3 of the auditors are required to sign each verification or update request before sending them to the blockchain, and only the auditors who correctly sign the request can get rewards.

Model Update. Different from FL, BDFL does not require a central server for model aggregation. Instead, every client can run the model aggregation locally using the model updates gathered from its neighbors. Once clients successfully prepare models locally, they can collect model updates from their neighbors for further aggregation. Clients always reject model updates from neighbors with low reputations. Clients then query auditors for model verification on the rest of the model updates. After verification by the auditors, clients run the model aggregation on all the correct model updates.

Reputation. In BDFL, each client is assigned a reputation value by auditors which reflects its trustworthiness. Clients should have a higher possibility to accept model updates from honest clients, and reject those from malicious ones. To prevent poisoning attacks from malicious clients, every time when auditors detect a model update of low accuracy, the auditors will decrease the reputations of the corresponding misbehaving client. On the other hand, honest clients will gain a reputation increase

Table 5.1: BDFL API

BDFL APIs	Inputs	Outputs	API Description
Topology Maintenance:			
`join_network`	u, v	u_{id}, rep_u	Join the DFL network to find its correct neighbors and register in the BDFL system.
`leave_network`	u, u_{id}	-	Terminate model exchange and leave the network.
`maintenance`	u	-	Maintain the correct DFL network topology by checking the liveness of all u's neighbors.
Model Exchange:			
`local_verify`	u, v, ω_v	Boolen	u locally pre-evaluate the accuracy of model update ω_v from their neighbor v.
`request_verify`	u, v, ω_v	σ_{ω_v}	u request model verification on the model update ω_v from their neighbor v.
`aggregate_model`	ω_u, ω_v	ω'	u locally aggregates the models from their neighbors.

by providing good model updates.

5.4 Protocol Design

This section describes the design of BDFL protocols.

5.4.1 Topology Maintenance

The Topology Maintenance Protocol in BDFL system includes `join_network`, `leave_network` and `maintenance` as shown in Table 5.1.

Join. Assume we have a correct DFL network topology with n clients currently. A new client u now boots up and wants to join the BDFL system for future model exchange. Before joining the DFL network, u has to know one existing client v in the overlay. u assigns itself a random coordinate in the virtual ring spaces as its position. Then it sends join requests to its neighbor v, and tries to find all its neighbors in the network. To achieve this, u lets v send a *Neighbor_discovery* message which includes u's IP address to the current DFL network using greedy routing to u's location in each ring space respectively. *Neighbor_discovery* stops at the client w who is closest to u. In each virtual ring space, w finds the adjacent node p from its two adjacent nodes to insert v in between according to u's coordinate.

Leave. When a client wants to leave the system, `leave_network` should guarantee that the BDFL system can still maintain a correct DFL network topology. Assume client u wants to leave by running `leave_network`. u sends messages to its two adjacent clients in each virtual ring space, and tells them to add each other to their neighbor sets.

5.4.2 System Maintenance

Auditors are responsible to help maintain the BDFL system. They will track client information such as client identity and reputation. They will update this information to the blockchain periodically, and all the clients can easily check this information by reading the blockchain. To do this, each auditor maintains a local table to record client information. It includes three parts: the reputations of all the valid clients, which are updated according to the quality of models provided by clients; a joining client set, which is the clients who join the system after the last update; and a leaving client set. This leaving client set includes two kinds of clients. One is the clients who want to stop exchanging models with others and leave the system. The other is the clients who have been inactive for a long period, or with a very low reputation. The system will kick them out by adding them to the leaving client set.

The BDFL system should also be able to maintain a correct DFL network topology experiencing client failures. The `maintenance` protocol requires every client to send neighbors a heartbeat message periodically, to filter out inactive clients.

Clients join BDFL to collaboratively train ML models. A new client joins the

BDFL system by calling the `join_network` function. In this function, in addition to the **Join** process to the network topology, as previously described, the client also needs to register on the BDFL blockchain to participate in future model exchanges. To do this, the new client, denoted as u, sends a join query to auditors. Auditors will record the client's information u_{id} and assign a default reputation rep_u to it. The auditor committee will pack the client information update to the blockchain periodically. Once the update message which includes the new client u is confirmed on the blockchain, u is then considered to have successfully registered within the BDFL system, and can start model training. Currently, u has no model, and has to initialize the training process by first gathering models from its neighbors. u has to verify the correctness of the model with the help of auditors before aggregating them. After getting verification results from auditors, u only chooses to use the correct models from the neighboring clients, and discard the others. u then locally generates an aggregated model as its own model. And later, it will keep gathering model updates from its neighbors and continue updating its local model to improve accuracy in the future. It will also exchange its local training model with its neighbors to contribute to the whole system and get profits.

Clients exchange local models with neighbors periodically, and the models will be evaluated by auditors which will affect their reputations. Thus, client reputations are updated by auditors during model verification. If a malicious model update is detected and verified by auditors, the corresponding client who provides this model will be punished with a low reputation. Clients with very low reputations will be removed from the BDFL system forever. To achieve this, auditors check their local client table

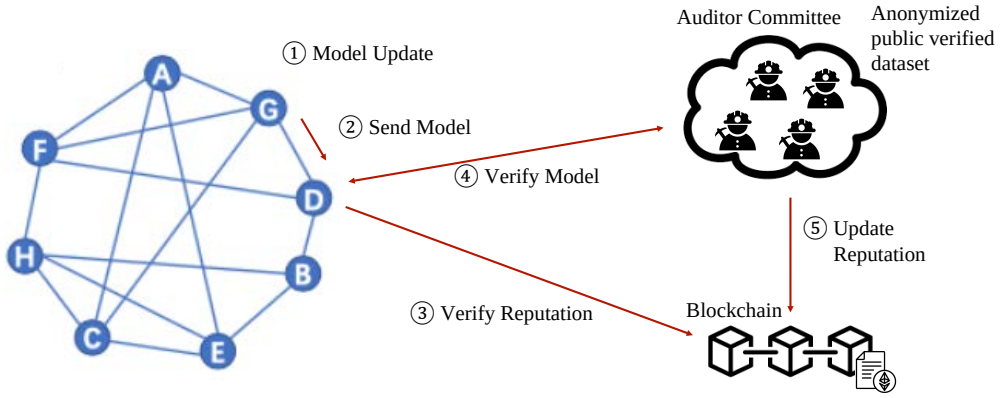


Figure 5.2: Protocol overview of BDFL.

periodically to filter out the clients with low reputations, and add them to the leaving client set. After this client table is confirmed in the blockchain, those clients are removed from the system successfully.

5.4.3 Model Exchange

In the model exchange, each client exchanges its local update with neighbors to do aggregation. However, clients might be not willing to send their models directly in concern that others might learn about their dataset information. It has long been established that gradients often leak sensitive information about clients' local datasets [15,18], and therefore, it is necessary for clients to hide their model updates to keep privacy. To prevent information leakage attacks, clients use differential privacy (DP) to hide their updates during model exchange and verification by adding noise sampled from a normal distribution [37]. We follow the concept of (ϵ, δ) differential privacy as being applied in many previous works [115, 129]. (ϵ, δ)-DP provides a strong criterion for privacy-preserving of distributed data processing systems. Thus, each client constructs a noisy

model update by adding some noise to the gradient such that $\omega^t = \zeta^t + \omega^t_{original}$.

Another challenge of performing decentralized model exchanges is that there is no central server to evaluate the quality of models from different clients. Thus, our Model Exchange Protocol is designed to validate the model update to mitigate the impact of malicious models. Fig. 5.2 illustrates the overview of the Model Exchange Protocol. It runs on both the clients and the auditors. Every client prepares its model locally, and exchanges the model update with neighbors once they finish training. BDFL uses asynchronous communication and allows each client to use a different communication and training time period in a round. Assume an honest client u has a local model ω^t_u in its current round t. And it has three neighbors v, w and x to exchange models with. In the neighbor set of u, in addition to the IP addresses and coordinates, it also stores the fingerprint f of the most recent model updates received from each neighbor, computed by hashing the model updates by a public hash function. When a client successfully prepares a model update to send, its neighbors would check the fingerprint f to avoid repetitive updates. This approach effectively mitigates unnecessary traffic, thereby reducing the frequency of exchanging duplicate models.

At client u, after gathering model exchanges ω^t_v, ω^t_w, and ω^t_x from all its neighbors, u first checks their reputations by reading the blockchain, and rejects the model updates from neighbors with low reputation. Let's say client x has a low reputation. u will directly reject the model update ω^t_x, and only continue to verify the correctness of the remaining model updates for aggregation. Now u needs to verify model updates ω^t_v and ω^t_w by sending verification requests $request\_verify(u, v, \omega^t_v)$ and

$request\_verify(u, w, \omega_w^t)$ to auditors. Auditors use the public validation data to verify the model updates. The smart contract maintains a history of model verification results, which include an average model accuracy μ_{t-1} and standard deviation σ_{t-1} of the latest 20 epochs. If the computed accuracy of the received model update, e.g. ω_w^t, is less than $\mu_{t-1} - 2\sigma_{t-1}$, this model update is considered to be malicious. Auditors will reply with the verification results to u, and punish w of the malicious update by decreasing its reputation. For models that pass the verification, auditors will increase the reputations of the corresponding clients, and the smart contract needs to update the average model accuracy μ_t and standard deviation σ_t as well. After receiving verification results from auditors, u finds that ω_v^t is the only valid model update. u will discard other model updates and later aggregate ω_v^t to its local model ω_u^t to train a new local model ω_u^{t+1}.

In a more complex scenario, client u discovers that it has received multiple valid model updates from various neighbors with different reputations after conducting model verification. u always tends to believe clients with high reputations will provide model updates of high quality, while low-reputation clients might provide low-quality models. Even if they are all valid updates, u still wants to limit the impact of low-quality models and amplify the impact of high-quality models in the aggregation. Thus, unlike the assumption that all model updates contribute equally to model aggregation, we let high-reputation clients have higher impacts. u defines a set of confidence parameters c for each model update, assigning higher c values to clients with better reputations. The

models from u's neighbors are then aggregated as follows:

$$\omega_u^{t+1} = \frac{\sum_{j \in N \cup \{u\}} c^j \omega_j^t}{\sum_{j \in N \cup \{u\}} c^j}$$

, with N being the neighbor set of client u. The model aggregation will be computed once every round and the models from each neighbor are always the most updated ones. In this way, clients with low reputations will have less impact on other clients.

5.4.4 Model Verification

In FL, every network provides a centralized validation dataset to validate the global model. The idea of using an anonymized public validation dataset for FL model validation is well adopted in the existing research [28, 116], and also used for detecting model poisoning attack [42]. We follow this assumption and adopt this idea for BDFL model update verification. We assume auditors have a public validation dataset D contributed by their clients and a global model ω_0. Before starting training, auditors collect training dataset samples from clients. We adopt ϵ-differential privacy [36] for protecting the data privacy of clients. After gathering all those anonymous datasets, auditors pre-evaluate their quality by comparing the accuracy of ω_0 on the public dataset D and each fetch dataset D_i. If the accuracy of D_i is much lower than the training result on D, auditors will reject this dataset D_i. Auditors then integrate the satisfied datasets into their local validation dataset to compose a new validation dataset D. During the training process, clients continue collecting local data, and there might be new clients

joining the system. Thus, auditors will gather clients' datasets periodically to update their validation datasets D.

When receiving a model verification request, auditors first check the fingerprint f. Then they can directly reply to the client with the previous verification result. Otherwise, auditors compute the accuracy by executing the model ω on the public validation dataset D. The smart contract aSC collects verification results from all auditors. It disregards those results that significantly deviate from the majority of the results, and considers auditors providing such results as malicious. aSC computes the average of the remaining accuracy results A_t^{ω}, and compares with the previously stored average model accuracy μ_{t-1} and standard deviation σ_{t-1} of the latest 20 epochs. If $A_t^{\omega} < \mu_{t-1} - 2\sigma_{t-1}$, ω will be judged as a malicious model update.

5.4.5 Reputation Management

Clients joining the system for the first time will be assigned an initial reputation *rep*. All the reputations are updated by auditors during each model verification. Let's say a client submits a model verification request on model update ω_u from its neighbor u. Auditors validate ω_u with public test data [101] and get the corresponding accuracy values. aSC filters out those outlier results and uses the average of the remaining results as the accuracy value $A_t^{\omega_u}$ of model update ω_u. Note that aSC maintains an accuracy history of previous model updates, μ_{t-1} and σ_{t-1}, which are average model accuracy and standard deviation of the latest 20 epochs respectively. If $A_t^{\omega_u} < \mu_{t-1} - 2\sigma_{t-1}$, auditors will consider this client u to be malicious, and punish u with a decrease in its

reputation from rep_u to $\frac{rep_u}{2}$. For honest clients, their model update accuracy might not always increase due to the imperfect of their dataset. The system tolerates this small accuracy decrease and will not lower their reputations as long as their model accuracy is larger than $\mu_{t-1} - 2\sigma_{t-1}$. Their reputations still keep the same in this round. Other honest clients can gain a reputation increase Δrep according to their model accuracy, which is computed as $\frac{A-\mu_{t-1}}{\sigma_{t-1}} \cdot 0.01$. For example, if the model accuracy of ω_u is $A_t^{\omega_u} = \mu_{t-1} + 2\sigma_{t-1}$, the reputation of the corresponding client u will increase by 0.02.

If a malicious model update is detected and verified by auditors, the corresponding client will be punished with a decrease in its reputation by 50%. However, a low reputation could result in its model update being declined by its neighbors in future model exchanges. Therefore, if such a client wants to continue participating in the model exchange and gain its reputation back, it must let auditors pre-verify its model updates prior to the model exchange period. If auditors validate this model update, the client will exchange its model along with the proof. Its neighbors can later confirm the correctness of this model update using the provided proof, thereby eliminating the need for another round of auditor verification.

Note that there is always a latency to post the new reputation (client table) to the blockchain. So in order to get the latest reputation information, clients can query auditors first. After some time, clients can verify the correctness of the query result by comparing it with the reputation confirmed in the blockchain. If clients detect that the reputation got from auditors is different from that in the blockchain, they can

submit disputes to the blockchain for compensation, and the malicious auditors will get punishment and lose all their collateral.

5.4.6 Incentives Mechanism

The involvement of the blockchain always triggers some fees, such as running smart contracts and submitting transactions to the blockchains. Thus, auditors require some incentives in order to do model verification and reputation management. In BDFL, clients are the devices that want to train a better model using model updates from other clients in the system. If they directly aggregate models without verification, potential malicious models from attackers could lead to poor performance on their training models. Thus, we assume clients are willing to pay auditors for verification, and honest clients who provide them with a high-quality model update. Clients can also gain rewards by providing honest model updates to other clients. Besides, there might be some services that want to use the final model directly without participating in the training process. Those services could request the model via the smart contract aSC by making a payment. aSC will then distribute the payment fee to auditors and clients according to their contribution.

An incentive mechanism is involved via the smart contract aSC to motivate participating clients and auditors to be honest and report the misbehavior. Any device with read-and-write access to the blockchain is allowed to become an auditor by registering with the auditor smart contract (aSC) and providing collateral. They are required to verify model updates, and distribute incentives to honest clients while impos-

ing punishment on misbehaving clients through the aSC. For each verification request, aSC gathers verification results from all auditors, and determines who provide correct verification results and record who always submit malicious results. If the number of incorrect verification results submitted by an auditor exceeds a predetermined threshold, this auditor will be judged as malicious and subject to penalties, such as losing all its collateral. It will then be removed from the auditor list in the smart contract. On the contrary, honest auditors who provide correct verification results will receive some rewards.

With the verification result of a model update from a client, the smart contract aSC will determine if this client is honest and should get some rewards. For example, currently aSC has a verification result $A_t^{\omega_u}$ on the model update ω_u provided by the client u. aSC finds that it is a high-quality update with $A_t^{\omega_u}$ larger than the average model accuracy of the latest 20 epochs μ_{t-1}, and u should be rewarded with $c(A_t^{\omega_u} - \mu_{t-1})$.

In model verification, when receiving a model update with high accuracy, auditors could assume this model is good enough to be a global model that reflects the features of data from all clients correctly. And auditors can choose to maintain this model ω locally and send a digest $h(\omega)$ to aSC. aSC will record which auditors maintain a local copy of the global model, and the corresponding digest. This is to prevent auditors from manipulating a malicious global model. If a service requests a model from aSC with a required fee, aSC will expose the auditor list who maintain the global model to the service. The service could randomly select an auditor to fetch the model and verify its correctness by querying aSC. aSC then distribute the fee collected from the service to

both the auditor who provided the model and to all clients. The distribution to clients is based on their respective reputations, which reflect their individual contributions to the model.

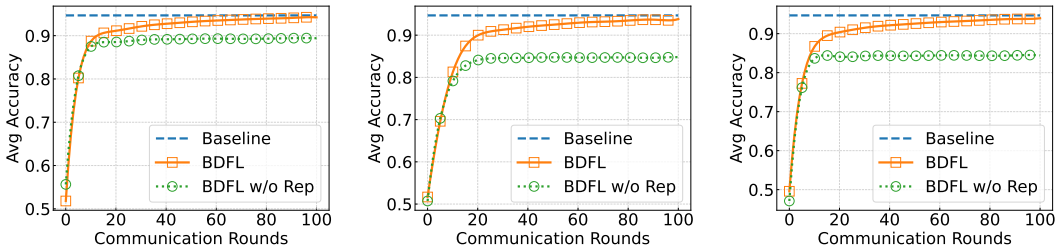
5.5 Performance Evaluation

5.5.1 Methodology

We use a desktop machine with an NVIDIA GeForce RTX3060 Ti as the platform for our experiment. We build a 100-client network, use real data training, and simulate reputation management and incentive distribution. The purpose of this experiment is to test the performance of effectiveness of the reputation mechanism and robustness in a simulated environment. We choose MNIST [33] and CIFAR-10 [70] image classification as the task of BDFL. We use MultiLayer Perceptron networks (MLP) and Convolutional Neural Networks (CNN) respectively as the machine learning models.

5.5.2 Evaluation Results

Robustness. We compare BDFL with the version without the reputation mechanism on two different datasets. The average accuracy of model updates verified by auditors is shown in Fig. 5.3 and Fig. 5.4 respectively. The Baseline shows the final model accuracy with no malicious clients in the system. In both two datasets, BDFL always demonstrates better accuracy under different proportions of malicious clients. The accuracy increase compared to the baseline without the reputation management

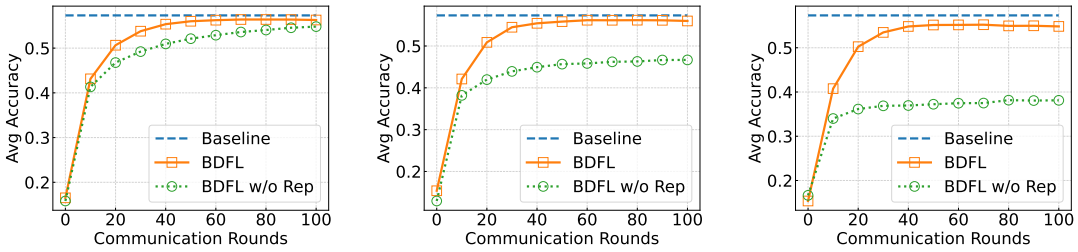


(a) 10% Malicious Clients (b) 20% Malicious Clients (c) 30% Malicious Clients

Figure 5.3: MNIST, Average Accuracy vs. Communication Rounds.

mechanism is more obvious with a higher proportion of malicious clients. Malicious clients can affect the average model accuracy by model poisoning attacks without verification. Honest clients will use those malicious models for aggregation directly, and thus harm the overall performance of the system. The more malicious clients in the system, the worse the average model accuracy. However, with the reputation management mechanism, malicious clients providing bad model updates can be detected by auditors and filtered out of the system. Malicious model updates will not be accepted by any client, and thus, even if there are 30% malicious clients in the system, The average accuracy of BDFL with reputation mechanism only has a degradation of 0.73% compared to baseline and BDFL without reputation mechanism has a degradation of 10.28% for MNIST classification as shown in Fig. 5.3(c). While in CIFAR-10 task, the average accuracy of BDFL with reputation mechanism has a degradation of 2.49% and BDFL without reputation mechanism has a degradation of 19.20% as shown in Fig. 5.4(c).

Reputation value evaluation. We monitor the reputation value changes for both honest and malicious clients in the MNIST dataset. The initial reputation of each client is set to be 0.5. The reputation value will be dynamically updated according

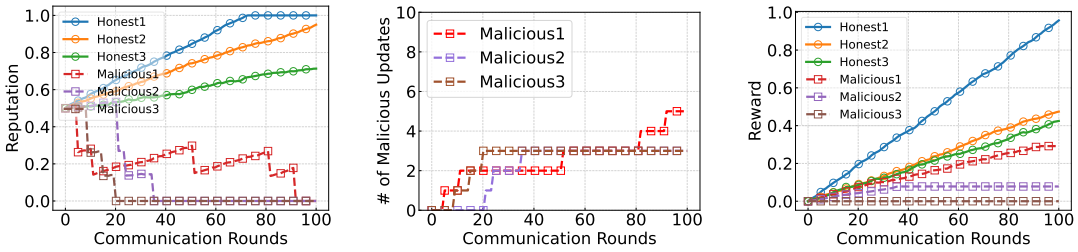


(a) 10% Malicious Clients (b) 20% Malicious Clients (c) 30% Malicious Clients

Figure 5.4: CIFAR-10, Average Accuracy vs. Communication Rounds

to their behaviors. The percentage of malicious clients in this set of experiments is set to 20%. Fig. 5.5(a) shows the reputation changes for 3 randomly chosen honest clients and 3 randomly chosen malicious clients. We find that honest clients always gradually achieve a high reputation value, even if they might not always be able to provide model updates of high accuracy due to the imperfect of their datasets. On the other hand, the reputation values of malicious clients will decrease to 0 in 100 rounds eventually, and they will be excluded from the system by auditors once extremely low reputations are detected. For malicious Client 1, even though it does not misbehave all the time, it can still be detected and penalized with a low reputation. Fig. 5.5(b) shows the total number of malicious model updates by malicious clients. After a short while, all these three malicious devices are detected and not able to perform poisoning attacks.

Incentive mechanism evaluation. We also monitor the accumulative incentives for the same 3 honest clients and 3 malicious clients in the MNIST dataset as in the previous experiment. As shown in Fig. 5.5(c), honest clients can always get rewards even if sometimes they are not able to provide good model updates. The honest Client 1 who always provides high-quality updates can gain more rewards. On the other hand,



(a) Reputation

(b) Malicious Updates

(c) Relative Accumulative Reward

Figure 5.5: MNIST, 20% Malicious clients. We sample 3 honest clients and 3 malicious clients and plot their reputation, accumulative reward (relative to 'Honest1') and the number of successful malicious updates in the training period. Client 'Malicious1' performs attacks on rounds 4,10,50,80,90. Client 'Malicious2' performs attacks on rounds 8,14,19,34,52,54,71,77,81,90,95. Client 'Malicious3' performs attacks on rounds 20,23,34,58,61,83,95

malicious clients might earn some rewards at the beginning, however, after a short period, they will be detected and cannot gain any further profit. The malicious Client 1 behaves correctly sometimes and it is hard to detect. But after round 95, it acquires an extremely low reputation which makes it expelled from the system, and cannot gain rewards.

5.6 Related Work

System	Asynchronous	Global model	Dynamic	Incentive	Privacy	Security
BLADE-FL [76]	No	Yes	No	No	Yes	Yes
Biscotti [115]	No	Yes	Yes	Yes	Yes	Yes
BAFFLE [108]	No	Yes	No	Yes	Not discussed	Not discussed
VFChain [101]	No	Yes	Yes	Yes	Yes	Yes
BAFL [43]	Yes	Yes	Yes	Yes	No	Yes
BDFL (this work)	Yes	No	Yes	Yes	Yes	Yes

Table 5.2: List of Blockchain-based Federated Learning System.

Federated Learning with a centralized aggregator reveals a single point of

failure and is vulnerable to malicious clients and false data. Thus, more decentralized setups have been proposed to address these limitations of centralized architecture, such as Blockchain-based Federated Learning. We review the existing state-of-the-art blockchain-based federated learning systems and summarize them in Table. 5.2.

BLADE-FL [76] proposed a blockchain-assisted decentralized FL, in which each client broadcasts the trained model to other clients, aggregates its own model with received ones, and then competes to generate a block before its local training of the next round. However, in FL, clients might not have enough computing capability to do both training and mining. Biscotti [115] focused on the security and privacy issue between peering clients. Instead of doing training and mining at the same time, clients' roles (such as verifier and aggregator) are selected randomly each round. However, it still requires all clients to agree on a global model at the end of each round. BAFFLE [108] leverages smart contracts to maintain the global model copy and the associated computational state of the users. The machine learning model weight vector is partitioned into numerous chunks to be stored in the smart contracts. However, with the dramatically growing size of the ML models, chunking might become extremely challenging. VFChain [101] utilized blockchain to verify and audit the correctness of the training process for federated learning. Instead of storing all the model updates, it only records the related verifiable information of models in the blockchain for audit in the future. However, same as in previous works, it requires synchronization between all clients. BAFL [43] introduced an asynchronous FL framework that uses a blockchain for model aggregation. However, every client needs to upload its local model to the blockchain,

which is even more inefficient. In traditional distributed machine learning, there are also works [100, 143, 144] concerning incentives for clients.

Compared to existing work, BDFL is the first solution for asynchronous blockchain-based fully decentralized FL, in which clients can join and leave the system dynamically and exchange models in a P2P manner without the need to maintain a global model in each round.

5.7 Conclusion

We present BDFL, a blockchain-based fully decentralized federated learning system that enables clients to exchange models in a P2P manner with model verification with high learning accuracy and system robustness. We design an incentive mechanism to encourage clients to participate in the model exchange, and a reputation model to evaluate the trustworthiness of each client to avoid malicious model updates from attackers. The evaluation results via simulations show that BDFL achieves fast model convergence and high accuracy on real datasets even if there exist 30% malicious clients in the system.

Chapter 6

Conclusion

In this thesis, we first improve the scalability of blockchain by presenting the design of a scalable and decentralized routing solution called WebFlow for large and dynamic PCNs. Our evaluation shows that WebFlow significantly outperforms existing solutions in per-node cost efficiency, resource utilization, and success rate. Then, we take a step further and propose APCN that supports shared funds among all the payment channels of a node with the help of Trusted Execution Environment. Results show that APCN achieves significantly higher success rates of multi-hop payments with lower average hops and latency, compared to existing PCNs. To achieve interoperability in the blockchain, we design XHub to support multi-hop paths across multiple blockchains, which achieves service availability, transaction atomicity, and auditability. Lastly, we explore the potential to integrate blockchain with real-world applications, and present BDFL, a blockchain-based fully decentralized federated learning system that enables model verification with high learning accuracy and system robustness.

Bibliography

[1] Bitcoin testnet. `https://tbtc.bitaps.com/`.

[2] Lightning network daemon. `https://github.com/lightningnetwork/lnd`.

[3] Networkx. `http://raiden.network/`.

[4] Visa fact sheet. https://usa.visa.com/dam/VCOM/global/about-visa/documents/visa-fact-sheet-july-2019.pdf, 2019.

[5] Poly network. `https://poly.network/`, 2020.

[6] Raiden network. http://raiden.network/, 2020.

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

[8] Coinmarketcap. `https://coinmarketcap.com/`, 2021.

[9] Sepolia testnet. `https://github.com/eth-clients/sepolia`, 2021.

[10] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov,

Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of ACM CCS*, 2016.

[11] Diego F Aranha. Relic is an efficient library for cryptography. *http://code. google. com/p/relic-toolkit/*, 2020.

[12] Md Armanuzzaman and Ziming Zhao. Byotee: Towards building your own trusted execution environments using fpga. *arXiv preprint arXiv:2203.04214*, 2022.

[13] Frederik Armknecht, Ghassan O Karame, Avikarsha Mandal, Franck Youssef, and Erik Zenner. Ripple: Overview and outlook. In *Proceedings of Springer TRUST*, 2015.

[14] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains*, 2014.

[15] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of ACM CCS*, 2020.

[16] Iddo Bentov, Yan Ji, Fan Zhang, Lorenz Breidenbach, Philip Daian, and Ari Juels. Tesseract: Real-time cryptocurrency exchange using trusted hardware. In *Proceedings of ACM CCS*, 2019.

[17] Battista Biggio, B Nelson, P Laskov, et al. Poisoning attacks against support vector machines. In *Proceedings of ICML*, 2012.

[18] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of ACM CCS*, 2017.

[19] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A Kroll, and Edward W Felten. Mixcoin: Anonymity for bitcoin with accountable mixes. In *Financial Cryptography and Data Security: 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers 18*, pages 486–504. Springer, 2014.

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

[21] Prosenjit Bose and Pat Morin. Online routing in triangulations. In *Proceedings of Springer ISAAC*, 1999.

[22] Prosenjut Bose and Pat Morin. Online routing in triangulations. 2004.

[23] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure:{SGX} cache attacks are practical. In *Proceedings of USENIX WOOT*, 2017.

[24] Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. Flyclient: Super-

light clients for cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 928–946. IEEE, 2020.

[25] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings IEEE Symposium on Foundations of Computer Science*, 2001.

[26] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM TOCS*, 2002.

[27] Sarthak Chakraborty and Sandip Chakraborty. Proof of federated training: accountable cross-network model training and inference. In *IEEE ICBC)*. IEEE, 2022.

[28] Olivia Choudhury, Aris Gkoulalas-Divanis, Theodoros Salonidis, Issa Sylla, Yoonyoung Park, Grace Hsu, and Amar Das. Anonymizing data for privacy-preserving federated learning. *arXiv preprint arXiv:2002.09096*, 2020.

[29] Coinbase. Coinbase-buy & sell bitcoin, ethereum, and more with trust. `https://www.coinbase.com/`, 2021.

[30] CoinMarketCap. Cryptocurrencies. `https://coinmarketcap.com/`, 2022.

[31] The Bitcoin community. M-of-n multisig, multisig output. `https://developer.bitcoin.org/devguide/transactions.html`, 2020.

[32] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed

Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *Proceedings of Springer FC*, 2016.

[33] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 2012.

[34] NetworkX Developers. https://networkx.github.io/documentation/networkx-1.9.1, 2014.

[35] Claudia Diaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. In *Proceedings of PET*, 2002.

[36] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT*. Springer, 2006.

[37] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 2014.

[38] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment hubs over cryptocurrencies. In *Proceedings of IEEE SP*, 2019.

[39] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. General state channel networks. In *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[40] Lisa Eckey, Sebastian Faust, Kristina Hostáková, and Stefanie Roos. Splitting

payments locally while routing interdimensionally. *IACR Cryptol. ePrint Arch.*, 2020.

[41] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *Proceedings of USENIX NSDI*, 2016.

[42] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local model poisoning attacks to byzantine-robust federated learning. In *Proceedings of USENIX Security*, 2020.

[43] Lei Feng, Yiqi Zhao, Shaoyong Guo, Xuesong Qiu, Wenjing Li, and Peng Yu. Bafl: A blockchain-based asynchronous federated learning framework. *IEEE Transactions on Computers*, 2021.

[44] Steven Fortune. Voronoi diagrams and delaunay triangulations. *Computing in Euclidean geometry*, pages 225–265, 1995.

[45] Ethereum Foundation. Ethereum project. http://www.ethereum.org/, 2020.

[46] Stellar Development Foundation. Stellar website. http://www.stellar.org/, 2020.

[47] Ryan Fugger. Money as ious in social trust networks and a proposal for a decentralized currency network protocol. Technical report, 2004.

[48] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. The limitations of federated learning in sybil settings. In *RAID*, 2020.

[49] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert

Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of ACM CCS*, 2016.

[50] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of ACM SOSP*, 2017.

[51] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 473–489, 2017.

[52] Huaqun Guo and Xingjie Yu. A survey on blockchain technology and its security. *Blockchain: research and applications*, 3(2):100067, 2022.

[53] Torben Hagerup, Kurt Mehlhorn, and J. Ian Munro. Maintaining discrete probability distributions optimally. In *Proceedings of the 20th International Colloquium on Automata, Languages and Programming*, volume 700 of *Lecture Notes in Computer Science*, pages 253–264, Berlin, 1993. Springer-Verlag.

[54] Lie He, An Bian, and Martin Jaggi. Cola: Decentralized linear learning. *Advances in NIPS*, 2018.

[55] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *Network and distributed system security symposium*, 2017.

[56] Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of ACM PODC*, 2018.

[57] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of ACM CCS*, 2017.

[58] Yifan Hua, Kevin Miller, Andrea L Bertozzi, Chen Qian, and Bao Wang. Efficient and reliable overlay networks for decentralized federated learning. *SIAM Journal on Applied Mathematics*, 82(4):1558–1586, 2022.

[59] Zhichao Hua, Jinyu Gu, Yubin Xia, Haibo Chen, Binyu Zang, and Haibing Guan. vtz: Virtualizing {ARM} trustzone. In *Proceedings of USENIX Security*, 2017.

[60] Intel. Intel 64 and ia-32 architectures software developer manuals. `https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html`.

[61] Intel. Product change notification. `https://qdms.intel.com/dm/i.aspx/5A160770-FC47-47A0-BF8A-062540456F0A/PCN114074-00.pdf`, 2015.

[62] Code Sample Intel. Intel software guard extensions remote attestation end-to-end example, 2018.

[63] R Intel. Intel(r) software guard extensions for linux os. `https://github.com/intel/linux-sgx`.

[64] R Intel. Software guard extensions programming reference. *Intel Corporation*, 2014.

[65] Jiawen Kang, Dongdong Ye, Jiangtian Nie, Jiang Xiao, Xianjun Deng, Siming Wang, Zehui Xiong, Rong Yu, and Dusit Niyato. Blockchain-based federated learning for industrial metaverses: Incentive scheme with optimal aoi. In *IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2022.

[66] David Kaplan, Jeremy Powell, and Tom Woller. Amd memory encryption. *White paper*, 2016.

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

[68] Rami Khalil and Arthur Gervais. Revive: Rebalancing off-blockchain payment networks. In *Proceedings of ACM CCS*, 2017.

[69] Neal Koblitz. Cm-curves with good cryptographic properties. In *Crypto*, volume 91, pages 279–287. Springer, 1991.

[70] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[71] Jae Kwon and Ethan Buchman. Cosmos whitepaper. *A Netw. Distrib. Ledgers*, 2019.

[72] Aurora Labs. Idex: A real-time and high-throughput ethereum smart contract exchange. `https://www.allcryptowhitepapers.com/wp-content/uploads/2018/05/IDEX.pdf`, 2019.

[73] Simon S Lam and Chen Qian. Geographic routing in d-dimensional spaces with guaranteed delivery and low stretch. In *Proceedings of ACM SIGMETRICS*, 2011.

[74] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Dawn Song, and Krste Asanović. Keystone: An open framework for architecting tees. *arXiv preprint arXiv:1907.10119*, 2019.

[75] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. Inferring fine-grained control flow inside {SGX} enclaves with branch shadowing. In *Proceedings of USENIX Security*, 2017.

[76] Jun Li, Yumeng Shao, Kang Wei, Ming Ding, Chuan Ma, Long Shi, Zhu Han, and H Vincent Poor. Blockchain assisted decentralized federated learning (blade-fl): Performance analysis and resource allocation. *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[77] Mingyu Li, Jinhao Zhu, Tianxu Zhang, Cheng Tan, Yubin Xia, Sebastian Angel, and Haibo Chen. Bringing decentralized search to decentralized services. In *Proceedings of USENIX OSDI*, 2021.

[78] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, 2020.

[79] Yunming Liao, Yang Xu, Hongli Xu, Lun Wang, and Chen Qian. Adaptive con-

figuration for heterogeneous participants in decentralized federated learning. In *Proceedings of IEEE INFOCOM*, 2023.

[80] Linaro. Open portable trusted execution environment. `https://www.op-tee.org/`, 2014.

[81] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter Pietzuch. Teechain: a secure payment network with asynchronous blockchain access. In *Proceedings of ACM SOSP*, 2019.

[82] Jianchun Liu, Hongli Xu, Lun Wang, Yang Xu, Chen Qian, Jinyang Huang, and He Huang. Adaptive asynchronous federated learning in resource-constrained edge computing. *IEEE TMC*, 2023.

[83] Chuan Ma, Jun Li, Ming Ding, Howard H Yang, Feng Shu, Tony QS Quek, and H Vincent Poor. On safeguarding privacy and security in the framework of federated learning. *IEEE network*, 2020.

[84] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Silentwhispers: Enforcing security and privacy in decentralized credit networks. In *Proceedings of USENIX NDSS*, 2017.

[85] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In *Proceedings of ACM CCS*, 2017.

[86] Vasilios Mavroudis, Karl Wüst, Aritra Dhar, Kari Kostiainen, and Srdjan Capkun.

Snappy: Fast on-chain payments with practical collaterals. In *Proceedings of USENIX NSDI*, 2020.

[87] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[88] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 2017.

[89] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology—CRYPTO'87: Proceedings 7*, pages 369–378. Springer, 1988.

[90] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning. In *Proceedings of Springer FC*, 2019.

[91] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. Cachezoom: How sgx amplifies the power of cache attacks. In *Proceedings of Springer CHES*, 2017.

[92] Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Kim Pecina. Privacy preserving payments in credit networks. In *Proceedings of USENIX NDSS*, 2015.

[93] Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, 1992.

[94] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2009.

[95] TS Eugene Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In *Proceedings of IEEE INFOCOM*, 2002.

[96] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. A survey of published attacks on intel sgx. *arXiv preprint arXiv:2006.13598*, 2020.

[97] M. Ogleari, Y. Yu, C. Qian, E. Miller, and J. Zhao. String Figure: A Scalable and Elastic Memory Network Architecture. In *Proceedings of IEEE HPCA*, 2019.

[98] Yasuhiro Ohara, Shinji Imahori, and Rodney Van Meter. Mara: Maximum alternative routing algorithm. In *Proceedings of IEEE INFOCOM*, 2009.

[99] Michael Oved and Don Mosites. Airswap whitepaper. `https://www.airswap.io/whitepaper.htm`.

[100] Jinlong Pang, Jieling Yu, Ruiting Zhou, and John CS Lui. An incentive auction for heterogeneous client selection in federated learning. *IEEE Transactions on Mobile Computing*, 2022.

[101] Zhe Peng, Jianliang Xu, Xiaowen Chu, Shang Gao, Yuan Yao, Rong Gu, and Yuzhe Tang. Vfchain: Enabling verifiable and auditable federated learning via blockchain systems. *IEEE TNSE*, 2021.

[102] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[103] Chen Qian and Simon Lam. ROME: Routing On Metropolitan-scale Ethernet . In *Proceedings of IEEE ICNP*, 2012.

[104] Chen Qian and Simon S. Lam. Greedy Distance Vector Routing. In *Proceedings of IEEE ICDCS*, 2011.

[105] Chen Qian and Simon S Lam. Greedy routing by network distance embedding. *IEEE/ACM ToN*, 2015.

[106] Xianrui Qin, Shimin Pan, Arash Mirzaei, Zhimei Sui, Oguzhan Ersoy, Amin Sakzad, Muhammed Esgin, Joseph K Liu, Jiangshan Yu, and Tsz Hon Yuen. Blindhub: Bitcoin-compatible privacy-preserving payment channel hubs supporting variable amounts. In *2023 IEEE Symposium on Security and Privacy (SP)*.

[107] Minghua Qu. Sec 2: Recommended elliptic curve domain parameters. *Certicom Res., Mississauga, ON, Canada, Tech. Rep. SEC2-Ver-0.6*, 1999.

[108] Paritosh Ramanan and Kiyoshi Nakayama. Baffle: Blockchain based aggregator free federated learning. In *IEEE international conference on blockchain (Blockchain)*, 2020.

[109] Stefanie Roos, Martin Beck, and Thorsten Strufe. Anonymous addresses for efficient and resilient routing in f2f overlays. In *Proceedings of IEEE INFOCOM*, 2016.

[110] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling

payments fast and private: Efficient decentralized routing for path-based transactions. In *Proceedings of USENIX NDSS*, 2017.

[111] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. *arXiv preprint arXiv:1806.01246*, 2018.

[112] Hex-Five Security. Multizone: The first trusted execution environment for risc-v. `https://hex-five.com/`, 2018.

[113] Jaebaek Seo, Byoungyoung Lee, Seong Min Kim, Ming-Wei Shih, Insik Shin, Dongsu Han, and Taesoo Kim. Sgx-shield: Enabling address space layout randomization for sgx programs. In *Proceedings of NDSS*, 2017.

[114] Andrei Serjantov and George Danezis. Towards an information theoretic metric for anonymity. In *Proceedings of PET*, 2002.

[115] Muhammad Shayan, Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. Biscotti: A blockchain system for private and secure federated learning. *proceedings of IEEE TPDS*, 2020.

[116] Micah J Sheller, Brandon Edwards, G Anthony Reina, Jason Martin, Sarthak Pati, Aikaterini Kotrotsou, Mikhail Milchenko, Weilin Xu, Daniel Marcus, Rivka R Colen, et al. Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data. *Scientific reports*, 2020.

[117] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. T-sgx: Eradicating controlled-channel attacks against enclave programs. In *Proceedings of NDSS*, 2017.

[118] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathleen Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Giulia Fanti, and Mohammad Alizadeh. High throughput cryptocurrency routing in payment channel networks. In *Proceedings of USENIX NSDI*, 2020.

[119] Tao Sun, Dongsheng Li, and Bao Wang. Decentralized federated averaging. *Proceedings of IEEE TPAMI*, 2022.

[120] Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. A2l: Anonymous atomic locks for scalability in payment channel hubs. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1834–1851. IEEE, 2021.

[121] Univ. Bordeaux The PARI Group. Pari/gp version 2.12.0. 2019.

[122] Stefan Thomas and Evan Schwartz. A protocol for interledger payments. *https://interledger.org/interledger.pdf*, 2015.

[123] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. In *Proceedings of USENIX Security*, 2018.

[124] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. Singular value decom-

position and principal component analysis. In *A practical approach to microarray data analysis*, pages 91–109. Springer, 2003.

[125] Jiaping Wang and Hao Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *Proceedings of USENIX NSDI*, 2019.

[126] Peng Wang, Hong Xu, Xin Jin, and Tao Wang. Flash: efficient dynamic routing for offchain networks. In *Proceedings of ACM CoNEXT*, 2019.

[127] Will Warren and Amir Bandeali. 0xproject whitepaper. `https://www.0x.org/pdfs/0x\_white\_paper.pdf`.

[128] Bernard M Waxman. Routing of multipoint connections. 1988.

[129] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE TIFS*, 2020.

[130] Herbert S Wilf. *Algorithms and complexity*. AK Peters/CRC Press, 2002.

[131] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[132] Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework. *White paper*, 2016.

[133] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkbridge: Trustless cross-chain bridges made practical. *arXiv preprint arXiv:2210.00264*, 2022.

[134] Han Xue, Qun Huang, and Yungang Bao. Epa-route: Routing payment channel network with high success rate and low payment fees. In *Proceedings of IEEE ICDCS*, 2021.

[135] Ye Yu and Chen Qian. Space shuffle: A scalable, flexible, and high-bandwidth data center network. In *Proceedings of IEEE ICNP*, 2014.

[136] Ye Yu and Chen Qian. Space shuffle: A scalable, flexible, and high-performance data center network. In *Proceedings of IEEE ICNP*, 2014.

[137] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of ACM CCS*, 2018.

[138] Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William Knottenbelt. Xclaim: Trustless, interoperable, cryptocurrency-backed assets. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 193–210. IEEE, 2019.

[139] Xiaoxue Zhang, Yifan Hua, and Chen Qian. Secure decentralized learning with blockchain. In *IEEE International Conference on Mobile Ad-Hoc and Smart Systems (MASS)*. IEEE, 2023.

[140] Xiaoxue Zhang and Chen Qian. Towards aggregated payment channel networks. In *IEEE International Conference on Network Protocols (ICNP)*. IEEE, 2022.

[141] Xiaoxue Zhang and Chen Qian. A cross-chain payment channel network. In *IEEE 31st International Conference on Network Protocols (ICNP)*. IEEE, 2023.

[142] Xiaoxue Zhang, Shouqian Shi, and Chen Qian. Low-overhead routing for offchain networks with high resource utilization. *Proceedings of International Symposium on Reliable Distributed Systems (SRDS)*, 2023.

[143] Ruiting Zhou, Jinlong Pang, Zhibo Wang, John CS Lui, and Zongpeng Li. A truthful procurement auction for incentivizing heterogeneous clients in federated learning. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 183–193. IEEE, 2021.

[144] Ruiting Zhou, Ne Wang, Yifeng Huang, Jinlong Pang, and Hao Chen. Dps: Dynamic pricing and scheduling for distributed machine learning jobs in edge-cloud networks. *IEEE Transactions on Mobile Computing*, 2022.

[145] Li Zhuang, Feng Zhou, Ben Y Zhao, and Antony Rowstron. Cashmere: Resilient anonymous routing. In *Proceedings of USENIX NSDI*, 2005.