# Toward Aggregated Payment Channel Networks

Xiaoxue Zhang, *Member, IEEE*, and Chen Qian, *Senior Member, IEEE, ACM*

***Abstract*— Payment channel networks (PCNs) have been designed and utilized to address the scalability challenge and throughput limitation of blockchains. It provides a high-throughput solution for blockchain-based payment systems. However, such "layer-2" blockchain solutions have their own problems: payment channels require a separate deposit for each channel of two users. Thus it significantly locks funds from users into particular channels without the flexibility of moving these funds across channels. In this paper, we proposed Aggregated Payment Channel Network (APCN), in which flexible funds are used as a per-user basis instead of a per-channel basis. To prevent users from misbehaving such as double-spending, APCN includes mechanisms that make use of hardware trusted execution environments (TEEs) to control funds, balances, and payments. The distributed routing protocol in APCN also addresses the congestion problem to further improve resource utilization. Our prototype implementation and simulation results show that APCN achieves significant improvements on transaction success ratio with low routing latency, compared to even the most advanced PCN routing.**

***Index Terms*— Blockchain, payment channel network, security, TEE.**

## I. Introduction

BLOCKCHAIN is a promising solution for decentralized digital ledgers, but *low throughput* remains a huge problem with growing numbers of users and transactions [2], [3]. For instance, Bitcoin can only support 10 transactions per second at peak in 2020 [4]. Payment channel networks (PCNs) [3] are a leading concept to provide a high-throughput solution for blockchains. In a PCN, two users can conduct transactions with each other through a bi-directional channel. The blockchain is only involved when the users open and close the channel [5]. Each user commits a certain fund at the opening of this channel. Then they can make any number of transactions that update the tentative distribution of the channel's funds as long as the remaining funds allow. These transactions only need to be signed by the two users, and do *not* need to be broadcast to the entire blockchain. Each user can establish channels with multiple other users. If a channel does not exist between two users, they can make a transaction via a multi-hop path, where any two consecutive users on the path share a channel. As shown in Fig. 1(a), if user $a$ wants to make a payment to another 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 $a-b$ and $b-c$ accordingly. The PCN is a promising solution to achieve the scalability of blockchains because most transactions can be achieved in an off-chain manner.

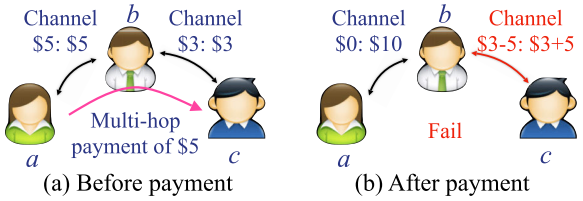



Fig. 1. A multi-hop payment in a PCN.

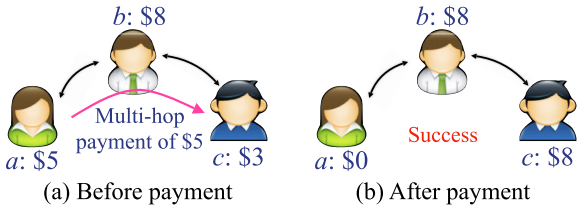



Fig. 2. A multi-hop payment in an APCN.

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 [6]. 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. 1(b), when $a$ pays $c$ \$5, the link from $b$ to $c$ only has balance of \$3 and hence cannot support this transaction. 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 in PCNs. Many recent studies focus on using routing protocols to improve resource utilization in PCNs, such as Spider [7] and Flash [8]. 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.

Manuscript received 20 December 2022; revised 28 February 2024; accepted 14 June 2024; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor D. Malone. This work was supported in part by the National Science Foundation under Grant 1750704, Grant 1932447, Grant 2114113, Grant 2322919, and Grant 2420632. The work of Chen Qian was supported in part by Army Research Office (ARO) under Grant W911NF-20-1-0253. The preliminary version of this paper [DOI: 10.1109/ICNP55882.2022.9940365] appeared at the Proceedings of IEEE ICNP, 2022. *(Corresponding author: Chen Qian.)*

Xiaoxue Zhang was with the Department of Computer Science and Engineering, University of California at Santa Cruz, Santa Cruz, CA 95064 USA. She is now with the Department of Computer Science and Engineering, University of Nevada Reno, Reno, NV 89557 USA (e-mail: xiaoxuez@unr.edu).

Chen Qian is with the Department of Computer Science and Engineering, University of California at Santa Cruz, Santa Cruz, CA 95064 USA (e-mail: cqian12@ucsc.edu).

Digital Object Identifier 10.1109/TNET.2024.3423000

Teechain [9] is a recent work to prevent parties from stealing funds, relying on trusted execution environments (TEEs). It allows funds to be moved in and out of the network and between payment channels dynamically using TEE. But such a method cannot improve the success ratio of transactions because it is unable to determine how much fund is sufficient in a channel before the transactions. None of these works allow sharing fund across different channels of a same user.

In this paper, 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. 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 [9] 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: When finding paths for concurrent payments and multiple payments use the same channel on their paths, There are two potential solutions in existing PCNs. 1) The two paths use the channel simultaneously, but a solution is needed when the channel capacity is not sufficient; 2) the channel is used in a first-come-first-serve manner. Then the second transaction needs to find another path. Compared to previous works in PCNs, our APCN solution can well support concurrency payments because multi-hop payments do not change the funds of intermediate nodes. The main challenge is that, 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.

In summary, this paper makes the following contributions:

- We propose APCN, a novel design of payment channel networks with shared funding that could improve success ratio of multi-hop payments, and avoid locked-in funds in channels as well.
- We design a routing protocol with congestion control for APCN that could lower the average processing time of the whole network with low per-node overhead while achieves high resource utilization.
- We simulate APCN based on real-world PCN topologies and transactions. The results show the claimed advantages of APCN compared to the state-of-the-art protocols.

The rest of this paper is organized as follows. The system overview and model are presented in Section II. We describe an overview in Section III and the detail design of the APCN and routing protocol in Section IV. Section VII presents the evaluation results of APCN. Section VIII describes the related work. Section IX concludes this work.

## II. Overview

### A. 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 $e_1 \dots e_k$ 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, $\psi_1$. Every node knows the links to its neighbors. When two nodes want to make a transaction, they can exchange their information via the Internet. However, they might not know the paths to connect them at the beginning.

**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 distributed greedy routing introduced in a recent work WebFlow [10] and extend it for APCN. Virtual coordinates-based distributed greedy routing can achieve a low per-note routing state and high routing success rate, since each node only knows and interacts with a small subset of other users, independent of the entire network size. It is a highly scalable and decentralized solution for payment channel networks. Other types of routing can also be used in APCN, but the current implementation has the best performance among other possible implementations.

### B. 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) [11]. 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, including Intel SGX [12], ARM TrustZone [13] and AMD SEV [11], with several others currently underway, such as KeyStone Enclave [14], Multizone [15] and OP-TEE [16]. Intel CPUs from the Skylake generation onwards support SGX [17], 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. **TEEs are widely available and many blockchain based applications have been using TEEs for other purposes such as Teechain** [9].

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

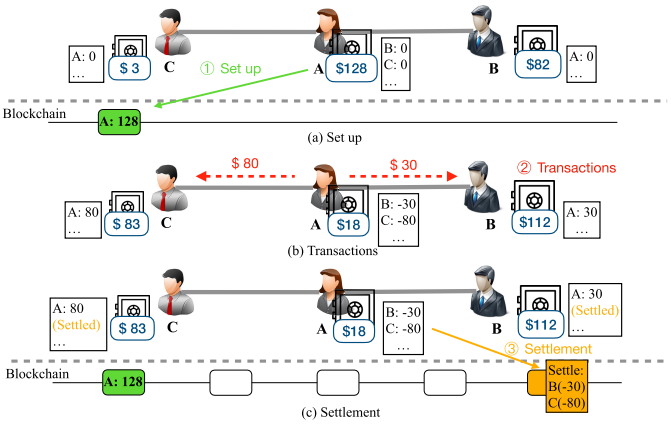



Fig. 3. APCN overview: APCN nodes operate TEEs to store and manage funds. Users construct payment channels between nodes to exchange funds directly, and execute multi-hop payments along concatenated payment channels.

Fig. 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 as defined in the settlement protocol in the TEE. After the ledger appears 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. The correctness of the execution of the protocol is guaranteed by the TEE. If there is any external adversary preventing TEEs from correct behavior, we treat it as TEE failure, and certain users need to restore and recover data before the TEE failure. The user's TEE only records its own channel and deposit information, independent of the entire network size. As shown

in Fig. 3, the TEE does not keep all the transaction history in a channel. Instead, it only updates the overall transaction value after each transaction happens in this channel. Thus, the system can achieve high scalability even with a large number of nodes and a large volume of transactions in the network.

### C. 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 [18]. To prevent information leakage from access patterns, existing oblivious RAM library can be adopted [19]. There are also existing timing and memory-access side-channel resistant libraries for sensitive data [9]. Shih et al. [20] presented a modified LLVM compiler dubbed T-SGX, which is effective against all known controlled-channel attacks. Lee et al. [21] proposed ZigZagger as a defence against their own branch shadowing attack. To defeat enclave specific attacks such as ROP attacks, Seo et al. [22] activated ASLR inside SGX enclaves to make exploitation more difficult. BYOTee [23] 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.

### D. 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. Overall, 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 – a user's perceived balance is their initial balance on the blockchain plus any payments received in the payment network, minus any payments made.

**Privacy:** For privacy, our goal is to hide values, and achieve anonymity of sender and receiver when making transactions. We use the term *value privacy*, *sender/receiver anonymity* respectively to refer to these three privacy goals. We say that the system can achieve value privacy if it is impossible for any adversary to know the total value of a transaction between two honest users. The system can achieve sender anonymity if the adversary cannot determine the original sender of a transaction. Similarly, receiver anonymity can be achieved if the adversary cannot determine the actual receiver.

**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.

We introduce another performance requirement that can further improve transaction success rates: efficient congestion control. When many transactions requests happen in a PCN at the same time, we call these requests as concurrent payment requests. Existing PCN systems such as Flash [8] and SlientWhispers [24] do not provide solutions to handle concurrent requests. When looking for paths for these concurrent payments, if multiple payments want to use the same channel based on routing, congestion on the channel occurs. Hence there are two ways to deal with this problem. 1) The two paths share this channel and each gets a lower capacity if the channel capacity is not sufficient. 2) The channel is used in a first-come-first-serve manner and the other transaction should use another path. Malavolta et al. [25] proposed two methods in PCN to handle concurrency which decrease transaction success rates and increase processing delays. APCN aims to support concurrent transactions to achieve high success rates.

### E. Analysis Methodology of This Work

From our observations 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 or anonymity of a routing algorithm. We will use extensive simulations with real network topologies to analyze the routing performance or anonymity.

## III. Design Overview of APCN

We provide an overview of transaction executions in APCN. Similar to traditional PCNs, two users can conduct transactions with each another via a bi-directional channel. They initiate a corresponding entry in their ledger at the opening of this channel. This entry will record all the transactions happening in this channel. Table I 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 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

TABLE I
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 $t$ |

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 [26]. 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 [9]. 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 [9].

## IV. Payment Protocol

This section describes the design of APCN protocols.

### A. Deposits Allocation

In order to join the system, users need to create a deposit in the ledger maintained by their TEEs, like making a deposit to the wallets in the blockchain. Each TEE has public/private key pairs for their wallet addresses, which are generated ahead securely inside the TEE with their private keys stored in TEE memory. The public key is used to make deposits to, which is safe to share and is what others will use to send funds to the users. In order to create a deposit for a user, they first need to obtain the wallet's public address first. They can simply query their TEE for the public key `k` and store it locally for future usage. 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. The user can only redeem the funds back from the TEE after deposit settlement on the blockchain.

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. After setting up payment channels, users can leverage channels and their locked deposits in TEEs to make any number of off-chain transactions.

### B. Using Payment Channels

To execute a payment $t$ along a channel, the sender $u$ calls `update_channel`, which specifies the amount $\omega$ 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.

Since a deposit is not associated with a single channel, participants will not suffer from *deposit lock-in* as in PCNs: when a large deposit is added to a channel but only a small fraction is spent, it leaves the remaining locked-in until the channel is settled. In APCN, all the channels can use the remaining deposit. Note that APCN still has deposit lock-in, and users cannot redeem the deposit back until they make a settlement on-chain. This deposit lock-in cannot be avoided since users need to leverage the deposit to make transactions in the system. Compared with traditional PCNs that lock-in funds can only be used in a certain channel and require lock-in funds for each channel, APCN is more flexible such that the lock-in funds can be used in all the channels. Thus, to support the same set of transactions with different users, APCN will require significantly fewer lock-in funds compared with PCNs. Besides, at any time, either party may shut down the channel using close_channel. Since the channel is stateless, the TEEs can terminate the channel off-chain by simply marking the channel $c$ as inactive as long as the channel is not in use. Different from channel settlement in PCNs, off-chain termination avoids writing a settlement transaction to the blockchain, which saves plenty of processing time.

### C. Congestion Control

To perform a multi-hop payment, the sender needs first to find a payment path to the recipient, based on the routing function. APCN routing is built upon the routing protocol introduced in WebFlow [10], which is a virtual coordinates based greedy routing. Each node has a set of coordinates and the routing protocol always choose the neighbor that is closest to the recipient in the coordinate space. If no neighbor is closer to the recipient than the current node, WebFlow will use pre-established paths to find a node that is closer to the destination. The sender invokes routing with the coordinates of the recipient. For each intermediate user received routing request, it needs to determine the next hop in a distributed manner. However, this routing protocol still has the problem of under-utilization and does not consider the end-to-end latency of the network.

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 [7]. Consider an example PCN shown in Fig 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 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.

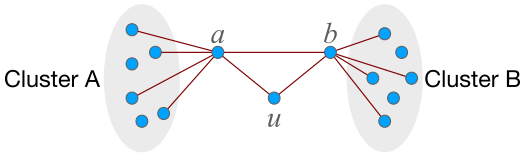



Fig. 4. Example illustrating the importance of congestion control in APCN.

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 $\Delta$, 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.

### D. 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 IV-B. 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. After receiving the proof, each neighbor could easily verify the existence of $u$'s ledger on the Blockchain. If the proof is correct, the neighbor nodes will mark the channel $u-v$ as `Settled` in their own ledgers. Here, we treat the Blockchain as a root-of-trust, and roll-back-attacks towards Blockchain is out of scope of this paper.

TABLE II
NOTATION

| Notation | Description |
|---|---|
| $U$ | User |
| $T$ | TEE service provider |
| $C$ | Committee |
| $k$ | TEE service provider's identifier, which is its public key |
| $L$ | Ledger's identifier in TEE |

### E. Data Recovery After TEE Failure

For users without TEEs on their devices, they rely on a service provider committee to manage fund deposits, ledgers and transactions. For each operation handled 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. However, for users with local TEE, their deposits and ledgers are only kept inside the TEEs. Since TEEs isolate data securely, if the local TEEs are compromised or fail, data stored within it can become inaccessible. Thus, in APCN, a robust backup and recovery mechanism is essential to maintain system integrity and ensure data resilience after TEE failures.

Upon a user joining the system and successfully creating a deposit, the corresponding TEE exports an encrypted copy of the deposit information, along with a timestamp and its signature, to the device outside the TEE. Every time when a new channel is created, the TEE needs to export the latest version of the ledger with all the channel information. Note that APCN strategically avoids backing up after every transaction due to the potential for high-frequency transaction activities, which could significantly strain the TEE's resources. Instead, backups are exported with the creation of new channels—a relatively less frequent event that balances the need for up-to-date data retention with the operational overhead on the TEE. After a TEE failure and subsequent restoration, the user first perform re-attestation to ensure the integrity and trustworthiness of the TEE. They import the previously saved encrypted ledger backup into the TEE. The TEE assesses the authenticity and correctness of this backup by checking the signature. Once validated, the TEE initiates the recovery of transaction histories for each channel by reaching out to the recorded neighbors in the backup.

## V. Protocol Details and Specification

In this section we provide the detail description and pseudo-code of our protocols introduced in the Sec. IV. We tabulate our notation in Table II. We denote the set of users as $U = \{U_1, \ldots, U_n\}$, in which users may goes offline and online some times. TEE service providers are denoted as $T = \{T_1, \ldots, T_m\}$, some of which belongs to APCN users, the others are from devices that are willing to provide their TEE to APCN system.

TABLE III
TRANSACTION FORMAT

| Field | Symbol | Description |
|---|---|---|
| To | $\tau_{to}$ | Recipient address |
| From | $\tau_f$ | Address of the last hop user |
| amount | $\tau_a$ | Transaction amount |
| Transaction index | $\tau_i$ | Monotonic counter for transaction index |

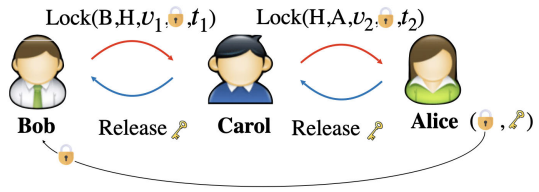



Fig. 5. Transaction from Bob to Alice using HTLC contract for atomicity.

TABLE IV
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` | $\omega$ | The version number of the latest transaction |

### A. Transaction Data Format

**Transactions.** All transactions among users are conducted via channels by TEE service providers, the format of which are shown in Table III. Each transaction $\tau$ includes the address of the transaction recipient $\tau_{to}$, the transaction amount $\tau_a$, the address of the last hop user $\tau_f$, and a monotonically increasing transaction index $\tau_i$. We note that $\tau_f$ here is not necessarily the sender. For multi-hop transactions, $\tau_f$ records the last hop where the transaction comes from. For each intermediate user received a transaction, it needs to replace the $\tau_f$ field to be the address of it, and relay the transaction to the next hop. The protocol has to guarantee atomicity, that is, either a multi-hop transaction is successful, or the fund goes back to the original sender. Malavota et al. [25] proposed a secure and privacy-preserving protocol for multi-hop payments. It requires the Hash Time Lock Contract (HTLC) supported in cryptocurrency, as the example shown in Figure 5. In this transaction, Alice's TEE first sets up the transaction by creating a secret key and sending the lock $=$ $H$(key) to Bob. Then, the commitment phase starts with Bob. He first sets on hold $v_1$, and then the intermediate user sets on hold the received amount minus his own fee. After the intermediate user sets $v_2$ on hold with Alice, Alice knows that the corresponding transaction amount is on hold at each transaction party and she can start the releasing phase. For that, she reveals the key to the intermediate user allowing him to fulfill the HTLC contract and settle the new funds. The key is then passed back to the sender Bob for his settlement.

**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 IV. 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 IV. The amount of the channel can be negative, which is the amount user $u$ owes $c_N$.

### B. Users With and Without TEE

Here we describe two categories of users separately, the user with local TEE, and the user without local TEE. For the first case, a client trusts its TEE, and uses single TEE to hold the balance. For the latter one, a client chooses a set of remote TEE service providers $T = \{T_1, \ldots, T_m\}$ as a committee. Algorithm 1 shows the protocol executed by each node and TEE service provider. To construct a new deposit $d$, a user with local TEE initiates the process by invoking `new_deposit_withTEE` (Alg. 1, line 1) and preparing a deposit transaction $t$ to the TEE. This transaction $t$ includes the amount of deposit $d$ and the TEE public key $k$ that $t$ sends funds to, which serves as the TEE's wallet address. The transaction $t$ needs to be signed by the user using their private key, which helps to verify the user's ownership of the deposit and secures the transaction integrity by encryption. This transaction can then be broadcast to the blockchain and placed on-chain. Broadcasting the transaction to the network could be performed through various online tools [27] or using a locally running blockchain node. Once the user has paid funding deposits into those addresses and the transaction $t$ is confirmed on the blockchain, they tell their TEEs about them, including the amounts paid, the transaction hashes and the transaction ids. TEE verifies that $t$ sends enough funds $d$ to the correct address $k$ by exploring the blockchain with the transaction id and its wallet address $k$. TEE can also verify the transaction signature using the user's public key, ensuring the transaction integrity. TEE can then begin creating a ledger with the corresponding deposit $d$ (line 6).

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, $k_1 \ldots k_m$, 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 closed 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

**Algorithm 1** 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:  end if
60:  end if
61:  return FALSE
62: def settle_deposit(d_i):
63:  for all channels c in U's ledger do
64:  close_channel(c_i)
65:  end for
66:  t ← construct_tx(d_i, k)
67:  return t
```

be set up, it must be approved by 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). At the network layer, the user who initiates the channel creation process needs to announce the local host/port and the remote IP address and port number of the remote user. Before this can be called, the remote user will also need to execute `new_pay_channel` to allow the remote user to receive an incoming channel creation handshake from the initiator. Once the channel has been established, both users will be notified of the channel identifier $c_i$ used to refer to this specific channel. The TEEs of two users 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 23). 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 is created successfully, can the channel $c$ be used by user $u$ and $v$ for future transactions using `pay_channel` (line 24). The sender's TEE sends the specified amount of funds $v$ along the given channel $c_i$ to the remote user. This transaction should include the funds to pay, the channel id, and the current remaining funds of the sender $c$.my_bal. The sender sends the encrypted transaction signed by their TEE to the receiver. The receiver's TEE verifies the integrity by checking the signature. They also need to check if the sender has enough remaining funds to support this transaction (line 26). They prepare an approval notice signed by their TEE's private key and send it back to the receiver. These two users then construct a commitment transaction reflecting their updated balance of the channel in the ledgers. This protocol requires the signatures of both users to authorize a transaction, ensuring that funds can only be moved with the consent of all required users. If one of them wants to close this channel, they need 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. The client who wants to settle her deposit first need to close all her channels by calling close_channel. After all his channels being closed, he can settle his deposit by constructing a transaction with his latest ledger signed by his TEE and send to the Blockchain for confirmation.

### C. 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 V-B, 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 IV. 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. 1. 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. The corresponding ledger entry of channel $c$ in Bob's TEE is set in the same way as Alice's.

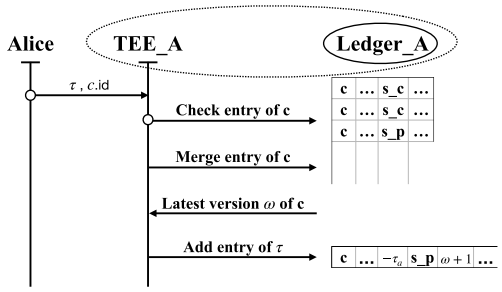



Fig. 6. Illustration of ledger update protocol.

**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 [3]) 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, $\omega$ is set to 0, and it is incremented after each transaction via channel $c$. Suppose Alice initiates the first transaction $\tau$ of amount $\tau_a$ 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 $\omega$ 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 $c$.

For concurrent transactions, we assume Alice wants to make another transaction $\tau'$ with Bob before the last transaction $\tau$ being complete. Alice's TEE first checks the ledger and finds that there is only one entry of channel $c$, but the current status is $s_p$, which means this entry cannot be updated at this time. So $TEE_A$ has to create a new temporary entry of channel $c$, with the Deposit, Channel and Neighbor field same as the original entry, the Amount field to be the transaction amount sent to Bob $-\tau'_a$, the State field to be $s_p$, and the version number $\omega$ to be the largest version number related with channel $c$ so far incremented by 1. Bob performs the same process, expect the Amount field to be the transaction amount received from Alice $\tau'_a$. Every time when a new transaction happens in channel $c$, both $TEE_A$ and $TEE_B$ check the ledger and collect all entries of channel $c$. They needs to merge all the entries with status $s_c$ by summing up the value in the Amount field, and deleting those redundant entries while leaving only one. They also need to find out the largest version number related with channel $c$ which is the latest one, and prepare it for the new transaction.

**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 $\Delta$ for those transactions to complete. After the waiting time $\Delta$, 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.

### D. 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 [9]. 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.

## VI. Protocol Security Analysis

APCN achieves payment security under our attacker model: users can always receive their funds on the blockchain, regardless of attacker's action. We first intuitively define the security guarantees a payment network should provide, and describe the framework we use to construct our proofs.

### A. Security Guarantees

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 [28] similar to prior work [9], [29]. 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, $\pi_{APCN}$, as described in Sec III, 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 II-C. To prove that APCN achieves fund security, we show that (i) the real and ideal worlds are indistinguishable to an external observer $\varepsilon$. 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 $\pi_{APCN}$ 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 [30], 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 $\varepsilon$ 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 $\varepsilon$ 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 $\varepsilon$ exactly $\mathcal{A}$'s output in the real-world. So it is equivalence between $\pi_{APCN}$ and $F_{TEE}$ to $\varepsilon$.

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. II-D.

*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 $\Delta$, 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.

## VII. Performance Evaluation

We present the evaluation results based on prototype implementation and simulations. The evaluations aim to answer the following research questions:

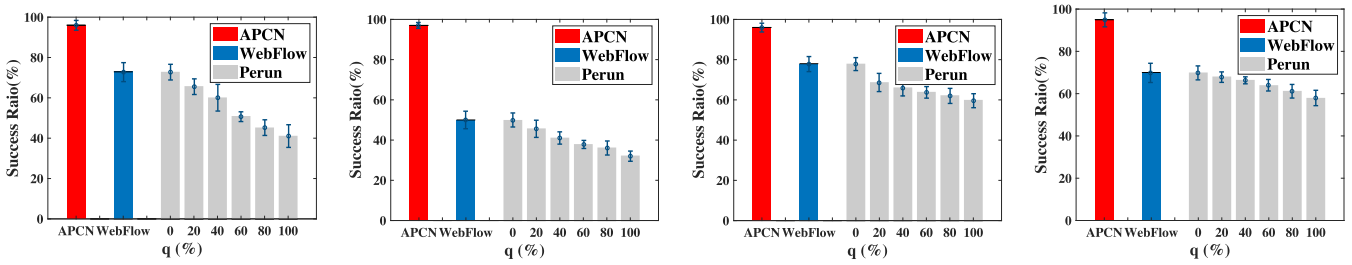
- What is the payment processing latency of APCN?
- How does congestion control mechanism affect the performance of APCN?
- What are the success rate of APCN, compared to other PCN routing under real PCN topologies and traces?

### A. Methodology

We implement the APCN prototype using Intel SGX SDK in C++. The prototype is mainly used for evaluating 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.

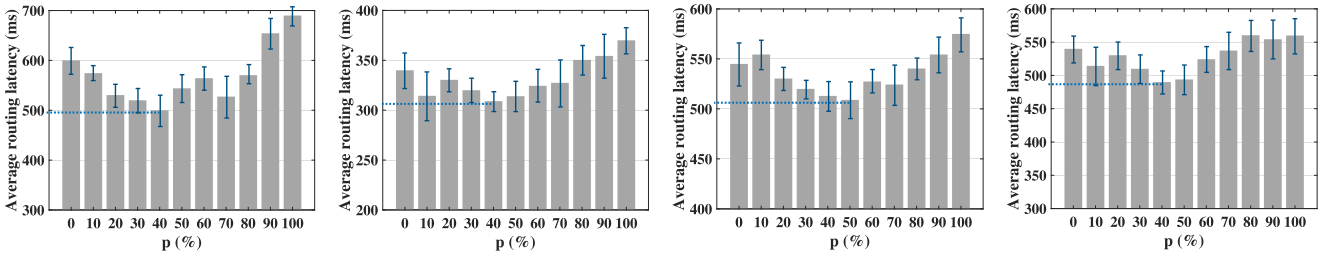
The simulations use two real PCN topologies: Ripple [31] and Lightning [3], as well as synthesis topologies. For Ripple, we use the data from January 2021 to December 2021, and get the network topology with 1,783 nodes and 18,395 edges in our simulation. For Lightning, we get the network topology with 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 generate payments by randomly sampling the Ripple transactions for the Ripple topology. Due to the lack of sender-receiver information in the Lightning network, we randomly sample the transaction volumes and sender-receiver pairs. We build two sets of synthetic PCN topologies based on the Waxman model [32] and the scale-free network model [33]. 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




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

Fig. 7. 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

Fig. 8. The average routing latency with varying $p$ values.

based AES-GCM [34], [35]. To further enhance the security of APCN, we apply T-SGX [20], a countermeasure for controlled-channel attacks.

**Comparisons.** To evaluate the performance of APCN, we compare the following payment channel routing schemes and systems.

*APCN*: The coordinates based greedy routing algorithm with the LA forwarding congestion control mechanism in APCN as introduced in Sec IV-C.

*WebFlow*: An MDT routing algorithm in PCNs, which should consider the channel balances. At each node, it first check all the physical neighbors closer to the receiver to see if they have enough balance to support the transactions, and then check at most 5 DT neighbors.

*SpeedyMurmurs (SM)* [36]: An embedding-based routing algorithm in PCNs that relies on assigning coordinates to nodes to find shorter paths with reduced overhead. We set the number of landmarks to 3 as [36] suggests.

*Spider* [7]: The off-chain routing algorithm in PCNs 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.

*Shortest-Path in PCN (SP):* It serves as the baseline. SP uses the path with the fewest hops between the sender and receiver to route a payment.

*Perun:* [37] Perun is a virtual payment channel system. Virtual channels can be built on top of the parties involved in the multi-party state channels that are connected via a path in the network of ledger channels. The underlying multi-hop path of virtual channel is the max-flow path between two parties. We assume users use the same routing algorithm as APCN to send transactions to recipients. Users can use either payment channels or virtual channels to send or relay transactions. According to Perun [37], we set that users pair locks 50% available funds in the virtual channels in the experiment.

TABLE V
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 |

**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 [8], [36], 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.

### B. 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 V 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 $\Delta$ in our congestion control mechanism in evaluation.

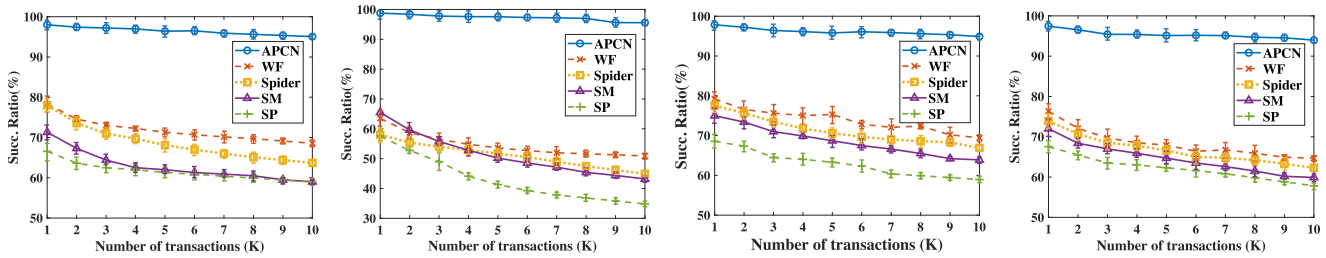
We then consider the case of non-SGX users. we construct a payment channel between two users, one is equipped with SGX, one is not and uses a TEE service provider committee at size of 3. Creation of such a payment channel takes 4.3 secs, as the non-SGX user must verify the integrity of TEEs of the committee. 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 $q$ value from 0% to 100% to test the performance. Here, we use 5,000 transactions in each run. Figure 7 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 influence 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 8 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.

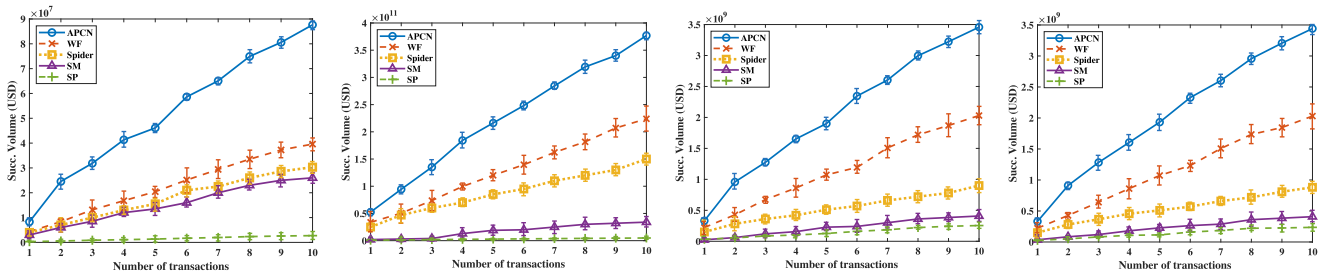
We also include shortest path routing (SP) as the baseline. Figure 12 shows the results of average hopcount, average routing latency and transaction success rate. In all these four topologies, APCN with congestion control has slightly higher average hopcount compared to WebFlow routing protocol. This is because with congestion control mechanism, users may detour to a longer path to avoid busy intermediate nodes. Compared to the baseline protocol, both APCN and WebFlow do not lead to much higher routing stretch. The figure also shows the average routing latency comparison results that APCN achieves lower average latency in all these topologies. This is consistent with our primary goal of congestion control. The routing latency of WebFlow and SP is quite similar because both schemes do not take processing latency of intermediaries into consideration. Besides, the success rate of these three routing schemes are similar in all four topologies. This is because in APCN, the key factor affecting the success rate is the balance of senders instead of paths. The results here demonstrate that our congestion control mechanism reduce the processing latency compared to WebFlow without decreasing the success rate.

**Performance with different networks.** We evaluate APCN with four PCN topologies and a varied number of transactions. As shown in Fig. 9, 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 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%. This is because in APCN, the channels do not keep funds. The transactions only depend on the funds of the senders rather than channels. As long as the sender has enough funds to trigger the payments, the transaction will success if a path is found. The results of success volume in Fig 10 are consistent



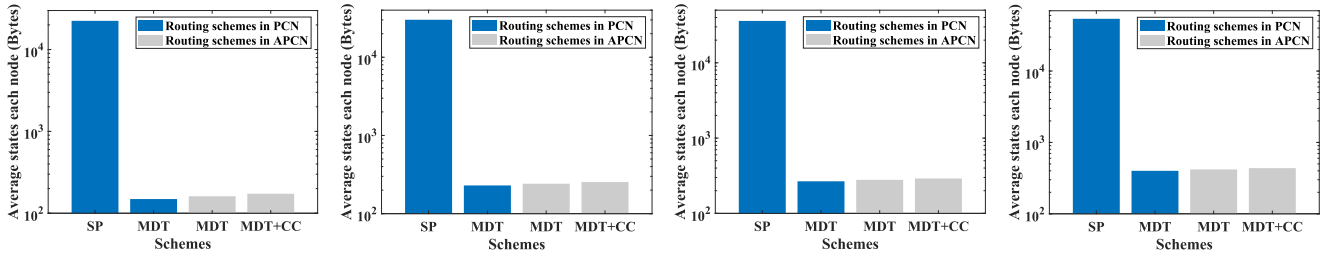

(a) Ripple (b) Lightning (c) Waxman (d) Scale-free

Fig. 9. The success ratio with varying transaction numbers under different network topologies.




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

Fig. 10. The success volume with varying transaction numbers under different network topologies.



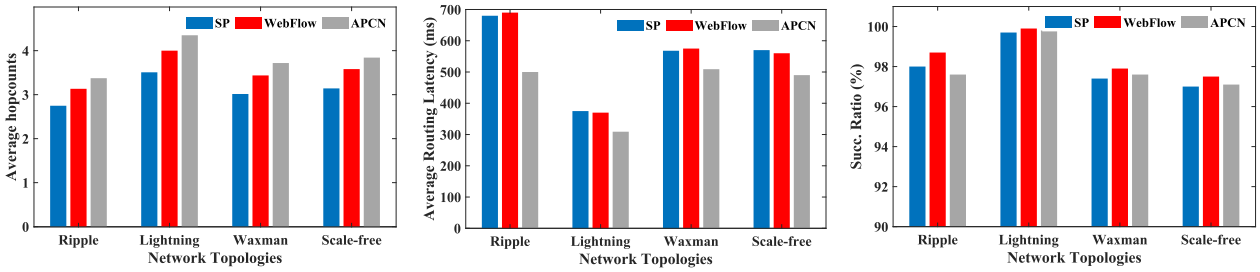

(a) Ripple (b) Lightning (c) Waxman (d) Scale-free

Fig. 11. Storage cost compared with benchmarks in PCN or APCN.

with the result of success ratio. APCN still outperforms than other schemes.

With the increase of the number of transactions, the success ratio and volume of both schemes (MDT with and without congestion control) in APCN are similar and decrease slightly in both Ripple and Lightning topologies as shown in Figure 9 and 10. Because the congestion control mechanism has no effect on success ratio and volume of payments, but only lower the routing latency. These result are much better than WebFlow, the MDT routing protocol in PCNs which is below 80% even when there are only 1000 transactions, and Shortest Path in PCNs which is around 60% with 1000 transactions. Note that even in PCNs, the MDT routing protocol shows much better performance than Shortest Path. The reason is that Shortest Path is static routing where the path for each payment is fixed after path discovery. While MDT is dynamic routing, if one neighbor cannot support the payment, the intermediary will probe other neighbor nodes. It is especially useful for PCNs, as the channel balance changes after each payments.

**Storage cost in each node.** We now show the storage efficiency of APCN by comparing the average states maintained in each node. In Shortest Path, every node needs to locally store the topology of the network, including all the information of the links. Different from this scheme, to perform MDT routing protocol in PCNs, nodes only need to maintain the information of their neighbors, including the coordinates and the channels information. It is similar for MDT routing protocol in the APCN system. For the TEE of each node, it needs to store the coordinates of its neighbors, the balance of itself, and a ledger including an account-based transaction record with each neighbor. It does not need to maintain the channel information since channels in APCN are stateless. For MDT with congestion control in APCN, nodes should keep one more information which is its processing latency of incoming transactions. Figure 11 shows the average states maintained in each node. It is reasonable that routing schemes in APCN cost more than those schemes in PCNs since each node in APCN needs to maintain a ledger locally. But it is still acceptable because the ledgers only create an account-based transaction record for each neighbor instead of keeping all the transaction history, and thus do not introduce too much storage cost. The storage overhead of a ledger is proportional to the number of its neighbors. Due to the memory limitations of TEE, we do have an upper bound of the number of neighbors. During system boot-up, a total of 128 MB is typically reserved for Intel SGX, and 96 MB of that is allocated to the Enclave Page Cache [38], which means it can store up to 100,000 neighbor information




(a) Average number of hops (b) Average processing latency (c) Success ratio

Fig. 12. Performance comparison of routing schemes: Shortest Path, WebFlow and APCN (with congestion control) in APCN system under different network topologies.

theoretically. Currently, the average channels per node in Lightning Network is only 7.85, and 95% of nodes have less than 25 neighbors according to the real-time statistics [39]. The local ledger is totally enough to support the workload in the current Lightning Network. Actually, with the availability of the new Ice Lake processors, the maximum configurable EPC size has been significantly increased, allowing for up to 1 GB of EPC in some configurations. Thus, APCN has the potential to support more complex transactions, records, and even smart contracts.

## VIII. Related Work

PCNs provide a high-throughput solution for blockchains [40]. In PCNs, channels do not always exist between two arbitrary users, and two users can make transactions via a multi-hop path. Hence, routing is crucial to achieve high resource utilization in PCNs. Lightning Network [3] uses max-flow routing algorithms to find paths. Flash [8] also uses modified max-flow routing algorithms but treats elephant and mice payments differently. SilentWhispers [24], Speedy-Murmurs [36], and have been proposed to improve routing scalability.

In order to improve the fund utilization and avoid channel imbalance, Spider [7] develops a multi-path congestion control algorithm. It is a centralized offline routing algorithm and still has a high probing overhead. REVIVE [41] enables users to securely rebalance their channels, according to the preferences of the channel owners. Sprites [42] supports partial withdrawals and deposits, during which the channel can continue to operate without interruption, but requires smart contracts. Teechain [9] 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.

## IX. Discussion and Conclusion

### A. Discussion About the Flash Loans

In this paper, we design APCN to support shared funds among the payment channels and high-throughput transactions between users. Besides normal transactions, APCN can be extended to support complex smart contracts in TEEs. It could enable APCN with more features such as flash loans in blockchains. A flash loan is a loan that is only valid within one atomic blockchain transaction. Flash loans fail if the borrower does not repay its debt before the end of the transaction borrowing the loan. Unlike traditional PCNs where funds are kept in one channel and cannot be moved to other channels, in APCN, funds can circulate throughout the whole network. Thus, users can borrow flash loans from a neighbor, and pay them back later. Users are required to develop a smart contract in TEE, defining the transaction atomicity that guarantees either the loan is returned, or the transaction will fail. The workflow can be generalized into the following steps. First, the lender's TEE transfer requested funds to the borrower. Then borrower can execute operations with borrowed funds. Once the execution is completed, the borrower has to return the borrowed funds with the extra fee charged by the lender. Finally, the lender's TEE will check their balance. If they discover that non-sufficient funds are returned by the borrower, they will revert the transaction immediately. Besides flash loans, APCN has the potential to support more complex features with smart contracts in TEE, enabling a wide range of financial services and applications within the network.

### B. Conclusion

In this paper, 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.

## Acknowledgment

The authors would like to thank Dejun Yang and the anonymous reviewers for their suggestions and comments. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the ARO or the U.S. government. The U.S. government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation herein.

## REFERENCES


[1] X. Zhang and C. Qian, "Towards aggregated payment channel networks," in *Proc. IEEE 30th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2022, pp. 1–11.

[2] K. Croman et al., "On scaling decentralized blockchains," in *Proc. Financial Cryptogr. Data Secur.* Berlin, Germany: Springer, 2016, pp. 106–125.

[3] J. Poon and T. Dryja, "The Bitcoin lightning network: Scalable off-chain instant payments," Lightning Netw., Seattle, WA, USA, Tech. Rep. DRAFT Version 0.5.9.2, 2016.

[4] (2020). *Transaction Rate of Bitcoin*. [Online]. Available: http://www.blockchain.com/en/charts/tran-sactions-per-second

[5] X. Zhang and C. Qian, "A cross-chain payment channel network," in *Proc. IEEE 31st Int. Conf. Netw. Protocols (ICNP)*, Oct. 2023, pp. 1–11.

[6] V. Mavroudis, K. Wüst, A. Dhar, K. Kostiainen, and S. Capkun, "Snappy: Fast on-chain payments with practical collaterals," in *Proc. USENIX NDSS*, 2020, pp. 1–17.

[7] V. Sivaraman et al., "High throughput cryptocurrency routing in payment channel networks," in *Proc. USENIX NSDI*, 2020, pp. 1–21.

[8] P. Wang, H. Xu, X. Jin, and T. Wang, "Flash: Efficient dynamic routing for offchain networks," in *Proc. ACM CoNEXT*, 2019, pp. 370–381.

[9] J. Lind, O. Naor, I. Eyal, F. Kelbert, E. G. Sirer, and P. Pietzuch, "Teechain: A secure payment network with asynchronous blockchain access," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, Oct. 2019, pp. 63–79.

[10] X. Zhang, S. Shi, and C. Qian, "Low-overhead routing for offchain networks with high resource utilization," in *Proc. 42nd Int. Symp. Reliable Distrib. Syst. (SRDS)*, Sep. 2023, pp. 198–208.

[11] D. Kaplan, J. Powell, and T. Woller, "AMD memory encryption," AMD Inc., Santa Clara, CA, USA, White Paper 13, 2016.

[12] *Software Guard Extensions Programming Reference*, R Intel, Intel Corp., Santa Clara, CA, USA, 2014.

[13] Z. Hua, J. Gu, Y. Xia, H. Chen, B. Zang, and H. Guan, "vTZ: Virtualizing ARM trustzone," in *Proc. USENIX Secur.*, 2017, pp. 541–556.

[14] D. Lee, D. Kohlbrenner, S. Shinde, D. Song, and K. Asanović, "Keystone: An open framework for architecting TEEs," 2019, *arXiv:1907.10119*.

[15] H-F Security. (2018). *Multizone: The First Trusted Execution Environment for RISC-V*. [Online]. Available: https://hex-five.com/

[16] Linaro. (2014). *Open Portable Trusted Execution Environment*. [Online]. Available: https://www.op-tee.org/

[17] Intel. (2015). *Product Change Notification*. [Online]. Available: https://qdms.intel.com/dm/i.aspx/5A160770-FC47-47A0-BF8A-062540456F0A/PCN114074-00.pdf

[18] A. Nilsson, P. Nikbakht Bideh, and J. Brorsson, "A survey of published attacks on Intel SGX," 2020, *arXiv:2006.13598*.

[19] M. Li et al., "Bringing decentralized search to decentralized services," in *Proc. USENIX OSDI*, 2021, pp. 331–347.

[20] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, "T-SGX: Eradicating controlled-channel attacks against enclave programs," in *Proc. NDSS*, San Diego, CA, USA, Feb./Mar. 2017.

[21] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring fine-grained control flow inside SGX enclaves with branch shadowing," in *Proc. USENIX Secur.*, 2017, pp. 557–574.

[22] J. Seo et al., "SGX-shield: Enabling address space layout randomization for SGX programs," in *Proc. NDSS*, San Diego, CA, USA, Feb./Mar. 2017.

[23] M. Armanuzzaman and Z. Zhao, "BYOTEE: Towards building your own trusted execution environments using FPGA," 2022, *arXiv:2203.04214*.

[24] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "SilentWhispers: Enforcing security and privacy in decentralized credit networks," in *Proc. NDSS*, San Diego, CA, USA, Feb./Mar. 2017.

[25] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 455–471.

[26] *Intel Software Guard Extensions Remote Attestation End-to-End Example*, CS Intel, Santa Clara, CA, USA, 2018.

[27] (2024). *Blockcypher*. [Online]. Available: https://live.blockcypher.com/btc/pushtx/

[28] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proc. 42nd IEEE Symp. Found. Comput. Sci.*, Oct. 2001, pp. 136–145.

[29] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 949–966.

[30] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1521–1538.

[31] F. Armknecht, G. O. Karame, A. Mandal, F. Youssef, and E. Zenner, "Ripple: Overview and outlook," in *Trust and Trustworthy Computing*. Heraklion, Greece: Springer, 2015.

[32] B. M. Waxman, "Routing of multipoint connections," *IEEE J. Sel. Areas Commun.*, vol. 6, no. 9, pp. 1617–1622, Dec. 1988.

[33] (2014). *N Developers*. [Online]. Available: https://networkx.github.io/documentation/networkx-1.9.1/reference/generated/networkx.generators.random_graphs.barabasi

[34] Intel. *Intel 64 and IA-32 Architectures Software Developer Manuals*. Accessed: May 2011. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

[35] R Intel. *Intel(R) Software Guard Extensions for Linux Os*. Accessed: 2016. [Online]. Available: https://github.com/intel/linux-sgx

[36] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," in *Proc. USENIX NDSS*, 2017, pp. 1–15.

[37] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," in *Proc. IEEE SP*, 2019, pp. 1–19.

[38] (2017). *Enclave Memory Measurement Tool for Intel Software Guard Extensions (Intel SGX) Enclaves*. [Online]. Available: https://www.intel.com/content/dam/develop/external/us/en/documents/enclave-measurement-tool-intel-sgx-737361.pdf

[39] (2024). *Real-Time Lightning Network Statistics*. [Online]. Available: https://1ml.com/statistics

[40] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine agreements for cryptocurrencies," in *Proc. 26th Symp. Operating Syst. Princ.*, Oct. 2017, pp. 7747–7758.

[41] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 439–453.

[42] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *Proc. Financial Cryptography and Data Security*. Springer, 2019, pp. 508–526.


**Xiaoxue Zhang** (Member, IEEE) received the B.E. degree in computer science and engineering from the University of Science and Technology of China in 2019 and the Ph.D. degree in computer engineering from the University of California at Santa Cruz in 2024. She is currently an Assistant Professor with the Department of Computer Science and Engineering, University of Nevada Reno, Reno. Her research areas are computer networks, payment channel networks, and routing. She also focuses on the scalability and security problems of blockchain.

**Chen Qian** (Senior Member, IEEE) received the B.Sc. degree in computer science from Nanjing University in 2006, the M.Phil. degree in computer science from The Hong Kong University of Science and Technology in 2008, and the Ph.D. degree in computer science from The University of Texas at Austin in 2013. He is currently a Professor with the Department of Computer Science and Engineering, University of California at Santa Cruz. He has published more than 80 research papers in a number of top conferences and journals, including ACM SIGMETRICS, IEEE ICNP, IEEE ICDCS, IEEE INFOCOM, IEEE PerCom, ACM UBICOMP, ACM CCS, IEEE/ACM TRANSACTIONS ON NETWORKING, and IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. His research interests include computer networking, quantum networks, data-center networks and cloud computing, the Internet of Things, and software defined networks. He is a Senior Member of ACM. He received the NSF CAREER Award in 2018.