BAR: A Balance-aware Routing Protocol in Payment Channel Networks

Qiushi Wei, Yuhui Zhang, Dejun Yang, and Guoliang Xue

***Abstract*—Payment channel networks (PCNs) have been proposed to tackle the scalability issues in blockchains by enabling off-chain transaction settlement. However, the balance depletion problem caused by unidirectional transactions may jeopardize the payments in PCNs. Existing works address this problem by sending artificial payments to rebalance the payment channels. In this paper, we take advantage of a unique property in PCNs, where the payments of opposite directions between two users can cancel each other out, to mitigate this channel depletion problem. Specifically, we design BAR, a distributed balance-aware payment routing protocol, subject to fee-based conservation, timeliness, and feasibility constraints. Moreover, to ensure payment security, we modify the original Hashed Time-Lock Contract (HTLC) protocol to adapt it to BAR, such that BAR achieves efficiency and atomicity. Extensive simulations demonstrate that BAR outperforms the state-of-the-art algorithms Spider [1] and LND [2] in terms of success ratio and success volume.**

I. Introduction

Over the past decade, blockchain-based cryptocurrencies have risen to more than $800 billion in peak capital, e.g., Bitcoin [3] and Ethereum [4]. Payment channel networks (PCNs), e.g., Bitcoin Lightning Network (LN) [2] and Ethereum Raiden Network [5], have been proposed to tackle the scalability issues. A payment in PCNs is routed through multiple intermediate channels (ICs). During the payment routing, an intermediate user (IU) who forwards a payment gains funds from its *transferor* and forwards funds to its *transferee*, which causes undesired balance depletion of certain channels. In existing PCN protocols [1, 6–9], channels must be closed and reopened, leading to costly on-chain operations, while also overlooking several constraints uncommon in traditional ad-hoc networks. The first constraint is the **feasibility constraint**, requiring that each involved IC's balance can cover the payment. The second constraint is the **timeliness constraint** because the tolerance of each user's cooperating time varies. Moreover, payment transactions in opposite directions can cancel each other out. A payment transaction through a channel is a fund **redistribution** between two users. Thus, we take advantage of this unique property to improve the performance of payment routing as shown in Fig. 1.

In this paper, we investigate balance-aware routing, which fulfills payments while considering the channel balance. First, a balance-aware routing protocol should offer a **rebalancing** option for channels. Second, a balance-aware routing should be **distributed**. Third, a balance-aware routing protocol should satisfy **efficiency**, *i.e.*, to reduce the latency incurred by transmitting multiple payments through one channel. Lastly, a balance-aware routing should satisfy **atomicity**, such that a payment can be sent as several partial payments, and the recipient cannot claim the payment until it receives all partial payments. Facing these challenges, we propose BAR, a balance-aware routing protocol that satisfies the above properties. In addition, Atomic Multi-Path Payment (AMP) [10] ensures secure payments in PCNs, thus we modify the original HTLC to provide efficiency and adapt it to the AMP. The main contributions of this paper are:

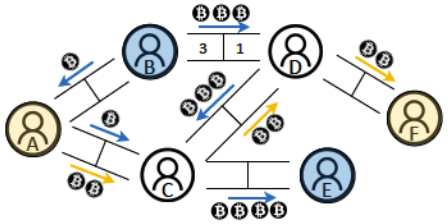


Fig. 1. An example illustrating the unique properties of routing in PCNs. A sends ฿2 to F; B sends ฿4 to E. Two values between two nodes represent the fund distribution in the channel, e.g., A and B have ฿4 and ฿2 in channel (A, B), respectively. If we consider these two payments separately, neither would be possible due to channel (C, D). However, since a payment transaction through a channel is a fund redistribution between two users, the two transactions (฿2 from C to D and ฿3 from D to C) are essentially one transaction of ฿1 from D to C.

- To the best of our knowledge, we are the first to consider balance-aware routing and propose a distributed balance-aware routing protocol, which reaches high throughput while considering the channel balance in PCNs.
- We investigate important design goals of payment routing in PCN, which are referred to as channel balance awareness, distributedness, efficiency, and atomicity.
- We achieve channel balance awareness and distributedness by designing a distributed algorithm while considering channel balances.
- We modify the original HTLC to provide efficiency and adapt it to balance-aware routing to guarantee atomicity.
- Extensive simulations demonstrate that BAR achieves superior success ratio and transaction throughput over the state-of-the-art algorithms Spider [1] and LND [2].

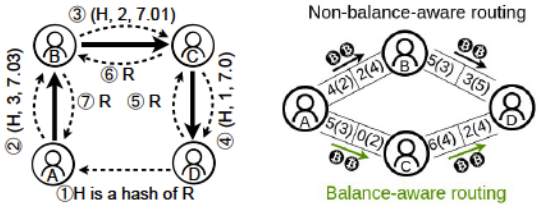
II. Background

A. Payment Channel and Hashed Time-Lock Contract

To overcome the scalability issue, payment channels have been proposed to eliminate the need to commit each transaction to the blockchain. Two users establish a payment channel by each depositing a certain amount into a joint account and broadcasting this transaction on the blockchain. When the

Wei, Zhang, and Yang ({qiushiwei, yuhzhang, djyang}@mines.edu) are with Colorado School of Mines, Golden, CO 80401, USA. Xue (xue@asu.edu) is with Arizona State University, Tempe, AZ 85281, USA. This work was supported by NSF grants CNS-2008935, CNS-2414522, and CNS-2007083.

channel closes, a closing transaction is broadcast and will send funds to each user based on the latest distribution. To enable transactions between any two users, payments can be routed through multiple ICs. HTLC was introduced in PCN [2], where the recipient generates a preimage R and sends its hash H to the sender. The sender and all IUs include H in the contract, such that the transferred payment can be claimed by the transferee only when R is provided to the transferor within an HTLC tolerance. HTLC ensures that a user can pull the payment from its transferor after its payment has been pulled by its transferee. The tolerance is the number of hops to the recipient [2] in LN [11]. For example, the HTLC $(H, 2, 7.01)$ from B to C in Fig. 2(a) means that C can receive ฿7.01 from B, if C provides the preimage R of H within $2T$.



(a) HTLC (b) Balance awareness

Fig. 2. (a) The recipient D receives a payment of ฿7 from the sender A via B and C with an HTLC tolerance of 3. Circled numbers represent the sequence of the operations. (b) Two values outside and inside the brackets in a channel represent the fund distribution before and after routing, respectively.

B. Related Work

The payment routing in PCNs has received increasing attention. Zhang *et al.* [6] proposed a distributed algorithm CheaPay to minimize the transaction fee. Wang *et al.* [7] developed a dynamic routing algorithm Flash, which differentiates elephant payments from mice payments. However, all previous studies ignore the channel depletion issue. Later, Subramanian *et al.* [8] designed a decentralized rebalancing in acyclic PCNs. Ni *et al.* [12] designed a cyclic rebalancing protocol to maximize the utility of payment channels. However, they treat rebalancing as a separate task independent of the payment routing. More recently, Panwar *et al.* [13] designed a routing protocol SPRITE for concurrent transactions. Sivaraman *et al.* [1] developed Spider, a payment routing algorithm maximizing the throughput. However, they do not offer rebalancing options and centralized schemes have high probing overhead. Therefore, we propose a distributed balance-aware high-throughput routing protocol BAR to fulfill the payments while rebalancing the channels.

III. System Model and Design Goals

A. System Model

A PCN can be represented as a directed graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the set of edges [1, 8, 14, 15] . Each node $v_i \in \mathcal{V}$ represents a user with a cryptocurrency account and establishes at least one payment channel with a peer user. Each edge $e_{i,j} = (v_i, v_j) \in \mathcal{E}$ represents a payment channel, where v_i is the *transferor* and v_j is the *transferee*. Each node v_i is associated with an HTLC tolerance τ_i, denoting the maximum time that v_i would wait for the preimage R provided by its transferee. Each edge is associated with several attributes. First, each edge $(v_i, v_j) \in \mathcal{E}$ has a channel balance $b_{i,j}$, denoting the amount of the remaining funds that v_i owns on this channel. Second, each edge has a rebalancing binary flag $r_{i,j}$ to denote whether this channel desires rebalancing. If $r_{i,j} = 1$, it indicates that the channel (v_i, v_j) is balance-aware, *i.e.,* the channel would like to be rebalanced in the direction from v_j to v_i. Third, for each edge, let $\delta_{i,j}$ denote the maximum amount that v_i is willing to transfer to v_j. It is natural to assume that $\delta_{i,j} \leq b_{i,j}$ because v_i cannot transfer more than it owns on this channel. The rationale is that v_i may want to limit the payment amount to avoid depletion. On the other hand, the channel (v_j, v_i) is rebalanced when v_i transfers a payment to v_j. Fourth, for a payment request (v_s, v_t, a), v_s and v_t are the **sender** and **recipient**, respectively, and a is the payment amount. Each intermediate node v_i can charge a fee denoted as $\phi_{i,j}^{s,t}$ for forwarding the payment to node v_j while utilizing the channel $e_{i,j}$. Let γ and b denote the proportional rate, and the base fee [11], respectively. Let $f_{i,j}^{s,t}$ denote the amount of forwarded payment on the channel (v_i, v_j) to fulfill the payment request. $\phi_{i,j}^{s,t}$ consists of the proportion portion $\gamma f_{i,j}^{s,t}$ and the base fee portion b: $\phi_{i,j}^{s,t} = \gamma f_{i,j}^{s,t} + b$.

There can be n split paths for a payment request to deliver the total amount. Suppose that the p^{th} path r_p denoted by a sequence $v_0 \to v_1 \to \ldots \to v_{l_p}$, where $v_0 = v_s$ and $v_{l_p} = v_t$. The delivered amount on r_p is a_p. Then the total transaction fee $\Phi^{s,t}$ paid by v_s should be the sum of the transaction fee charged by all IUs along the n paths, for $j = 2, \ldots, l_p - 1$:

$$\Phi^{s,t} = \sum_{p=1}^{n} \sum_{i=1}^{l_p - 1} (\gamma(a_p + \sum_{k=j+1}^{l_p} \phi_{j,k}^{s,t}) + b) \tag{1}$$

Let $\mathcal{Q} = \{(v_{s_1}, v_{t_1}, a_1), \ldots, (v_{s_n}, v_{t_n}, a_n)\}$ denote a set of n payment requests. Here we abuse the notation $f^{s,t} \in f$ to represent that an (s,t) payment flow $f^{s,t}$ is involved in the overall payment flow f. Each payment request is performed via an (s,t) *payment flow* defined as follows.

Definition 1 (Payment Flow). *Given a PCN $G = (\mathcal{V}, \mathcal{E})$ and a payment request (v_s, v_t, a), an (s,t) payment flow is a function $f^{s,t} : \mathcal{E} \to \mathbb{R}^{\geq 0}$, such that $\sum_{(v_i, v_t) \in \mathcal{E}} f_{i,t}^{s,t} = a$.*

Besides, we assume that each user only has local knowledge of its incoming and outgoing channels, including HTLC tolerances, maximum transfer amount, and rebalancing flags, due to privacy concerns and dynamic network changes [14, 15].

B. Design Goals and Constraints

There are four desirable design goals: **Channel Balance Awareness**: A payment routing protocol satisfies channel balance awareness if it provides the rebalancing option. We take advantage of the redistribution of the channel balances to rebalance the channels off-chain. An example illustrates the importance of channel balance awareness in Fig. 2(b). Sender A sends ฿2 to D. Two payment paths can fulfill this payment: $A \to B \to D$ and $A \to C \to D$. In the balance-aware routing, (A, C) has a higher priority to be involved

in routing to rebalance (A, C). **Distributedness**: A payment routing protocol satisfies distributedness if it does not rely on a centralized party. Centralized routing is subject to a single point of failure, and hence cannot be trusted. **Efficiency:** A payment routing protocol satisfies efficiency, if it minimizes the payment latency incurred by transmitting multiple payments through a payment channel simultaneously. **Atomicity:** A payment routing protocol satisfies atomicity for security purposes, if a sender can split and send a large payment into several partial payments, and a recipient cannot claim payment until it receives all partial payments. To achieve these goals, we aim to design a high-throughput routing protocol that finds payment flows satisfying the following constraints:

1) Fee-based conservation constraint: The total incoming flow equals the sum of the total outgoing flow and the fees if $\forall v_i \in \mathcal{V} \backslash \{v_s, v_t\}$: $\sum_{(v_k,v_i)\in\mathcal{E}} f_{k,i}^{s,t} = \sum_{(v_i,v_j)\in\mathcal{E}} (f_{i,j}^{s,t} + \phi_{i,j}^{s,t})$. This constraint guarantees that the IUs only gain transaction fees by forwarding the payment requests.

2) Feasibility constraint: The payment flows are feasible if $\forall v_i \in \mathcal{V}$, $0 \leq \sum_{(v_s,v_t,a)\in\mathcal{Q}} (f_{i,j}^{s,t} - f_{j,i}^{s,t}) \leq \delta_{i,j}$.

3) Timeliness constraint: An (s, t) payment flow satisfies timeliness constraint, if $\forall (v_i, v_{i+1})$ along an (s, t) payment path $r = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_L$ where $v_0 = v_s$ and $v_L = v_t$, $\tau_i \geq L - i, \forall i \in [0, L-1]$. This ensures that payment requests are fulfilled within the HTLC tolerance of each IC, which guarantees the commitment to honest processing at any IC.

IV. Design of BAR

A. Design Challenges

We propose BAR, a distributed balance-aware routing high-throughput protocol. BAR achieves channel balance awareness and distributedness by designing a distributed algorithm derived from Goldberg-Tarjan algorithm [16] and Dinic's algorithm [17], which were initially to tackle the maximum flow problem in centralized systems. In Goldberg-Tarjan algorithm, each node finds augmenting paths based on local knowledge about its neighbors. In Dinic's algorithm, each node finds augmenting paths based on node heights in a depth-first-search way. Meanwhile, BAR achieves efficiency by modifying HTLC and ensures atomicity by adapting it to AMP [10].

There are several challenges to be addressed: (1) To design a distributed algorithm, each node should execute the same local algorithm. (2) To satisfy the timeliness constraint, the distance (the number of edges) from a node to the recipient should be labeled to guarantee that it does not exceed the node's HTLC tolerance. (3) On the one hand, the Goldberg-Tarjan algorithm is suitable for the distributed PCNs, because each node makes decisions locally based on the knowledge about its transferors and transferees. It makes use of a height on each node to pull flows from uphill transferors. However, a node's height does not always reflect its distance, because the flows can only be pulled back from the transferees by lifting the height of a transferee. On the other hand, the Dinic's algorithm finds blocking flows by depth-first-search, which is not suitable for the distributed PCNs. Hence, we need to modify the current algorithms carefully to satisfy distributedness. (4) The payment requests in opposite directions cancel each other in PCNs, which is a unique characteristic that does not exist in the conventional ad-hoc network. The fundamental reason is that the data packets transmitted in the ad-hoc network differentiate in opposite directions, while the payments in PCNs do not. However, the current PCN protocol does not support this, because each HTLC is associated with a transaction to guarantee the off-chain security almost the same as the original blockchain. Although the cancellation of multiple transactions can result in one transaction, the HTLCs are constructed and executed separately. To enable transaction cancellation that involves multiple transactions and HTLCs, we need to modify the current HTLC protocol efficiency and atomicity as well as off-chain security. We address these challenges below and explain how users execute payment requests in BAR.

B. Payment Flow Construction

1) Key Notations: **Loss** is the inverse concept of the **excess** from the original Goldberg-Tarjan algorithm [16]. The loss of v_i is denoted as x_i, which is computed first by summing the total payment flows forwarded by v_i and the fees charged by v_i, then deducting the total incoming flows of v_i: $x_i = \sum_{v_j\in\mathcal{O}_i} f_{i,j}^{s,t} + \sum_{v_j\in\mathcal{O}_i} \phi_{i,j}^{s,t} - \sum_{v_k\in\mathcal{I}_i} f_{k,i}^{s,t}$. **Height** is denoted as h_i^t, an integer that labels the number of hops from v_i to v_t. A node v_j is v_i's **uphill transferor**, if $\delta_{j,i} > 0$ and v_j's height is 1 unit higher than v_i's height, i.e., $h_j^t = h_i^t + 1$. A node v_j is v_i's **downhill transferee**, if $\delta_{i,j} > 0$ and v_j's height is 1 unit lower than v_i's height, i.e., $h_j^t = h_i^t - 1$.

2) Algorithm Design: BAR leverages the concept of node heights that allows each node to make local decisions by exchanging messages with its transferors and transferees [16, 17]. Additionally, fees can not be determined before routing finishes, since the flow that goes through each IU cannot be determined before routing finishes. However, we notice that the total forwarded payment amount to the recipient equals the payment demand, which remains constant and unaffected by the routing path selection. Routing from the recipient to the sender enables simultaneous determination of routing paths and forwarding fees. When an IU sends outgoing flow before identifying its flow source, it incurs a temporary positive flow loss. The payment flow construction consists following stages: **Request, Initialization, Label, Pull-Pullback, and Update**.

In Algorithm 1, each node v_i runs BAR-Request to generate a payment flow that constitutes the overall transaction (tx) flow. BAR-Request adds the payment demand a to the recipient v_t's loss (Line 1). The sender v_s broadcasts a relabel message to relabel the height for each node when no IU incurs any loss (Line 2). Then, BAR-Request calls BAR-Init and BAR-Label to initialize rebalancing flags and height for node v_i, respectively (Line 3). When $x_i > 0$, v_i starts to pull and pullback (Lines 4 to 6). If the rebalancing parameter of v_i on the channel $e_{i,j}$ is depleted (Line 7), the rebalancing flag is set as 1. If the tx is partially satisfied, all other nodes will receive the relabel message to update their heights (Line 8). Once the request is fulfilled or times out, BAR-Request terminates and returns an (s, t) payment flow $f^{s,t}$ (Line 9).

Algorithm 1: BAR-Request

Input: a network $G = (\mathcal{V}, \mathcal{E})$, a node v_i
Output: an (s,t) payment flow $f^{s,t}$

$f^{s,t} \leftarrow \emptyset$; **if** $v_i = v_t$ **then** $x_t \leftarrow x_t + a$;
if $v_i = v_s$ **then** Broadcast the relabel message;
BAR-Init(G, v_i); BAR-Label(G, v_i);
while $x_i > 0$ **and** $v_i \neq v_s$ **do**
  $(G, \Delta^{s,t}) \leftarrow$ BAR-Pull-PullBack(G, v_i);
  $f^{s,t} \leftarrow f^{s,t} \cup \Delta^{s,t}$
if $\delta_{i,j} = 0$ **then** $r_{i,j} = 1$;
$h_i^t \leftarrow$ BAR-Label(G, v_i) upon receiving relabel message ;
return $f^{s,t}$

Algorithm 2: BAR-Init

Input: a network $G = (\mathcal{V}, \mathcal{E})$, a node v_i
Output: v_i

1 $x_i \leftarrow 0$; $\mathcal{O}_i^0 \leftarrow \emptyset$; $\mathcal{O}_i^1 \leftarrow \emptyset$; $\mathcal{I}_i^0 \leftarrow \emptyset$; $\mathcal{I}_i^1 \leftarrow \emptyset$;
2 $\mathcal{O}_i \leftarrow \mathcal{O}_i^1 \cup \mathcal{O}_i^0$; $\mathcal{I}_i \leftarrow \mathcal{I}_i^1 \cup \mathcal{I}_i^0$;
3 **for** $v_j \in \mathcal{O}_i$ **do** // Initialize rebalancing flags
4 $f_{i,j}^{s,t} \leftarrow 0$;
5 **if** (v_i, v_j) desires rebalancing **then**
6 $r_{i,j} \leftarrow 1$; $\mathcal{O}_i^1 \leftarrow \mathcal{O}_i^1 \cup \{v_j\}$;
7 **else** $r_{i,j} \leftarrow 0$; $\mathcal{O}_i^0 \leftarrow \mathcal{O}_i^0 \cup \{v_j\}$;
8 **for** $v_k \in \mathcal{I}_i$ **do** Repeat Steps 4 to 7 ;
9 **return** v_i

In Algorithm 2, each node v_i runs BAR-Init to initialize its loss and rebalancing flag. The loss of v_i is initialized to 0. $\mathcal{O}_i^0$ and $\mathcal{O}_i^1$ denote v_i's outgoing transferees who don't and do wish to rebalance, respectively (Lines 1 to 2). Similarly for $\mathcal{I}_i^0$ and $\mathcal{I}_i^1$, which denote the set of v_i's incoming transferors and the set of v_i's outgoing transferees, respectively. Then v_i sets the rebalancing flags based on the rebalancing desires of its transferees and its transferors (Lines 3 to 8).

In Algorithm 3, each node v_i runs BAR-Label to label its height in a breadth-first-search way. v_i first initializes its height as -1 (Line 1). The height of the recipient v_t is set as 0 and v_t broadcasts its height to all its neighbors (Lines 2 to 3). When v_i receives h_j^t from its transferee v_j, if its height is newly set or greater than its newly updated height, then v_i updates its height (Lines 4 to 6). If h_i^t exceeds its HTLC tolerance τ_i or is not reachable to v_t, it is invalid for routing (Line 7). Once v_i finishes updating its height, v_i broadcasts h_i^t (Line 8).

In Algorithm 4, each node with positive loss shall pull flow. v_i pulls its loss from its transferors whose heights are $h_i^t + 1$ until its loss turns 0 or there are no valid transferors (Lines 1 to 5). Note that the channel directions with rebalancing flags as 1 are preferred to be pulled from. $\phi_{k,i}^{s,t}$ is computed and broadcast with $\Delta_{k,i}^{s,t}$ (Lines 6 to 7). v_k updates its loss and rebalancing parameter by running BAR-Update (Line 8). Each node shall pull back its loss if it remains positive. (Lines 9 to 11). For the pullback operation, both the sent flow from node v_i to v_j and the sent fee should be pulled back to node v_i. Two cases of the returning fee $\epsilon_{j,i}^{s,t}$ are: (1) return the whole fee; (2) return a partial fee. Note that when the returned flow $\Delta_{j,i}^{s,t}$ equals the previously pulled flow, the whole fee $\phi_{i,j}^{s,t}$

Algorithm 3: BAR-Label

Input: a network $G = (\mathcal{V}, \mathcal{E})$, a node v_i
Output: the height h_i^t of node v_i

1 $h_i^t \leftarrow -1$;
2 **if** $v_i = v_t$ **then**
3 $h_i^t \leftarrow 0$; Broadcast h_i^t to all nodes in $\mathcal{I}_i \cup \mathcal{O}_i$;
4 **for** $v_j \in \mathcal{O}_i$ upon v_i receiving h_j^t **do**
  // if v_i's height is firstly set or new height's smaller
5 **if** $\delta_{i,j} > 0$ **and** $(h_i^t = -1$ **or** $h_j^t + 1 < h_i^t)$ **then**
6 $h_i^t \leftarrow h_j^t + 1$
7 **if** $h_i^t > \tau_i$ **or** $h_i^t = -1$ **then** $h_i^t \leftarrow \infty$;
8 Broadcast h_i^t to nodes in $\mathcal{I}_i \cup \mathcal{O}_i$;
9 **return** h_i^t

Algorithm 4: BAR-Pull-PullBack

Input: a network $G = (\mathcal{V}, \mathcal{E})$, a node v_i.
Output: a residual graph G, a payment flow $\Delta^{s,t}$

1 **for** $v_k \in \mathcal{I}_i$ **and** $h_k^t = h_i^t + 1$ **do** // pull
2 v_k and v_i exchange message of $\delta_{k,i}$;
3 **if** $\delta_{k,i} > 0$ **then**
4 $\Delta_{k,i}^{s,t} \leftarrow \min\{x_i, \delta_{k,i}\}$; $x_i \leftarrow x_i - \Delta_{k,i}^{s,t}$;
5 $\delta_{i,k} \leftarrow \delta_{i,k} - \Delta_{k,i}^{s,t}$; $f_{k,i}^{s,t} \leftarrow f_{k,i}^{s,t} + \Delta_{k,i}^{s,t}$;
6 $\phi_{k,i}^{s,t} \leftarrow \phi_{k,i}^{s,t} + \gamma \Delta_{k,i}^{s,t} + b$;
7 Broadcast $\Delta_{k,i}^{s,t}$ and $\phi_{k,i}^{s,t}$ to v_k;
8 BAR-Update(G, v_k);
9 **for** $v_j \in \mathcal{O}_i$ **and** $h_i^t = h_j^t + 1$ **do** // pullback
10 v_j and v_i exchange message of $\delta_{j,i}$;
11 **if** $\delta_{j,i} > 0$ **then**
12 **if** $\gamma \frac{x_i - b}{1+\gamma} + b = \phi_{i,j}^{s,t}$ **then** // return whole fee
13 $\Delta_{j,i}^{s,t} \leftarrow \min\{\frac{x_i - b}{1+\gamma}, \delta_{j,i}\}$; $\epsilon_{j,i}^{s,t} \leftarrow \phi_{i,j}^{s,t}$;
14 **else** // return partial fee
15 $\Delta_{j,i}^{s,t} \leftarrow \min\{\frac{x_i}{1+\gamma}, \delta_{j,i}\}$; $\epsilon_{j,i}^{s,t} \leftarrow \gamma \Delta_{j,i}^{s,t}$;
16 $x_i \leftarrow x_i - \Delta_{j,i}^{s,t} - \epsilon_{j,i}^{s,t}$; $\phi_{i,j}^{s,t} \leftarrow \phi_{i,j}^{s,t} - \epsilon_{j,i}^{s,t}$;
17 $\delta_{i,j} \leftarrow \delta_{i,j} + \Delta_{j,i}^{s,t}$; $f_{i,j}^{s,t} \leftarrow f_{i,j}^{s,t} - \Delta_{j,i}^{s,t}$;
18 Broadcast $\Delta_{j,i}^{s,t}$ to v_j; BAR-Update(G, v_j);
19 **return** $(G, \Delta^{s,t})$

including the base fee b is returned (Lines 12 to 13), this is to set the channel $e_{i,j}$ back to unused state. Otherwise, $\epsilon_{j,i}^{s,t}$ only covers the proportional rate portion $\gamma \Delta_{j,i}^{s,t}$ (Line 15). Then the downhill transferee v_j runs BAR-Update to update its own loss and rebalancing parameter (Line 18). To summarize, each node first initializes and labels its height by exchanging $O(|\mathcal{E}|)$ messages in both BAR-Init and BAR-Label. Next, each node exchanges $O(|\mathcal{V}||\mathcal{E}|)$ messages in BAR-Pull-PullBack. Running BAR-Request terminates within $|\mathcal{V}|$ iterations. Thus, the overall message complexity of BAR is $O(|\mathcal{V}|^2|\mathcal{E}|)$.

C. Modified HTLC Establishment

HTLC [2] is a script permitting a designated party (the transferee) to spend funds by disclosing the preimage of a hash, originally designed for payment routing in a single

Algorithm 5: BAR-Update

Input: a network $G = (\mathcal{V}, \mathcal{E})$, a node v_i.
Output: a residual graph G, a payment flow $\Delta^{s,t}$

if $h_i^t = h_j^t + 1$ **or** $h_i^t = h_j^t - 1$ upon v_i receiving $\Delta_{i,j}^{s,t}$ and $\phi_{i,j}$ **then** $\delta_{i,j} \leftarrow \delta_{i,j} - \Delta_{i,j}^{s,t}$; $x_i \leftarrow x_i + \Delta_{i,j}^{s,t}$;

if $v_i \neq v_s$ **or** v_t **then** $x_i \leftarrow x_i + \phi_{i,j}^{s,t}$;

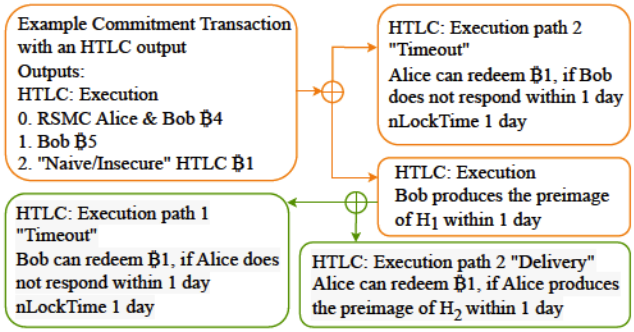
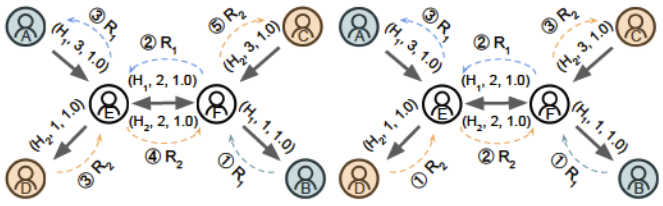


Fig. 3. HTLC Establishment. Alice (A) sends ฿1 to Bob (B) and B sends ฿1 to A via an HTLC tolerance of 2. There are 3 possible spends from an HTLC output. If B does not produce the preimage of H_1 within 1 day, then HTLC terminates. They can redeem path 2. If B produces the preimage of H_1 within 1 day, B does not need to reveal it. Then, if A produces the preimage of H_2 within another 1 day, they reveal and swap R_i and R_2 via fair exchange and redeem path 2. If A does not produce the preimage of H_2 within another 1 day, B reveals the preimage of H_1 to A and redeems path 1.

payment path. Multiple payment requests that go through the same payment channel have to be fulfilled sequentially. An illustration of the modified HTLC adapted to BAR is shown in Fig. 3: Suppose two txs q_1 and q_2 are in opposite directions on a payment channel $A \leftrightarrow B$, and $a_1 \geq a_2$ without loss of generality, where a_i is the payment amount of q_i. Then q_1 can split a_1 into two partial payments, *i.e.,* a_2 and $a_1 - a_2$. To cancel a_2 in both directions, the modified HTLC holds a_2 from A and takes 2 hashes as the input, which are the hash H_1 of the preimage R_1 and the hash H_2 of the preimage R_2. Specifically, the modified HTLC has 2 possible outputs depending on the preimages provided by both A and B. If B does not produce R_1 within T_1, then the HTLC terminates, and A gets the refund a_2 without requiring R_2 from path 2. If B produces R_1 within T_1 to prove that the recipient on path 1 has received a_2, then the HTLC extends for a time duration of T_2. Note that B can prove the correctness of R_1 by Zero-Knowledge Proof [18] without revealing R_1. If A produces R_2 within T_2, then A and B reveal and swap R_2 and R_1, and A gets a_2 refunded. If A does not produce R_2 within T_2, then B reveals R_1 and receives a_2. Such modified HTLC guarantees efficiency and security for balance-aware routing.

An example compares the efficiency of the original HTLC and the modified HTLC in Fig 4. Two payment requests go through the same payment channel in opposite directions. When adopting the original HTLC shown in Fig. 4(a), the payment request from C to D can only be fulfilled after the payment from A to B is fulfilled. In total, it consumes the liquidity of up to $5T \cdot 1 + 4T \cdot 1 + 3T \cdot 1 + 3T \cdot 1 + 2T \cdot 1 + 1T \cdot 1 = 18T \cdot 1$ to fulfill both payment requests. When adopting the modified HTLC shown in Fig. 4(b), both payment requests



(a) Original HTLC (b) Modified HTLC

Fig. 4. Payment Forwarding HTLC establishment in BAR. A sends ฿1 to B. C sends ฿1 to D. Circled numbers represent the sequence of the operations.

can be fulfilled. In total, it consumes the liquidity of up to $3T \cdot 1 + 2T \cdot 1 + 1T \cdot 1 + 3T \cdot 1 + 2T \cdot 1 = 6T \cdot 1$ to fulfill both payment requests. Thus, the modified HTLC can significantly improve the efficiency of BAR. A large payment sometimes has to be split into partial payments to avoid tx failures. To guarantee atomicity, *i.e.,* the recipient cannot claim the payment until it receives all partial payments by embedding the AMP protocol [10] to BAR using secret sharing. Specifically, the sender derivates the based preimage R as partial preimages $r_1, \ldots, r_i, \ldots, r_n$. The hash h_i of r_i is associated with the i^{th} partial payment. The recipient reconstructs the base preimage to claim the payment only after receiving all partial preimages.

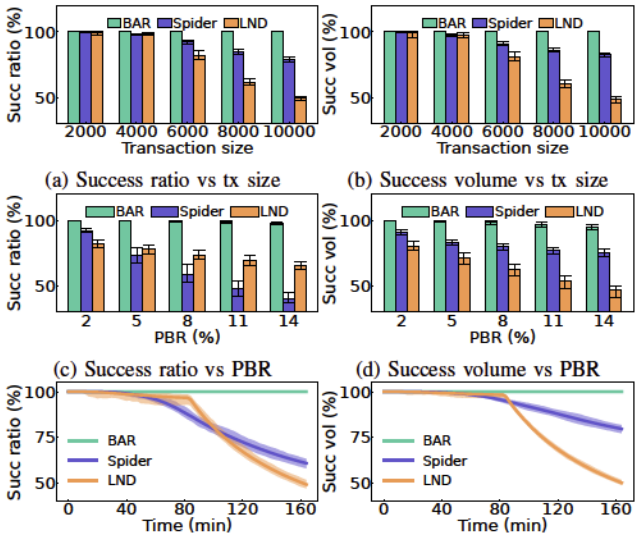
D. Payment Forwarding

After the payment flow construction and the modified HTLC establishment processes, a sender can now forward the payment to its recipient via the HTLCs as shown in Fig. 4. Two edge-joint payment paths have been constructed in the previous stage, where A and C are the senders, and C and D are the respective recipients. Since E requires the preimage of H_1 from F to release 1 on the blue payment path, and F requires the preimage of H_2 from E to release 1 on the orange payment path, E and F can reveal and swap their preimages through the fair exchange. Thus, the balance-aware routing is more efficient with funds being locked only once.

V. Performance Evaluation

A. Environment Setup

We crawled a snapshot topology of the LN [11] on Oct 10, 2020. The network consists of 5,622 nodes and 32,814 channels. The largest connected component that contains the top 0. 4% channels ranked by capacity is retained. We used a real-world credit card tx dataset [19] from September 2013 in Europe, with a mean of $94, a median of $27, and a maximum of $4205. Then it was pre-processed with a final mean $95, a median $24, and a maximum value $1089. The pair of sender and recipient for each tx is sampled 100 times to average out random noise. We use two metrics for evaluation: success ratio and success volume. The success ratio is the proportion of fulfilled payment requests, and the success volume is the total value of fulfilled payments over all requests. We compare BAR with two algorithms Spider [1] and LND [11]. Spider is the most relevant payment routing algorithm that maintains channel balances during routing and finds 4 edge-disjoint paths for each payment. LND finds a payment path with minimal edges from the sender to the recipient with the timeliness and feasibility satisfied. If both are satisfied, the payment is accepted; otherwise, it is rejected.



(a) Success ratio vs tx size (b) Success volume vs tx size

(c) Success ratio vs PBR (d) Success volume vs PBR

(e) Success ratio vs time (f) Success volume vs time

Fig. 5. Comparison between BAR, Spider and LND

B. Evaluation Results

Fig. 5(a) and Fig. 5(b) show the success ratios and the success volumes achieved by BAR, Spider and LND protocols with different tx size. As more txs are processed through the network, there is an increased likelihood of unbalanced channels, resulting in fewer available channels for payment routing. As tx size increases, BAR still manages larger tx sizes effectively, maintaining success ratios and volumes close to 100%. A possible reason for the decrease for Spider is splitting larger payments into multiple partial payments. If any partial payments fail, the overall payment cannot be fulfilled, leading to a decrease in success ratios and volumes. The decrease in LND can be due to the lack of channel rebalancing while routing. To assess the impact of payment amounts on channel depletion, we introduce a novel metric payment-to-balance ratio (PBR), defined as the ratio of the average payment amount to the average balance. A higher PBR implies a greater risk of channel depletion, while a lower PBR suggests less risk. Fig. 5(c) and Fig. 5(d) show the impact of PBR on the success ratios and the success volumes of the three protocols. The performance gap between BAR and Spider demonstrates the importance of considering channel balance and the feasibility constraint. BAR outperforms Spider and LND on both success ratio and success volume, because Spider forces the payment channels to maintain the original balances and LND does not split payments, nor consider rebalance options. In BAR, channel balances are rebalanced whenever routing is performed. Thus, even though a tx can be split into partial txs, rebalancing channels in BAR results in a lower failure probability for any partial payment.

Fig. 5(e) and Fig. 5(f) show the impact of time on the success ratios and the success volumes for the three protocols. Both the success ratio and volumes of Spider and LND decrease during the second half of the simulation, indicating that channels become unbalanced over time, while BAR still exhibits the highest success ratio and success volume. We also evaluate the message complexity of BAR as shown in Fig. 6. We observe that the average number of messages (megs) increases with the number of nodes. BAR has a growing trend of message complexity similar to that of the distributed Goldberg-Tarjan algorithm [16].

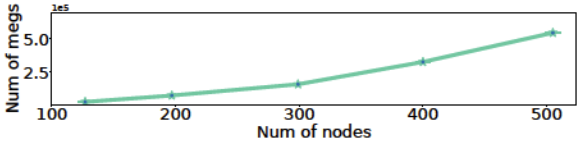


Fig. 6. Message complexity of BAR

VI. Conclusion

In this paper, we investigated the balance-aware high-throughput routing protocol BAR. We suggested four crucial design goals for payment routing in PCNs: channel balance awareness, distributedness, efficiency, and atomicity. BAR achieved channel balance awareness by rebalancing payment channels. We also adapted the original HTLC to BAR to provide transaction efficiency and atomicity. Simulation results demonstrated that BAR outperforms existing approaches in success ratio and success volume.

References

[1] V. Sivaraman, S. B. Venkatakrishnan, K. Ruan, P. Negi, L. Yang, R. Mittal, G. Fanti, and M. Alizadeh, "High throughput cryptocurrency routing in payment channel networks," in *NSDI*, 2020, pp. 777–796.

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

[3] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized business review*, p. 21260, 2008.

[4] "Ethereum project." [Online]. Available: https://www.ethereum.org/

[5] "Raiden network." [Online]. Available: https://raiden.network/

[6] Y. Zhang, D. Yang, and G. Xue, "Cheapay: An optimal algorithm for fee minimization in blockchain-based payment channel networks," in *ICC*. IEEE, 2019, pp. 1–6.

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

[8] L. M. Subramanian, G. Eswaraiah, and R. Vishwanathan, "Rebalancing in acyclic payment networks," in *PST*. IEEE, 2019, pp. 1–5.

[9] G. Xue, A. Chang, X. Lin, R. Yu, and D. Yang, "Max-min hub pricing in payment channel networks," in *GLOBECOM*, 2024.

[10] O. Osuntokun and C. Fromknecht, "Amp: Atomic multi-path payments over lightning." [Online]. Available: https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html

[11] "The Lightning Network." [Online]. Available: https://lightning.network/

[12] W. Ni, P. Chen, L. Chen, P. Cheng, C. J. Zhang, and X. Lin, "Utility-aware payment channel network rebalance," *Proceedings of the VLDB Endowment*, vol. 17, no. 2, pp. 184–196, 2023.

[13] G. Panwar, R. Vishwanathan, G. Torres, and S. Misra, "Sprite: Secure and private routing in payment channel networks," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, 2024, pp. 1878–1894.

[14] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *SIGSAC*. ACM, 2017, pp. 439–453.

[15] R. Yu, G. Xue, V. T. Kilari, D. Yang, and J. Tang, "CoinExpress: A fast payment routing mechanism in blockchain-based payment channel networks," in *ICCCN*. IEEE, 2018, pp. 1–9.

[16] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *JACM*, vol. 35, no. 4, pp. 921–940, 1988.

[17] E. A. Dinic, "Algorithm for solution of a problem of maximum flow in networks with power estimation," in *Soviet Math. Doklady*, vol. 11, 1970, pp. 1277–1280.

[18] C. Rackoff and D. R. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack," in *Annual International Cryptology Conference*. Springer, 1991, pp. 433–444.

[19] M. L. G. ULB, "Credit card fraud detection." [Online]. Available: https://www.kaggle.com/mlg-ulb/creditcardfraud