1.5em 0pt

DEBAL:Distributed Rebalancing in the Payment Channel Network using DRL

***Abstract*—Payment Channel Networks (PCNs) enable scalable, off-chain blockchain transactions, but liquidity imbalances across channels cause frequent transaction failures, hindering adoption. We propose DEBAL, a hierarchical framework to optimize PCN liquidity. Locally, Deep Reinforcement Learning (DRL) assesses node liquidity by monitoring channel balances and transaction patterns, proactively triggering rebalancing. A randomized, fallback-enabled leader election protocol ensures secure, decentralized coordination, resistant to manipulation. Globally, a BalanceAware Graph Neural Network (GNN) drives path selection, optimizing balance uniformity while considering fees and latency. Multi-objective scoring and iterative rebalancing execute partial transfers along validated paths, maintaining stability. Evaluations on real and synthetic PCN topologies demonstrate an 82.1% transaction success rate, a 57.4% improvement in balance uniformity, and a 168% reduction in transaction latency from 8.2 to 3.1 seconds, outperforming state-of-the-art methods with near-linear scalability. Our contributions include a novel DRL-GNN integration for dynamic liquidity management, a privacy-preserving leader election mechanism, and a scalable rebalancing framework, significantly enhancing PCN performance for high-frequency blockchain applications.**

***Index Terms*—Payment Channel Networks, liquidity rebalancing, Deep Reinforcement Learning, Graph Neural Networks, blockchain scalability**

I. INTRODUCTION

Most current mobile payment systems depend on centralized intermediaries, which bring challenges related to trust, security, data privacy, and high operational costs. With its decentralized, trustless structure, blockchain technology has provided a foundation for secure digital transactions and has gained popularity, especially within cryptocurrency applications. However, conventional blockchain networks face inherent transaction speed and throughput limitations, making them less suitable for large-scale applications like mobile payments [**?**], [**?**].

To address these scalability challenges, *Payment Channel Networks* (PCNs) have emerged as a promising solution. PCNs enable peer-to-peer off-chain transactions that bypass the need for costly on-chain operations, allowing users to settle payments quickly and securely. This approach significantly enhances transaction throughput, reduces latency, and lowers transaction costs, making PCNs well-suited for high-frequency microtransactions among the users. Furthermore, PCN allows multi-hop transactions across the network using *Hashed Timelock Contracts* (HTLCs) facilitating payment between the users that doesn't have a direct channel. As a result, PCNs provide a compelling alternative for mobile payment systems.

However, a key limitation preventing the widespread adoption of PCNs is the issue of liquidity imbalance. As transactions accumulate, the distribution of funds across channels often becomes skewed, with some channels depleting while others remain overfunded. This imbalance leads to higher transaction failure rates. One solution to get rid of such depletion is to fund the channel by sending an on-chain transactions to restore liquidity, called as on-chain rebalancing. However, such rebalancing is limited by the limitations on the onchain blockchain payments. To mitigate this problem, effective rebalancing mechanisms are essential, enabling PCN users to redistribute funds along circular paths and restore network equilibrium through off-chain transactions, termed as off-chain rebalancing.

Recent research has explored the use of Deep Reinforcement Learning (DRL) to address the liquidity rebalancing challenge in Payment Channel Networks (PCNs). One notable approach involves utilizing submarine swaps, which allow for the transfer of liquidity between on-chain and off-chain channels, aiming to maximize the profits of relay nodes in the network [**?**]. This approach formulates the rebalancing task as a Markov Decision Process (MDP), where the system's state transitions depend on the actions taken by the nodes. The Soft Actor-Critic (SAC) algorithm, a popular DRL method known for its stability and efficiency in continuous action spaces, is employed to optimize these rebalancing actions. While this technique has shown potential in enhancing transaction success rates by effectively redistributing liquidity, it suffers from scalability issues. The method is primarily effective for nodes with simple network topologies and struggles to generalize to more complex, highly interconnected networks.

Another innovative solution adopts a leader election mechanism to facilitate coordinated rebalancing through circular transactions. This approach leverages a Graph Neural Network (GNN) to model and capture the intricate dependencies between paths and channels within the PCN. By understanding these dependencies, the system can make more informed decisions about which channels to prioritize during rebalancing. However, the reliance on a deterministic leader election process introduces potential vulnerabilities. Furthermore, involved parties in the rebalancing ends up sharing channel balancing to the leader. This way the predictability of the leader selection mechanism makes it susceptible to manipulation, thereby raising significant concerns about privacy and security in the network, while rebalancing. These limitations highlight the need for more robust and scalable solutions that can balance the trade-offs between efficiency, complexity, and security in PCN rebalancing strategies.

In this paper, we propose DEBAL an advanced, distributed

rebalancing mechanism that can be easily adopted for the complex networks, such as Payment Channel Network. Furthermore, it embeds the randomized leader election mechanism, unlike the existing work and the channel balances of individual channels are kept secrete. Inspired by [**?**], DEBAL estimate the time to depletion for each channel based on transaction patterns, allowing for proactive rebalancing actions. Furthermore, Utilizing the GNN model from [**?**], DEBAL compute rebalancing paths that account for dependencies across the network, enhancing the system's ability to manage complex PCN structures. Our reward function is also modified to incorporate pending and predicted transactions, optimizing current and future liquidity. Thus, the actual contributions of individual parties, in the channel, remain concealed from the leader.

We implement a Randomized Leader Election Protocol leveraging cryptographic hashing to enhance privacy and security. This randomized approach safeguards against manipulation of the leader election process by malicious nodes. To ensure transparency and prevent duplicate fraudulent submissions, candid submissions are recorded on a public ledger.

We have implemented DEBAL and evaluated its performance using a combination of real-world LN topology snapshots and synthetic topologies [**?**], [**?**], tested against the Ripple [**?**] transaction workload. DEBAL consistently outperforms state-of-the-art algorithms, even under high transaction generation rates. For LN topology, DEBAL enhances the success ratio (successful transactions to total transactions issued) from 30.18% to 79.54% and improves the success volume (successful transaction amount to total transaction amount processed) from 3.98% to 29.99%, all while maintaining minimal impact on transaction latency.

In summary, we make the following contributions in this paper:

- We propose a fully distributed, proactive in-place rebalancing mechanism for Payment Channel Networks. Our approach incorporates randomized leader election to enhance fairness and security and leverages Deep Reinforcement Learning (DRL) to predict the time to channel depletion accurately.
- A concrete, real implementation and comprehensive evaluation of DEBAL, with real transaction data on a real topology and empirical evidence that DEBAL outperforms the state-of-the-art routing schemes.

The rest of this paper is organized as follows:

II. Background & Motivation

Despite advancements in Payment Channel Network (PCN) rebalancing, existing approaches face critical challenges in balancing local adaptability, network-wide coordination, and computational scalability:

- **Localized DRL Approaches**: While effective for managing liquidity at the node level, DRL models lack network-wide awareness, often resulting in uncoordinated, reactive rebalancing that risks destabilizing high-traffic subnetworks.
- **Network-Wide GNN Models**: Although GNN-based approaches optimize global rebalancing paths, they are hindered by significant computational overhead and limited adaptability in dynamic environments. Their dependence on static path selection reduces responsiveness to real-time liquidity fluctuations, compromising performance in high-transaction networks.
- **Direct and Indirect Rebalancing Models**: These models offer flexibility but are constrained by static path dependencies and high on-chain costs, limiting their adaptability and cost-efficiency in dynamic settings.
- **Hybrid DRL-GNN Models**: Hybrid models attempt to balance local and global rebalancing but face challenges in coordinating objectives, often resulting in computational inefficiencies and conflicts that hinder optimal network performance.

These limitations motivate us to propose a hierarchical rebalancing framework that integrates local adaptability with network-wide optimization. At the node level, each participant operates a DRL agent to monitor local liquidity and make real-time, proactive rebalancing decisions. At the network level, a leader node, securely elected using a cryptographic randomized protocol, employs a GNN-based model to compute optimal multi-hop paths for global rebalancing.

By combining decentralized, secure leader election with efficient local and global coordination, our approach ensures a scalable, cost-effective, and unified framework that balances individual liquidity requirements with overall network stability, advancing the state of PCN rebalancing.

III. Methodology

This section presents the methodology for a novel hierarchical framework designed to optimize liquidity distribution in Payment Channel Networks (PCNs). PCNs enable scalable, low-cost off-chain transactions in blockchain systems by maintaining bidirectional payment channels between nodes. However, frequent transactions can lead to imbalanced channel states, where funds accumulate disproportionately in one direction, causing transaction failures due to insufficient liquidity. The proposed framework addresses this challenge by rebalancing channel funds to ensure uniform liquidity, maximize transaction success rates, and minimize operational costs.

The methodology integrates four key components organized hierarchically to achieve efficient rebalancing. First, a Deep Reinforcement Learning (DRL)-based node assessment mechanism evaluates local liquidity conditions, enabling nodes to proactively request rebalancing based on channel balances and transaction dynamics. Second, a randomized leader election process selects a coordinating node to manage rebalancing operations, ensuring decentralization and fairness across the network. Third, a balance-aware Graph Neural Network (GNN) calculates optimal rebalancing paths by modeling the network topology and prioritizing paths that maintain balanced liquidity. Finally, a multi-cycle rebalancing strategy executes

fund transfers iteratively, validating each cycle against balance constraints to prevent extreme imbalances. Together, these components form a cohesive framework that leverages advanced machine learning and graph-based techniques to enhance PCN performance.

The methodology is structured to provide a comprehensive and reproducible approach, detailing the problem formulation, graph modeling, GNN and DRL designs, training procedures, assumptions, limitations, and implementation specifics. Mathematical formulations and algorithmic descriptions are provided to ensure technical rigor, with the goal of achieving robust and scalable rebalancing in dynamic PCN environments.

A. Step 1: Node-Level Liquidity Assessment and Rebalancing Request Decision

B. System Model and Problem Formulation

This subsection establishes the system model for a Payment Channel Network (PCN) and formulates the node-level liquidity assessment and rebalancing request decision as a Markov Decision Process (MDP). The formulation integrates state variables, a constrained Time to Depletion (TTD) calculation with balance-ratio checks, and rebalancing trigger criteria to enable adaptive liquidity management, ensuring uniform channel balances and high transaction success rates.

*1) **System Model**:* A PCN is modeled as an undirected graph $G = (V, E)$, where V denotes the set of nodes (users or entities) and E represents bidirectional payment channels. Each channel $c \in E$ between nodes $u, v \in V$ is characterized by:

- **Capacity** C_c: The total funds allocated to the channel, fixed upon creation.
- **Local Balance** l_c^u: Funds available for node u to send to v.
- **Remote Balance** r_c^u: Funds available for v to send to u, where $r_c^u = C_c - l_c^u$.

The balance constraint ensures $l_c^u + l_c^v = C_c$. Each node $v \in V$ is associated with:

- **Outgoing Transaction Rate** T_{out}^v: The rate of funds sent from v, measured in units per hour.
- **Incoming Transaction Rate** T_{in}^v: The rate of funds received by v, in units per hour.
- **Total Outgoing Balance** b_{local}^v: The sum of local balances across all incident channels, defined as:

$$b_{\text{local}}^v = \sum_{c \in \mathcal{C}_v} l_c^v, \tag{1}$$

  where $\mathcal{C}_v \subseteq E$ is the set of channels incident to v.

The state variables for each node v include:

- **Channel Balances**: $\{(l_c^v, r_c^v) \mid c \in \mathcal{C}_v\}$, capturing liquidity in each direction.
- **Transaction Rates**: $T_{\text{out}}^v, T_{\text{in}}^v$, reflecting transaction activity.
- **Total Outgoing Balance**: b_{local}^v, indicating available funds for sending.
- **Time to Depletion (TTD)**: TTD^v, estimating the duration until liquidity depletion.

These variables provide a comprehensive view of the node's liquidity state, enabling precise assessment for rebalancing decisions.

*2) **Problem Formulation**:* The node-level liquidity assessment and rebalancing request decision aim to determine when a node should initiate a rebalancing request to maintain sufficient liquidity and uniform channel balances. The objectives are:

1) **Maximize Transaction Success Rate**: Ensure channels have adequate balances to support outgoing transactions, minimizing failures due to insufficient funds.
2) **Ensure Uniform Liquidity**: Maintain a minimum balance ratio for each channel c:

$$l_c^u \geq \theta C_c \quad \text{and} \quad l_c^v \geq \theta C_c, \tag{2}$$

   where $\theta = 0.2$ is the minimum balance ratio threshold.
3) **Minimize Rebalancing Costs**: Reduce the frequency and cost of rebalancing operations, where costs are proportional to the transferred amount and channel fee rate f_c.

The constraints include:

- **Channel Capacity**: Fund transfers must satisfy $0 \leq l_c^u, l_c^v \leq C_c$.
- **Skewness Limit**: Post-rebalancing skewness, defined as:

$$\text{Skewness}(c) = \frac{|l_c^u - l_c^v|}{C_c}, \tag{3}$$

  must satisfy $\text{Skewness}(c) \leq \sigma$, where $\sigma = 0.8$.
- **Circular Rebalancing**: Rebalancing operations use circular paths to redistribute funds off-chain.

To achieve these objectives, the decision process is formulated as an MDP for each node v, defined by the tuple $(S_v, A_v, P_v, R_v, \gamma)$.

*3) **Markov Decision Process Formulation**:* The MDP models the node's decision to request rebalancing, optimizing long-term liquidity and balance uniformity. The components are:

- **State Space** S_v: The state $s_v \in S_v$ encapsulates the node's liquidity and transaction dynamics, defined as:

$$s_v = \left(\{l_c^v, r_c^v \mid c \in \mathcal{C}_v\}, T_{\text{out}}^v, T_{\text{in}}^v, \text{TTD}^v\right), \tag{4}$$

  where:
  - $\{l_c^v, r_c^v \mid c \in \mathcal{C}_v\}$: Local and remote balances for all incident channels, providing a detailed view of liquidity distribution.
  - $T_{\text{out}}^v, T_{\text{in}}^v$: Outgoing and incoming transaction rates, capturing the node's transaction activity.
  - TTD^v: Time to Depletion, calculated as:

$$\text{TTD}^v = \frac{b_{\text{local}}^v}{\max(T_{\text{out}}^v - T_{\text{in}}^v, \epsilon)}, \tag{5}$$

    where $\epsilon = 0.001$ prevents division by zero, and b_{local}^v is given by (1). TTD estimates the duration until

the node's outgoing funds are depleted, serving as a critical indicator of liquidity risk.

The state space is continuous, with $s_v \in \mathbb{R}^{2|\mathcal{C}_v|+3}$, assuming a fixed number of channels per node.

- **Action Space** A_v: The action set is binary:

$$a_v \in \{0, 1\}, \tag{6}$$

where $a_v = 1$ indicates a request for rebalancing, prompting the network to initiate a rebalancing cycle, and $a_v = 0$ indicates no request, maintaining the current state.

- **Transition Dynamics** $P_v(s'|s, a)$: The transition probabilities describe how the state evolves based on the action and network dynamics. If $a_v = 0$, the state updates according to transaction flows, adjusting channel balances based on T^v_{out} and T^v_{in}. Specifically, for each channel $c \in \mathcal{C}_v$, the local balance l^v_c decreases by the outgoing transaction rate and increases by the incoming rate, subject to capacity constraints. If $a_v = 1$, a rebalancing request is submitted, and if accepted by the network's leader, a rebalancing cycle adjusts channel balances along a selected circular path, updating l^v_c and r^v_c. The TTD and transaction rates may also change based on network-wide effects. As the framework employs model-free reinforcement learning, the exact probabilities are not modeled explicitly but are learned from experience in a simulated environment.
- **Reward Function** $R_v(s, a, s')$: The reward function encourages actions that enhance liquidity and channel balance uniformity while minimizing unnecessary rebalancing requests. It is defined as:

$$\begin{aligned} R_v(s,a,s') = \; & w_1 \cdot (\text{TTD}^v(s') - \text{TTD}^v(s)) \\ & + w_2 \cdot \left(\frac{1}{|\mathcal{C}_v|} \sum_{c \in \mathcal{C}_v} \left(\min\left(\frac{l^v_c(s')}{C_c}, \frac{r^v_c(s')}{C_c} \right) \right.\right. \\ & \left.\left. - \min\left(\frac{l^v_c(s)}{C_c}, \frac{r^v_c(s)}{C_c} \right) \right) \right) \\ & - w_3 \cdot \mathbb{I}(a = 1), \end{aligned} \tag{7}$$

where:

  - $\text{TTD}^v(s') - \text{TTD}^v(s)$: Improvement in TTD, rewarding actions that extend the node's liquidity lifespan.
  - $\frac{1}{|\mathcal{C}_v|} \sum_{c \in \mathcal{C}_v} \left(\min\left(\frac{l^v_c(s')}{C_c}, \frac{r^v_c(s')}{C_c} \right) - \min\left(\frac{l^v_c(s)}{C_c}, \frac{r^v_c(s)}{C_c} \right) \right)$: Average improvement in balance ratios across incident channels, promoting uniformity.
  - $\mathbb{I}(a = 1)$: Indicator function penalizing rebalancing requests to avoid excessive operations, with a small penalty to balance proactive and conservative actions.
  - Weights: $w_1 = 0.5$, $w_2 = 0.5$, $w_3 = 0.1$, balancing TTD improvement, balance uniformity, and request cost.

- **Discount Factor** γ: The discount factor is set to:

$$\gamma = 0.99, \tag{8}$$

emphasizing long-term rewards to ensure sustained liquidity and balance uniformity over multiple decision epochs.

*4) **Constrained TTD Calculation and Balance-Ratio Checks**:* The TTD calculation is constrained by balance-ratio checks to ensure proactive rebalancing. For each node v, the TTD is computed using (5), reflecting the time until outgoing funds are depleted based on the net outflow $T^v_{\text{out}} - T^v_{\text{in}}$. The calculation aggregates local balances across all incident channels, providing a holistic view of the node's liquidity capacity.

Balance-ratio checks are performed for each channel $c \in \mathcal{C}_v$, evaluating:

$$\min\left(\frac{l^v_c}{C_c}, \frac{r^v_c}{C_c} \right) < \theta, \tag{9}$$

where $\theta = 0.2$. This condition identifies channels where either direction has less than 20% of the channel's capacity, indicating a risk of imbalance that could hinder bidirectional transactions.

The rebalancing trigger criteria within the MDP framework are:

- **TTD Threshold**: A low TTD ($\text{TTD}^v < \tau$, where $\tau = 2$ hours) signals rapid depletion of outgoing funds, increasing the likelihood of a rebalancing request.
- **Balance Ratio Violation**: If any channel satisfies (10), the node is more likely to request rebalancing to restore balance uniformity.

These criteria are embedded in the state space and reward function, enabling the DRL agent to learn a policy that dynamically balances TTD and balance-ratio considerations.

*5) **Rebalancing Request Decision**:* The MDP enables each node to learn an optimal policy $\pi_v(a_v|s_v)$ that maps states to actions, maximizing the expected cumulative discounted reward:

$$J(\pi_v) = \mathbb{E}_{\pi_v} \left[\sum_{t=0}^{\infty} \gamma^t R_v(s_t, a_t, s_{t+1}) \mid s_0 \right]. \tag{10}$$

The policy is learned using the Soft Actor-Critic (SAC) algorithm, which optimizes the policy by maximizing both the expected reward and policy entropy, ensuring robust decision-making in the continuous state space.

The state variables, constrained TTD, and balance-ratio checks provide a detailed representation of the node's liquidity state. The reward function (8) aligns with the objectives by rewarding TTD improvements and balance uniformity while penalizing excessive requests, ensuring efficient and effective rebalancing decisions.

The node-level liquidity assessment process is formalized in the following algorithm, which computes the TTD and checks balance ratios to determine if a rebalancing request is warranted.

Algorithm 1 Constrained Node-Level Liquidity Assessment

function CONSTRAINEDLIQUIDITYASSESSMENT(node v, channels $\mathcal{C}^v_{\text{out}}$)
  $\theta \leftarrow 0.2,\ \tau \leftarrow 2,\ \epsilon \leftarrow 0.001$
  $b^v_{\text{local}} \leftarrow \sum_{c \in \mathcal{C}^v_{\text{out}}} l^v_c$
  net_flow $\leftarrow \max\big(T^v_{\text{out}} - T^v_{\text{in}},\ \epsilon\big)$
  for each $c \in \mathcal{C}^v_{\text{out}}$ **do**
    ratio $\leftarrow \min\big(\frac{l^v_c}{C_c},\ \frac{r^v_c}{C_c}\big)$
    if ratio $< \theta$ **then**
      return True
    end if
  end for
  $\text{ttd} \leftarrow \dfrac{b^v_{\text{local}}}{\text{net\_flow}}$
  if $\text{ttd} < \tau$ **then**
    return True
  end if
  return False
end function

C. Step 2: Decentralized Leader Election for Rebalancing Coordination

The leader election protocol is triggered when one or more nodes issue rebalancing requests, or at predefined time intervals. Upon activation, each node v with an active request computes a secure hash of its identifier concatenated with the current timestamp: $h_v = \text{SHA256}(ID_v \parallel T)$. Nodes broadcast their hashes or a signed vote to the network, enabling a decentralized selection without a single point of failure. The candidate set is then sorted by hash value in ascending order.

The election iterates through the sorted list of candidates until a valid leader is found. Each candidate node v is checked for sufficient liquidity and balanced channel conditions. We require the total outgoing balance of v,

$$b^{\text{out}}_v = \sum_{c \in C^{\text{out}}_v} l_{vc},$$

to exceed a threshold κ (ensuring it can fund a rebalancing transaction). In addition, for each outgoing channel c of v, the balance ratio

$$\min\Big(\frac{l_{vc}}{C_c},\ \frac{r_{vc}}{C_c}\Big)$$

must exceed a threshold θ, guaranteeing that neither side of any channel is nearly depleted. If a candidate fails any constraint, the election proceeds to the next node in the sorted list. This fallback continues until a candidate satisfies all requirements or the list is exhausted. If no valid leader emerges, the election is deferred to the next trigger, avoiding an overloaded coordinator.

Once a leader v_{leader} is elected, it announces its status to the network by broadcasting a signed `LEADER_ANNOUNCE` message containing $ID_{v_{\text{leader}}}$ and the election timestamp. Other nodes verify the announcement and update their local state to recognize the new leader. This announcement serves as a handover signal: the previous leader (if any) finalizes any in-flight rebalancing tasks and relinquishes coordination to the new leader, who then begins the next rebalancing cycle. The announcement ensures a seamless transition without disrupting ongoing or future rebalancing operations.

Algorithm 2 Multi-Path Leader Election

procedure MULTIPATHLEADERELECTION(V, T, κ, θ)
  $S \leftarrow \{v \in V \mid \text{RebalancingRequest}(v) = \text{True}\}$
  if S is empty **then**
    return None ▷ no candidates
  end if
  Compute hashes $\{h_v = \text{SHA256}(ID_v \parallel T) : v \in S\}$
  Sort candidates $\{(h_v, v)\}$ in ascending order of h_v
  for each (h_v, v) in sorted list **do**
    $b^{\text{out}}_v \leftarrow \sum_{c \in C^{\text{out}}_v} l_{vc}$
    if $b^{\text{out}}_v < \kappa$ **then**
      continue ▷ insufficient outgoing balance
    end if
    $valid \leftarrow$ **True**
    for each channel $c = (v, u) \in C^{\text{out}}_v$ **do**
      $\rho \leftarrow \min\big(l_{vc}/C_c,\ r_{vc}/C_c\big)$
      if $\rho < \theta$ **then**
        $valid \leftarrow$ **False** ▷ channel near depletion
        break
      end if
    end for
    if $valid$ **then**
      return v ▷ select as leader
    end if
  end for
  return None ▷ no valid leader found
end procedure

Periodic re-election uses the same MULTIPATHLEADERELECTION function at each interval Δt, with a refreshed timestamp to ensure unpredictability in leader selection.

D. Step 3: Global Path Calculation using GNN

With the leader elected, the network-wide path calculation process begins. The leader uses a GNN to compute candidate paths across the PCN, capturing network dependencies and selecting paths that effectively rebalance liquidity with minimal cost. The GNN leverages the graph structure $G = (V, E)$, previously defined with node features (e.g., total outgoing balance, transaction rates) and edge features (e.g., capacity, balance ratio, fee rate), to compute paths that maximize liquidity redistribution while enforcing balance constraints. The architecture integrates message-passing mechanisms with a constraint layer to penalize paths causing channel imbalances, ensuring uniform liquidity distribution.

*1) **BalanceAware GNN Architecture**:* The BalanceAware GNN processes the PCN graph to generate node embeddings and path scores, prioritizing circular paths that maintain balanced channel liquidity. The architecture comprises three components: two Graph Convolutional Network (GCN) layers for

message passing, a constraint layer for balance enforcement, and a path scoring module. The input features are:

- **Node Features**: For node $v \in V$, the initial embedding $h_v^{(0)} = [b_{\text{local}}^v, T_{\text{out}}^v - T_{\text{in}}^v] \in \mathbb{R}^2$, where $b_{\text{local}}^v = \sum_{c \in \mathcal{C}_v} l_c^v$ is the total outgoing balance, and $T_{\text{out}}^v - T_{\text{in}}^v$ is the net transaction flow.
- **Edge Features**: For edge $c = (u, v) \in E$, the feature vector $e_c = [C_c, \min(l_c^u/C_c, r_c^u/C_c), f_c] \in \mathbb{R}^3$, capturing capacity, balance ratio, and fee rate.

The message-passing mechanism proceeds as follows:

- **First GCN Layer**: Aggregates neighbor information to update node embeddings:

$$h_v^{(1)} = \text{ReLU}\left(W_1 \cdot \sum_{u \in \mathcal{N}(v)} \frac{h_u^{(0)} + e_{(u,v)}}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}} + B_1 h_v^{(0)}\right), \tag{11}$$

  where $\mathcal{N}(v)$ is the set of neighbors of v, $W_1 \in \mathbb{R}^{16\times 2}$ and $B_1 \in \mathbb{R}^{16\times 2}$ are learnable weights, and the normalization factor stabilizes training. The output $h_v^{(1)} \in \mathbb{R}^{16}$ captures local graph structure and liquidity information.
- **Second GCN Layer**: Further refines embeddings:

$$h_v^{(2)} = \text{ReLU}\left(W_2 \cdot \sum_{u \in \mathcal{N}(v)} \frac{h_u^{(1)} + e_{(u,v)}}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}} + B_2 h_v^{(1)}\right), \tag{12}$$

  where $W_2 \in \mathbb{R}^{8\times 16}$, $B_2 \in \mathbb{R}^{8\times 16}$, producing $h_v^{(2)} \in \mathbb{R}^8$. This layer enhances the representation by incorporating higher-order neighbor interactions.
- **Constraint Layer**: Computes a per-edge constraint score to penalize paths that violate balance constraints:

$$\sigma_c = \text{Sigmoid}\left(W_3 \cdot [h_u^{(2)}, h_v^{(2)}, e_c] + b_3\right), \tag{13}$$

  where $W_3 \in \mathbb{R}^{1\times(8+8+3)}$, $b_3 \in \mathbb{R}$, and $[h_u^{(2)}, h_v^{(2)}, e_c]$ concatenates the embeddings of nodes u and v and edge features. The output $\sigma_c \in [0, 1]$ quantifies the suitability of edge c for rebalancing, with lower values indicating potential imbalance (e.g., $\min(l_c^u/C_c, r_c^u/C_c) < 0.2$).

The constraint outputs modulate node embeddings to penalize imbalance by scaling the final node embeddings:

$$h_v^{(3)} = h_v^{(2)} \cdot \left(\frac{1}{|\mathcal{N}(v)|} \sum_{c \in \mathcal{C}_v} \sigma_c\right), \tag{14}$$

where $\mathcal{C}_v$ is the set of edges incident to v. This modulation reduces the influence of nodes connected to imbalanced edges, ensuring that paths involving such edges receive lower scores.

The path scoring module evaluates candidate circular paths identified via depth-first search. For a path $p = (c_1, c_2, \ldots, c_k)$, the score is:

$$s_p = \sum_{c_i=(u_i,v_i)\in p} \sigma_{c_i} \cdot \text{MLP}([h_{u_i}^{(3)}, h_{v_i}^{(3)}]), \tag{15}$$

where the MLP (Multi-Layer Perceptron) has two hidden layers (16 and 8 units, ReLU activation) and outputs a scalar. The constraint scores σ_{c_i} ensure that paths violating balance constraints are penalized, while the MLP aggregates node embeddings to prioritize paths with high liquidity potential.

*2) **Implementation Details**:* The GNN is implemented using PyTorch Geometric, with hyperparameters:

- **Learning Rate**: 0.001, optimized via Adam.
- **Dropout Rate**: 0.2, applied after each GCN layer.
- **Layer Dimensions**: Input (2), GCN1 (16), GCN2 (8), MLP (16, 8, 1).
- **Batch Size**: 64, for training on simulated PCN graphs.

The computational complexity is $O(|V| \cdot d_{\text{max}} \cdot F + |E| \cdot F')$, where $|V|$ and $|E|$ are the number of nodes and edges, d_{max} is the maximum node degree, and F and F' are the feature dimensions of GCN and constraint layers, respectively. For a typical PCN with $|V| = 500$, $|E| = 2000$, and $d_{\text{max}} = 10$, forward propagation is efficient on modern GPUs (e.g., NVIDIA A100).

*3) **GNN Path Calculation Algorithm**:* The constrained GNN path calculation process is formalized in Algorithm 3, which details the forward propagation through the GNN and the scoring of candidate paths. The algorithm ensures efficient computation and constraint integration, producing a ranked list of paths for rebalancing.

E. Step 4: Iterative Rebalancing Execution

The leader initiates a multi-cycle rebalancing protocol. At each cycle i, it first generates and ranks candidate paths using the Constrained GNN (Step 3). The election then attempts to execute transfers along the highest-ranked paths while validating constraints. If the highest-scoring path is infeasible (e.g., due to capacity or skew constraints), the protocol proceeds to the next-best path. For each candidate path p, we compute the maximum transferable amount x_p along p and attempt a transfer of amount $A = \min(x_p, A_i)$ in that cycle.

After executing a path, the framework checks for skew violations: if any channel $c \in p$ has

$$\text{Skew}(c) = \left|\frac{(l_u + A) - (r_v - A)}{C_c}\right| > \sigma,$$

or if the minimum balance ratio falls below θ, the transfer is considered too imbalancing. The leader then rolls back to the previous state and reduces $A_i \leftarrow F \cdot A_i$ (with $F < 1$) to allow a smaller partial transfer. This reduction can be retried on the same path or used for the next-best path, enabling partial rebalancing along an otherwise infeasible route. If a transfer passes validation, the network state is updated and checked for overall liquidity improvement (e.g., reduction in maximum channel imbalance beyond ϵ). If improvement exceeds a threshold ϵ, the protocol may terminate early.

This process repeats for up to M cycles or until convergence. Clear stopping criteria include reaching the maximum cycles or finding no path that yields significant improvement. The iterative logic is outlined in Algorithm 6 below.

Hyperparameters used include:

- M: maximum number of rebalancing cycles (e.g., 10).

Algorithm 3 Constrained GNN Path Calculation

function GNNPATHCALCULATION(graph $G = (V, E)$, node$_{features}\{\mathrm{h}_v^{(0)}\}$, edge$_{features}\{\mathrm{e}_c\}$)
  Initialize $h_v^{(1)}, h_v^{(2)}, h_v^{(3)} \leftarrow 0$ for all $v \in V$
  Initialize $\sigma_c \leftarrow 0$ for all $c \in E$
  for each $v \in V$ **do** ▷ First GCN Layer
    $m_v \leftarrow \sum_{u \in \mathcal{N}(v)} \frac{h_u^{(0)} + e_{(u,v)}}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}}$
    $h_v^{(1)} \leftarrow \mathrm{ReLU}(W_1 \cdot m_v + B_1 h_v^{(0)})$
  end for
  for each $v \in V$ **do** ▷ Second GCN Layer
    $m_v \leftarrow \sum_{u \in \mathcal{N}(v)} \frac{h_u^{(1)} + e_{(u,v)}}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}}$
    $h_v^{(2)} \leftarrow \mathrm{ReLU}(W_2 \cdot m_v + B_2 h_v^{(1)})$
  end for
  for each $c = (u, v) \in E$ **do** ▷ Constraint Layer
    $\sigma_c \leftarrow \mathrm{Sigmoid}(W_3 \cdot [h_u^{(2)}, h_v^{(2)}, e_c] + b_3)$
  end for
  for each $v \in V$ **do** ▷ Modulate Embeddings
    $h_v^{(3)} \leftarrow h_v^{(2)} \cdot \left(\frac{1}{|\mathcal{N}(v)|} \sum_{c \in \mathcal{C}_v} \sigma_c \right)$
  end for
  $P \leftarrow \mathrm{DepthFirstSearch}(G)$ ▷ Find circular paths
  scores $\leftarrow \emptyset$
  for each path $p = (c_1, \ldots, c_k) \in P$ **do**
    $s_p \leftarrow 0$
    for each $c_i = (u_i, v_i) \in p$ **do**
      $s_p \leftarrow s_p + \sigma_{c_i} \cdot \mathrm{MLP}([h_{u_i}^{(3)}, h_{v_i}^{(3)}])$
    end for
    scores $\leftarrow$ scores $\cup \{(s_p, p)\}$
  end for
  return sort(scores, key $= \lambda x : x[0]$, descending) ▷ Ranked paths
end function

- F: transfer reduction factor ($0 < F < 1$, e.g., 0.8) used when rolling back a failed transfer.
- σ: skew tolerance threshold (e.g., 0.8) for aborting excessive imbalance.
- θ: minimum balance ratio threshold (e.g., 0.2) used in leader election and validation.
- ϵ: improvement threshold; if net liquidity gain falls below this, the protocol converges.
- λ, μ, ν: weights for skew, fee, and latency penalties in the scoring function (from Step 3).
- K: maximum path length (number of channels) for cycle search (used in Step 3).

F. Step 5: Periodic Leader Re-Election

At each fixed interval Δt (e.g., every 10 minutes or after a set number of transactions), a periodic election is triggered using the same protocol as in Step 2. All nodes with active rebalancing requests recompute their hash $h_v = \mathrm{SHA256}(ID_v \parallel T)$ with the new timestamp, and the MULTIPATHLEADERELECTION function is invoked. The sorted-hash fallback logic ensures that the first node satisfying the capacity constraints becomes the new leader.

When a new leader v_{new} is elected, it broadcasts a signed `LEADER_ANNOUNCE` message to the network. The previous leader (if different) finalizes its current rebalancing cycle, transfers any necessary state, and then relinquishes control. This coordination ensures no cycle is left incomplete or lost. After verification of the announcement, v_{new} assumes coordination of subsequent cycles under the updated state. This process repeats at each interval, ensuring a fair rotation of leadership over time.

Algorithm 4 Iterative Rebalancing Execution

procedure ITERATIVEREBALANCE($\mathcal{N}, M, A_1$)
  $original \leftarrow \mathrm{Copy}(\mathcal{N})$
  for $i = 1$ to M **do**
    $paths \leftarrow \mathrm{GNNPathCalculation}(\mathcal{N})$ ▷ ranked by Score
    for each (Score, p) in $paths$ **do**
      $x_p \leftarrow \min_{c \in p} r_c$ ▷ max transferable
      if $x_p \leq 0$ **then**
        continue
      end if
      $A \leftarrow \min(x_p, A_i)$
      if not $\mathrm{ValidatePath}(p, A)$ **then**
        continue ▷ path not feasible
      end if
      Execute transfer of amount A along p
      if $\max_{c \in p} \mathrm{Skew}(c) > \sigma$ or min balance ratio $< \theta$ **then**
        $\mathcal{N} \leftarrow original$ ▷ rollback
        $A_i \leftarrow F \cdot A_i$ ▷ reduce amount
        continue
      end if
      if $\mathrm{LiquidityImprovement}(\mathcal{N}) > \epsilon$ **then**
        return $\mathcal{N}$ ▷ early convergence
      end if
      break ▷ move to next cycle
    end for
  end for
  return $\mathcal{N}$ ▷ return final state
end procedure

REFERENCES

[1] M. Shayan, C. Fung, C. J. M. Yoon, and I. Beschastnikh, "Biscotti: A blockchain system for private and secure federated learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1513–1525, July 2021. [Online]. Available: https://doi.org/10.1109/TPDS.2020.3044223

[2] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[3] K. Elissa, "Title of paper if known," unpublished.

[4] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[5] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[6] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.
[7] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, arXiv:1312.6114. [Online]. Available: https://arxiv.org/abs/1312.6114
[8] S. Liu, "Wi-Fi Energy Detection Testbed (12MTC)," 2023, gitHub repository. [Online]. Available: https://github.com/liustone99/Wi-Fi-Energy-Detection-Testbed-12MTC
[9] "Treatment episode data set: discharges (TEDS-D): concatenated, 2006 to 2009." U.S. Department of Health and Human Services, Substance Abuse and Mental Health Services Administration, Office of Applied Studies, August, 2013, DOI:10.3886/ICPSR30122.v2
[10] K. Eves and J. Valasek, "Adaptive control for singularly perturbed systems examples," Code Ocean, Aug. 2023. [Online]. Available: https://codeocean.com/capsule/4989235/tree