On Increasing Scalability and Liquidation of Lightning Networks for Blockchains

Jie Wu, *Fellow, IEEE* and Suhan Jiang, *Student Member, IEEE*

***Abstract*—The lightning network (LN) is a layer-two solution in Bitcoin for support scalability. LN uses offchain micropayment channels to scale the blockchain's capability to perform instant transactions without a global block confirmation process. However, micropayment scalability in a large LN is still limited by its relatively large searching space for a suitable route. Liquidation for small nodes still remains major challenges for the LN as the amount of transactions along a channel is predetermined by the channel capacity defined by two end nodes of the channel. In this paper, we introduce the notion of supernodes and the corresponding supernodes-based pooling to address these challenges. In order to meet the high adaptivity and low maintenance cost in the dynamic LN where users join and leave, supernodes are constructed locally to avoid global information or label propagation. Each supernode, together with a subset of (non-supernodes) neighbors, forms a supernode-based pool. These pools constitute a partition of the LN. Additionally, supernodes are self-connected. Micropayment scalability is supported through node set reduction as only supernodes are involved in searching and in payment with other supernodes. Liquidation is enhanced through pooling to redistribute funds within a pool to external channels of its supernode. Extensive simulations using LN simulator CLoTH have been conducted to validate the improvement in routing scalability and liquidation of the proposed architecture under different settings.**

***Index Terms*—Bitcoin, blockchain, lightning networks, liquidation, localized algorithms, pooling, scalability, supernodes.**

I. Introduction

Nowdays, the Blockchain technology has been integrated into multiple areas. The primary use of blockchains is as a distributed ledger for cryptocurrencies, most notably bitcoin. However, the blockchain application is limited by the time to mine a block. Such time is called block time. Each network has its own block time, for example, the Bitcoin's block time is around 10 minutes while the Etherum's block is about 13 seconds. There are several proposals to improve blockchain scalability [2], among which, payment channel network (PCN) [3, 4] provides an off-chain solution. Composed of multiple bidirectional payment channels, a PCN allows a transaction to be quickly settled between two parties without submitting it to the blockchain for confirmation, thus acting as a second layer of the original blockchain network.

J. Wu and S. Jiang are with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA, 19122. E-mail:{jiewu and Suhan.Jiang}@temple.edu This paper is extended based on [1].

This research was supported in part by NSF grants CPS 2128378, CNS 2107014, CNS 2150152, CNS 1824440, CNS 1828363, CNS 1757533, and ARO grant W911NF17-1-0378.

(a) Simple path (b) Multiple paths

Fig. 1: A fund transfer of \$4 from u to v via w in simple path (a) and in multiple paths (b).

Lightning networks (LNs) [3] recently emerged as a promising PCN network that addresses the scalability issue of the blockchain and its applications in Bitcoin [5]. In the original blockchain, each transaction has to go through a global block confirmation process. Using smart contracts, *trusted neighbors* (or simply neighbors) in LNs can set up *micropayment channels* (or simply channels) that support instant transactions without block confirmation. Such a transaction can be done via neighbors directly or non-neighbors with a path of microchannels connecting these nodes. In this way, LNs offer more opportunities for fast transactions between nodes. Each LN node with a given amount of funds will split funds to channels to set up bidirectional payment channels with its neighbors. A node can send a payment to another party in LNs as long as this payment does not exceed the funds allocated to this node on the channel. Each channel maintains its balances on two numbers associated with its two end nodes. The summation of these numbers is called the *capability* of the channel.

LNs support non-neighbor transactions by supporting transactions between two nodes that are not neighbors (as u and v in Fig. 1 (a)) or two neighbors with inefficient funds along connecting channels (as u and v in Fig. 1 (b)). In both cases, a transaction involving a transfer of \$4 from u to v has w as the third node in the path. When u sends \$4 to v in Fig. 1 (a), its balance on the channel $\{u, w\}$ is changed to \$1. The balance of the channel associated with w is changed to \$6. The transaction completes through transferring \$4 from w to v. Various extensions of LNs have been proposed that address different performance and/or security aspects since its advent in 2016. For example, we can use multiple flows to implement a transaction, e.g., \$2 from u to v and \$2 more from u to v via w in Fig. 1 (b).

However, LNs and their extensions still face two challenges: micropayment *scalability* in a large LN and *liquidation* for small nodes (i.e., nodes with a small amount of funds with funds assigned to different channels). A small node with the lack of liquidation on one channel will seek help from its other channels through a path connection process. However, when a

transaction amount exceeds the total asset of a node, i.e., the total sum of its adjacent channel values, no process can help out the liquidation issue.

In this paper, we introduce an *LN pooling* method based on a special clustering approach using the notion of *supernodes*. Given a connected graph $G = (V, E)$ with node set V and channel (also known as a link) set E, we assume that E only connects trust neighbors who are willing to join for payment channels. It has been shown in [6] that when the node degree of a random graph is sufficiently large, the resultant graph is connected with a high probability.

Supernode-based pooling is constructed by partitioning G into clusters such that each cluster has one supernode. This supernode connects to all its members (as shown in Fig. 2). Additionally, all supernodes are self-connected. In graph terminology, we require that supernode-induced subgraph $G[S]$ be a connected graph for $S \subset V$ and $V - S \subseteq N(S)$ is in G, where $N(S)$ is the neighbor set of supernode set S. In supernode-based pooling, each supernode pools and manages all funds within the cluster to increase liquidation of the network. As $G[S]$ is an induced subgraph from G, there is no extra maintenance cost. To support path searching (or routing) scalability, we require that a relatively small S is derived to reduce the routing space. In order to meet the high adaptivity and low maintenance cost in the dynamic LN where users join and leave, the selection process for S should be local.

In this paper, we address three challenges: (1) Formation of connected supernodes in LNs should be local, rather than global, to better fit a dynamic environment. (2) Maintenance of supernodes and the corresponding clusters should be lightweight. (3) Effectiveness of the proposed architecture in addressing the issues related to microchannel scalability for simple searching and liquidation for small nodes must be experimentally validated. The following summarizes our contributions:

1) We apply a localized algorithm to determine S without any global information or label propagation by removing nodes and pruning links.
2) We discuss how S can be maintained and updated in LNs, including adding and deleting nodes and/or links.
3) We propose how to further reduce the size of S through a hierarchy of supernodes and other means.
4) We evaluate the performance of the proposed scheme on an LN simulator CLoTH in terms of scalability and liquidation on different network settings.

II. Background

A. Payment channel in LNs

Micropayment channels are the core element of the LN. A channel can be seen as a smart contract between two nodes, allowing them to make multiple payments without the need to commit every payment to the blockchain. In Fig. 1, u and w jointly create a payment channel, in which they deposit funds, *e.g.* u deposits \$5 and w deposits \$2. After this transaction is committed to the blockchain, a channel with a capacity of \$7 (\$5 + \$2) is open between u and w. Thereafter, u and w are able to perform payments back and forth freely by issuing transactions. At any moment, u and w can close the channel and refund the balance each one has in the channel by committing a closing transaction with their final balances to the blockchain. All commitment transactions are not published in the blockchain to avoid the costly validation process among the blockchain network, but they are stored by each end node of the channel. The balance of each node is updated after each successful transaction is executed based on mutual agreement.

B. Payment path in LNs

Payment channels are a suitable choice for any two nodes with long-term and high-frequency mutual transactions. The lifecycle of a payment channel starts with a creating transaction and ends with a closing transaction, that have to be committed to the blockchain. LNs can perform payments between nodes not directly connected. Two nodes can make a transaction as long as they can find a path consisting of multiple payment channels between them where the transaction amount is no larger than the minimum channel balance of the path. The transaction sender is required to reward each intermediate node with a small *routing fee* for routing help. For example, in Fig. 1, if u wants to pay \$4 to v, he could route this transaction through w since the balance requirement can be satisfied along with the path of (u, w, v). In general, routing fees are significantly lower than blockchain transaction fees.

C. LN scalability and liquidation

In the current LN, the network topology together with all channel capacities, is published to allow a routing algorithm to find an existing path between two indirectly-connected nodes. However, to preserve users' privacy, the particular balances of the channel at a given time is set confidential only to these two channel holders, making efficient payment routing hard to fulfill as each path has to be validated for balance even with channel capacity information. Flare [7] uses a routing strategy that optimizes the averaging searching time for a legitimate payment channel. For a large LN, the search and validation process will impact on the scalability of the LN.

Another issue associated with LN is the *liquidation* associated with channels with small capacity and nodes with small total capacity of adjacent channels. Assume that we use single path routing, a transfer of \$4 is not possible from u to v in Fig. 1 (b). *Circular rebalancing* [8] helps liquidation for unbalanced channels and nodes. In Fig. 1 (b), we can first perform a circular rebalancing involving nodes v, w, and u by transferring \$1 from v, w, v and back to v in a counter-clockwise direction. A direct path from u to v via w with a transfer of \$4 is then possible. However, if the transfer amount exceed \$5, which exceeds the capacity of both channels (u, v) and (w, u), no simple path route is possible.

D. Pooling via clustering

We assume that each node has a distinct ID and all nodes are initially colored white in coloring schemes for pooling.

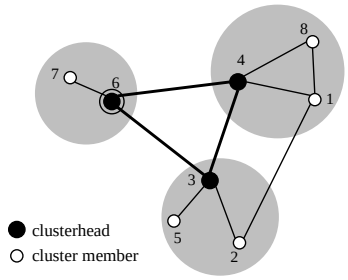
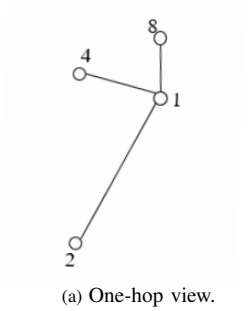
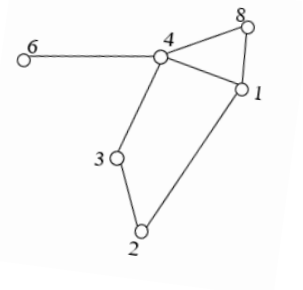


Fig. 2: Supernode-based pooling.



(a) One-hop view.



(b) Two-hop view.

Fig. 3: . One-hop view (a) and two-hop view (b) of node 1 in Fig. 1.

In pseudo local clustering [9], when a white node has the maximum ID among all its white neighbors, it becomes a clusterhead and colors itself black. All white neighbors of a black node join in the cluster and change their colors to gray. This iterative process continues until there are no white nodes left. The black nodes form the set of clusterheads. Each gray node joins one clusterhead. This process is pseudo local because it may require multiple rounds as color labels can propagate sequentially. In another clustering approach [10], a white node selects the node with the maximum ID within its 1-hop neighborhood (including itself) as its dominator. After the white node has chosen its dominator, it colors itself gray if it is not selected as a dominator by itself or by its neighbors; otherwise, it marks itself black. The coloring process continues until no white nodes are left. All of the black nodes become clusterheads. This process is local in one step with no label propagation.

The above approaches are not suitable in LNs as two adjacent clusters may not be connected by their clusterheads, making transactions among pools more complex. The third clustering scheme [11, 12] works as follows: A node marks itself black (i.e., a clusterhead) only when it has two unconnected neighbors. We use k-hop neighborhood information (for a small k, say $k = 2$) to determine the connectivity of two neighbors for each node locally in a decentralized way. Black nodes form a set of connected clusterheads, also called the connected dominating set. In this case, a false negative - as two connected neighbors u and v are falsely identified as unconnected, may occur if after removing link (u, v), the shortest path between u and v is longer than k hops. There are several local pruning methods to further reduce the size of the clusterhead set.

III. Supernode-based Local Pooling

A. Local pooling

Given an undirected connected graph $G = (V, E)$, each node $v \in V$ is identified with a long term public key as its distinct ID. The priority of a node $pri(v)$ is a distinct integer from each distinct ID based on mapping pri, to be used in the symmetric breaking process of local pooling. To avoid cheating at each node, asymmetric and homomophic encryption are used for secure ID priority comparison without decryption during local pooling, together with neighbor watchdogs. We assume that trusted neighbors are willing to exchange their budget and neighbor sets. Each node v maintains a k-hop view $(N_k(v), E_k(v))$. For example, a 2-hop subgraph has its neighbor set $N(v)$ and its 2-hop neighbor set $N(N(v))$, denoted as $N_2(v)$, through neighbor set exchanges with its neighbors. That is, v's 2-hop subgraph is $(N_2(v), E_2(v))$, where $E_2(v) = E \cup (N_1(v) \times N_2(v))$. Note that in Fig. 2, $(3, 4) \in E_2(1)$, but $(4, 6) \notin E_2(2)$, although both 4 and 6 are in $N_2(2)$. Fig. 3 (a) and (b) show a 1-hop view and 2-hop view of node 1 of Fig. 2, respectively.

The following are highlights of our 3-step local pooling scheme for a given graph $G = (V, E)$:

1) **Supernode selection**. Using k-hop subgraph to find a set of supernodes $S \subset V$ locally at each node, such that the induced subgroup $G[S]$ is connected and $V - S \subseteq N(S)$.
2) **Fund pooling**. Each node in $V - S$ joins one pool headed by one of its neighbor supernodes. Each supernode "virtually" pools funds from its members and re-distributes funds to its external channels in $G[S]$.
3) **Routing in the induced subgraph**. All fund transfers among clusters are conducted in $G[S]$.

Supernode set S basically forms a dominating set of G, i.e., each node not in S has a neighbor in S. $G[S]$ is also connected. In the proposed pooling, funds within a cluster are virtually pooled and re-assigned to external channels of the supernode. Note that each supernode and its members still maintain and manage internal channels so that all members will not be over-committed in transactions. Although, each member can still maintain payment channels directly with nodes outside the pool (like channel $\{1, 2\}$ in Fig. 2).

To simplify discussion, we assume that all members allow theirs supernode to bookkeeping their funds by only using the channels connected to the supernode. Members still make their transaction decisions independently. Although members can even perform transactions with others directly using other channels, we focus on using supernodes as bookkeepers of their members and all their funds. The supernode then virtually re-allocates pool funds to its external channels to boost liquidation. In Fig. 2, funds associated with nodes 1, 4, and 8 are redistributed to two external channels $\{4, 6\}$ and $\{4, 3\}$ in the cluster headed by supernode 4, with each internal channel to 4 having the initial budget of each member. Note that each supernode itself is a member so its channel allocation should be managed accordingly as well. Searching (via routing) is

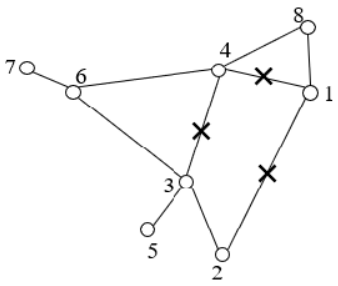


Fig. 4: Replacement paths for channel (1, 2) and subsequent channels (3, 4) and (1, 4).

handled exclusively by supernodes. As S is smaller than V, the scalability issue is alleviated. Note that a supernode may not have any other members – its role is primarily a connector.

B. Supernode selection

We adopt a scheme proposed in [13]:

- **Supernode selection**. All nodes are initially supernodes. A supernode v becomes a non-supernode if any two neighbors of v are connected by (a) a link or (b) a path (constructed from the local 2-hop view of v) such that for each node u (excluding two end nodes) on the path, $pri(u) > pri(v)$.

When two neighbors of v do not meet the conditions (a) and (b), it is simply called *unconnected*, otherwise, they are connected.

The formation of supernodes in a given graph depends on topology, priority distribution, and the amount of local information. For example, with 2-hop view, node 1 is a non-supernode in Fig. 2, because any pair of node 1's neighbors are connected via nodes with a higher priority than node 1. However, if link $\{1, 4\}$ does not exist, node 1 with 2-hop view will be labelled a supernode as node 1 does not "see" link $\{3, 4\}$. In this case, 3-hop view ($k = 3$) is needed to label node 1 as a non-supernode. The supernode selection process is intriguing as non-supernodes are "removed" asynchronously without any centralized coordination. The priority is used so that nodes with the highest priority in the 2-hop view cannot be removed to ensure both coverage and connectivity.

Local pooling can potentially be extended in a couple of ways. To further improve searching (i.e., routing) scalability, we can construct supernodes of supernodes. For example, in Fig. 2, node 6 is the supernode (double-circled) in the supernode set {3, 4, 6}. With this new structure, the routing process becomes multi-level. The benefit of a multi-level process is the ease of routing discovery, but at the cost of with more maintenance overhead.

C. Neighbor set reduction

When we conduct routing, it is assumed that all neighbors of a node in $G[S]$ are involved in the routing process. In reality, only a subset of the neighbors is needed in order to allocate more funds per channel to improve liquidation. How can we reduce a node's neighbor set in $G[S]$ without losing reachability?

Here, we propose a *link pruning* (i.e., neighbor set reduction) process without global connectivity information in $G[S]$. This is analogous to the supernode selection process by replacing nodes with links. We first define the link priority of $\{u, v\}$ (like node priority in supernode selection) based on the reverse lexicographical order of the two end nodes of the link as $pri(\max\{pri(u), pri(v)\}, \min\{pri(u), pri(v)\})$, *e.g.* $pri(3, 2) > pri(3, 1) > pri(2, 1)$.

- **Link pruning**. Link $\{u, v\}$ can be removed if a replacement path (under k-hop view) connecting u and v exists such that all of the links along the path have a higher priority than $\{u, v\}$.

To ensure the end nodes of a link make the same decision, we assume that both nodes of each link exchange its local view before the decision. In Fig. 2, suppose we assume that link pruning happens before supernode selection, thus link {2, 1} can be removed under the 2-hop view of each node because it can be replaced by a path ({2, 3}, {3, 4}, {4,1}). Link {3, 4} itself can be replaced by path ({3, 6}, {6, 4}), and link {4, 1} by path ({4, 8}, {8, 1}).

Given a graph, we can apply local pooling to generate the supernode set S, and then, apply link pruning on $G[S]$; we can also apply the link pruning on G first, followed by local pooling. Note that in the latter case, link pruning reduces the average node degree which will make it easier to reduce the supernode set in the second step.

Fig. 5 (a) shows a 25-node LN that follows the power-law distribution. Figs. 5 (b) and (c) apply supernode set selection of 10 nodes colored red (node reduction), followed by link pruning (link reduction). Figs. 5 (d) and (e) adopt link pruning first, followed by supernode set selection of 7 nodes.

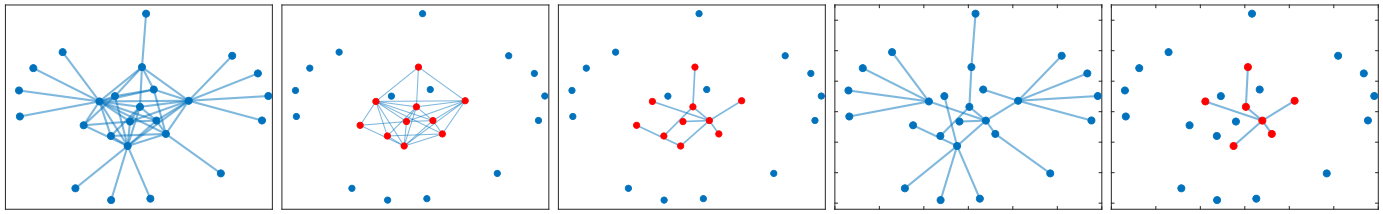
IV. Supernode Properties and Maintenance

A. Properties

The link pruning process is asynchronous at each node and without global coordination. To show that the link pruning is correct, *i.e.*, the resultant graph is still connected, we only need to show that for a link $\{u, v\}$ connecting u and v to be removed, there always exists an "irreplaceable" replacement path connecting u and v after link pruning. First, we give a definition of a *max-min link* $\{x', y'\}$ from replacement paths connecting x and y with link priorities higher than $\{u, v\}$. x and y are node IDs, which are initialized as u and v.

Definition 1. Max-min link for $(x, y, \{u, v\})$**:** *A* min link *in a path is a link with the lowest priority. Assume* $\{P\}$ *is a set of paths connecting* x *and* y *such that all links of each path in the set have a higher priority than* $\{u, v\}$. *A* max-min link $\{x', y'\}$ *in* $\{P\}$ *is a link with the highest priority among all min links in* $\{P\}$.

In the following, we define a recursive process called MAXMIN$(u, v, \{u, v\})$ to construct an irreplaceable replacement path for replacing link $\{u, v\}$.



(a) The original LN topology. (b) Pooling only. (c) Pooling then pruning. (d) Pruning only. (e) Pruning then pooling.

Fig. 5: Two different orders of processes using local pooling and neighbor set reduction on a 25-node LN (red nodes: supernodes).

MAXMIN$(x, y, \{u, v\})$:

1) **If** $x = y$ **then return** $\emptyset$.
2) Determine the max-min link $\{x', y'\}$ for $(u, v, \{u, v\})$.
3) **Return** replacement path (MAXMIN$(x, x', \{u, v\})$, $\{x', y'\}$, MAXMIN$(y', y, \{u, v\})$), assuming x' is closer to x than y' does in the corresponding replacement path.

To illustrate, for link $\{2, 1\}$ in Fig. 2, the max-min link is $\{2, 3\}$ among four replacement paths: $(\{2, 3\}, \{3, 4\}, \{4, 1\})$, $(\{2, 3\}, \{3, 6\}, \{6, 4\}, \{4, 1\})$, $(\{2, 3\}, \{3, 4\}, \{4, 8\}, \{8, 1\})$, and $(\{2, 3\}, \{3, 6\}, \{6, 4\}, \{4, 8\}, \{8, 1\})$. Fig. 3 shows how a replacement path $(\{2, 3\}, \{3, 4\}, \{4, 1\})$ itself is replaced by other replacement paths, both $\{3, 4\}$, and $\{4, 1\}$. 3 is closer to 1 than 2 is in the replacement path. Then, the max-min link for a replacement path connecting node 3 and node 1 is $\{3, 6\}$, selected among four min links $\{4, 1\}$, $\{4, 1\}$, $\{3, 4\}$, and $\{3, 6\}$ from the same four replacement paths above after removing link $\{3, 2\}$, respectively. The max-min link for a placement path connecting node 6 and node 1 is $\{6, 4\}$. Eventually, the irreplaceable replacement path for link $\{2, 1\}$ is $(\{2, 3\}, \{3, 6\}, \{6, 4\}, \{4, 8\}, \{8, 1\})$.

Theorem 1. *The process MAXMIN$(u, v, \{u, v\})$ will complete in a finite number of steps and generate an irreplaceable replacement path for $\{u, v\}$ that cannot be further replaced.*

Proof. We show that all links generated are distinct. Suppose $\{x', y'\}$ is the max-min link among all links in replacement paths connecting u and v. Clearly, $\{x', y'\}$ will not be selected as the max-min link in MAXMIN$(u, x', \{u, v\})$ or MAXIMIN$(y', v, \{u, v\})$ as any path includes no repeated nodes/links. Next, we show that MAXMIN$(u, x', \{u, v\})$ and MAXIMIN$(y', v, \{u, v\})$ have no common links. We assume that MAXMIN$(u, x', \{u, v\})$ is non-empty: $(\{u, u_1\}, \{u_1, u_2\}, ..., \{u_n, x'\})$ and MAXIMIN$(y', v, \{u, v\})$ is non-empty: $(\{y', v_m\}, ..., \{v_2, v_1\}, \{v_1, v\})$. Suppose $\{u_i, u_{i+1}\} = \{v_{j+1}, v_j\}$ (a common link), then $\{\{u, u_1\}, ..., \{u_i, v_j\}, ..., \{v_1, v\}\}$ is a replacement path for $\{u, v\}$. The fact that all links in this path have a higher priority than $\{x', y'\}$ contradicts the fact that $\{x', y'\}$ is the max-min link. Since the recursive call selects a distinct link, the process will complete in a finite number of steps. Based on the max-min link definition, $\{x', y'\}$ cannot be further replaced and the path generated from MAXMIN is irreplaceable. □

Corollary 1. *Given a connected graph, the resultant graph after the link pruning process is still connected.*

Proof. This corollary follows Theorem 1 that each link has an irreplaceable replacement path that cannot be replaced. □

B. Node joining and leaving

When a node v joins or leaves, local pooling can perform a status update of the 2-hop neighborhood of v without propagation. This is because the status of a node (with label supernode or non-supernode) depends only on the connections of this node's 2-hop neighborhood, not the status (*i.e.*, label) of the 2-hop neighborhood. When a node v joins or leaves, v will first inform its neighbors $N(v)$. Each node u in $N(v)$ in turn will inform its neighbors in $N(u)$ through neighbor set exchanges. Finally, each node in $N_2(v)$ (*i.e.*, nodes within 2-hop view of v) will update its status. Once updated, node v, together with the status changes of its 2-hop neighborhood, will be published in the blockchain. When a channel $\{u, v\}$ is added or deleted, the status updates of the two end nodes u and v, together with all nodes in $N(u)$ and $N(v)$, can be done in a similar way.

Note that adding or deleting a node will cause status changes in its 2-hop neighborhood, from supernode to non-supernode or from non-supernode to supernode. For example, adding a new node 9 connecting nodes 3, 4, 6, and 7 in Fig. 2 will change node 6 to a non-supernode, while node 9 has a supernode label. Adding a new node 9 connecting node 2 alone will change node 2 to a supernode, while node 9 is labeled a non-supernode. Similarly, adding or deleting a node may not cause status changes of any nodes within its 2-hop neighborhood. For example, removing node 2 from Fig. 2 will not cause status changes of any node. The following theorems show the general properties when k-hop neighborhood is used in both pooling and pruning.

Theorem 2. *When a node is added to and deleted from an LN, the status change of supernodes and non-supernodes only affects the k-hop neighborhood of the node.*

Proof. As supernodes are constructed using k-hop neighborhood without status propagation, it is easy to see that the status of any node outside k-hop neighborhood of this node will not be changed. □

Similarly, a link can be added and deleted from an LN. The status change is also limited to its k-hop neighborhood.

Theorem 3. *When a link is added to and deleted from an LN, the status change of supernodes and non-supernodes only affects the k-hop neighborhood of two end nodes of the link.*

The proof of this theorem is similar to the one for the node. The correctness follows from the fact that the supernode construction uses k-hop neighborhood information of two end nodes. In Fig. 2 with $k = 2$, deleting (1, 8) will not change the status of any node. However, deleting (1, 4) will change the status of node 1 from non-supernode to supernode. Addition link (2, 5) will not change any node. For instance, neighbors 2 and 5 of node 2 is connected through nodes 3 and 4.

C. Other extensions

Supernode-based pooling can be extended in several ways. The pooling idea can be applied on top of supernodes as they are connected. Fig. 2 shows an example of applying pooling on top of supernodes 3, 4, and 6 to form a new supernode of supernode: 6. On the other hand, this hierarchical routing process also increases the routing complexity and maintenance.

The efficiency of pooling and pruning depends on not only network topology and but also node ID distribution. In the regular pooling and pruning, we use the reverse lexicographical order. To increase efficiency, we can apply different priority orders in a sequential way in multiple rounds. Fig. 5 (a) shows the result of applying two rounds of priority rotation: reverse and then regular lexicographical orders. The supernode set is reduced from 10 nodes (shown in Fig.5 (b)) to 7 nodes.

In applying pooling and pruning, there are two ways to implement in LNs. In the first approach, only topology change is broadcast in LNs and each node run the status change individually. In the second approach, each local node calculates status changes based on local topology changes and then broadcast both topology and status changes are broadcast in LNs.

D. Roles of supernodes

If we define *intra-cluster* transactions as transactions between members from different pools while *inter-cluster* transactions as transactions between members of the same pool. Both types of transactions will be managed (bookkeeping) through supernodes. However, our proposed method does not preclude direct transaction using the direct channel connecting the two parties. In general, the bookkeeping process should be managed by their supernodes, unless funds are partitioned into two in advance: pooled funds and individual funds. Note that setting aside individual funds will weaken the liquidation of fund transfers.

The role of the supernode is primarily organized around bookkeeping of all members' transactions and managing capacity of channels to outside the pool. Each pool member is still in charge of its own transactions. In addition, the number of supernodes still represents a sizable fraction of the total number of nodes in the blockchain networks (e.g., between 35% and 49% based on our simulation shown later) to avoid centralized control.

In our paper, it is assumed that supernodes are selected from trusted neighbors. To ensure that trusted nodes will correctly process all transactions through proper bookkeeping, we can extend our approach based on the notation of k-connected k-dominated supernodes (i.e., dominating nodes) as discussed in [14] . In this case, each member is covered (i.e., managed) by k supernodes, no collusion can happen unless all k are untrusted and collude in a coordinated fashion. In addition, such a system is $k - 1$ fault tolerant and the resultant supernode-induced subnetwork is still connected. This redundancy design is a form of fault-tolerant architecture [15] for robustness support.

The LN network is a layer-two network protocol. The network partition issue has been addressed in [16, 17] in the traditional layer-one blockchain protocol on the sub-chain management. Since the proposed supernode structure ensures network connectivity, it will not introduce any new type of network partition. Therefore, the proposed architecture can resort to the exact same network partition solution provided by the layer-one protocol in the regular blockchain.

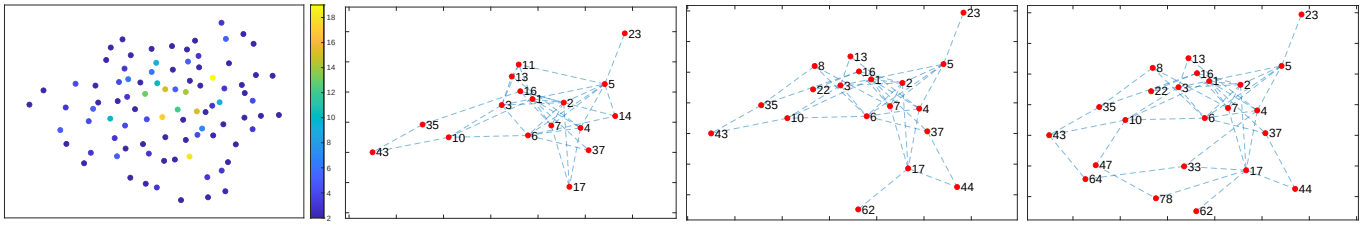
V. Performance Evaluation

Our evaluation includes four parts, starting with a brief description on how to set up our simulation and how to measure the liquidation as well as scalability in LNs (Subsection V.A). Then, we will focus on our proposed methods. First, we will compare our local pooling algorithm with a classic supernode selection algorithm [18] which uses global information (Subsection V.B). The results indicate an obvious tradeoff on the supernode set size and the membership update cost once the LN's topology changes. Second, we show how the proposed pooling series affect the LN's performance (Subsection V.C). We evaluate the LN running under four different methods, *i.e.*, (1) a regular LN without pooling, (2) an LN with pooling only, (3) an LN with pooling then pruning, and (4) an LN with pruning then pooling, and the corresponding results show that our proposed pooling series outperform in terms of liquidation as well as scalability. Last, we conduct experiments on CLoTH, a testbed for HTLC payment network (Subsection V.D). We compare our pooling method with an existing mechanism, Revive, which aims to improve the LN's performance by rebalancing channel funds. The outcome not only proves the feasibility of our pooling method in real time, but also shows a better performance when comparing with Revive.

A. Setup

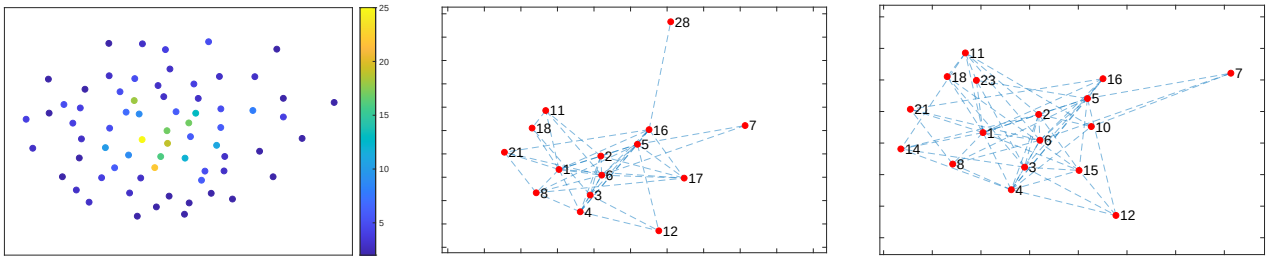
In the simulation, we generate LN topologies using the GraphStream library [19] in Java [20] and implement routing algorithms using the Graph package in Matlab R2021a [21].

1) Topology: Based on [22], LNs can be approximated by the scale-free model where the node degree distribution follows the power law [6]. The network is comprised of a small central clique and a loosely connected periphery. Low degree nodes tend to connect to high degree nodes rather than low degree ones. Thus, we apply the Barabasi-Albert (BA) model [23] based on the preferential attachment rule: nodes are generated one by one by attaching one or more edges to other existing nodes, using a biased random selection that gives more chance to a node with a higher node degree.



(a) The original LN topology. (b) Size of global: 17. (c) Size of supernodes w/ 3-hop: 19. (d) Size of supernodes w/ 2-hop: 23.

Fig. 6: An LN with 90 nodes and 188 edges.



(a) The original LN topology. (b) Size of global: 15. (c) Size of supernodes w/ 3-hop (2-hop): 17.

Fig. 7: An LN with 70 nodes and 197 edges.

2) Channel capacity and balance: Each channel's capacity is set randomly from an interval ranging from $[1000, 1500)$ with probability 50%, $[1500, 2000)$ with probability 35%, and $[2000, 2500)$ with probability 15%. For the balance of the pair of nodes of a given channel at the start of an experiment, we consider two scenarios: (1) randomly balanced, *i.e.*, the two nodes partition the channel capacity in a stochastic manner, and (2) perfectly balanced, *i.e.*, the two nodes of a channel have an identical balance, half of the channel capacity.

3) Transaction parameters: For each transaction, the sender-receiver pair is randomly selected. For the transaction size, we use two settings, (1) homogeneous: each transaction has an identical transfer amount, and (2) heterogeneous: 40% of them are micro, with the transfer amount from $(0, 200]$; 30% of them are small from $(200, 800]$; 20% of them are medium from $(800, 1000]$; and 10% of them are large from $(1000, 1600]$. All selections are random.

4) Metric and benchmark: We use two evaluation metrics for liquidation: (1) *single transaction* success ratio: the number of transactions that can find a reachable path over the number of total transactions, and (2) *transaction flow* success ratio: after sequentially executing all of the transactions of random pairs in the given flow, the number of completed transactions over the number of total transactions. Scalability is measured by the node reduction ratio, *i.e.*, the size of the supernode set over the original set. We only use single path routing under BSF searching, which favors the shortest path to save routing fees for the sender. Routing fees are not included in our evaluation, however, we measure the average path length and node degree.

B. Supernode Selection

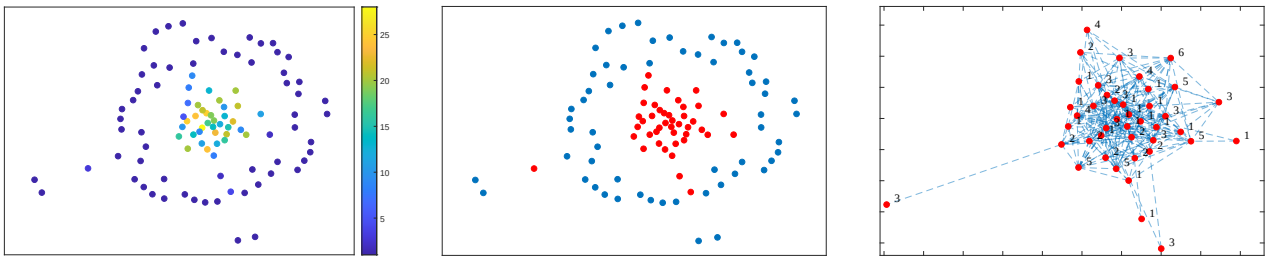
1) Global algorithm: In addition to our local pooling, we will compare it with another pooling method using global information for constructing a small connected supernodes:

D = 8 / 4	Global	Local	
		3-hop	2-hop
# of supernode	17 / 15	19 / 17	23 / 17
rm-edge	23.4 / 22.7	1.82 / 1.47	1.78 / 1.44
rm-node	24.2 / 24	4.80 / 3.67	4.20 / 3.58
add-edge	23.6 / 22.2	1.96 / 1.33	1.90 / 1.29
add-node	23.9 / 22.6	1.11 / 1.06	1.10 / 1.05
add-node-with-edges	24.0 / 22.3	1.45 / 1.78	1.17 / 1.52

TABLE I: Average supernode set size and membership updates

The classic Guha and Khuller's algorithm [18] works as follows: All nodes are initially colored white. The node with the maximum node degree is selected and colored black, all its neighbors are colored gray. An iterative selection process runs until there is no white node left. Select a gray node that has the maximum number of white neighbors, color the selected node black and its white neighbors gray. A modified algorithm selects the gray node u and its neighbor v if they can cause the maximum number of white nodes to change color to gray when both u and v are changed to black. The modified algorithm guarantees an approximation ratio $O(\ln \Delta)$ under any random graph, where Δ is the maximum node degree of the network.

2) Measurement: We consider two ways to measure a supernode selection algorithm in the setting of LNs. The most important measurement is the size of the supernode set, which reflects the LN's scalability. We also consider the cost of membership updates as another essential measurement. Here, we take the network dynamics into consideration, *i.e.*, the joining and leaving of nodes as well as the addition and deletion of channels. We show how the topology changes will affect the supernode set as well as the membership binding between supernodes and non-supernodes. For a given network, we compute the supernode set and the memberships first using different selection algorithms. For each topology change, we recompute the newest supernode set and memberships based



(a) The custom network. (b) Pooling: supernodes marked red. (c) Supernodes: numbers for pool sizes.

Fig. 8: A custom network of 100 nodes and 340 edges (not shown), where node degrees are represented in colors (yellow: high degree, blue: low degree).

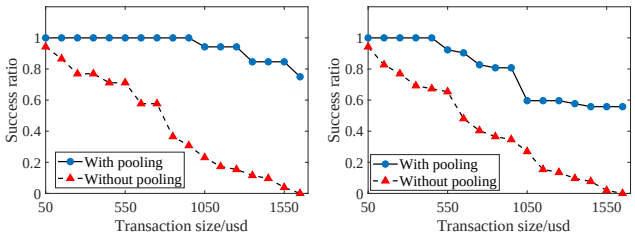
on the new topology. We will record the supernode size as well as the cost of membership updates, *i.e.*, how many nodes change its leaders, including those nodes that currently become supernodes even if they were not selected as non-supernodes in the original network. To change the network topology, we perform 5 operations: (1) rm-edge, *i.e.*, each time an edge will be removed, (2) rm-node, *i.e.*, each time a node will be removed, (3) add-edge, *i.e.*, each time there will be an edge added between two randomly selected nodes, (4) add-node, *i.e.*, each time a new node will be added to the network and will connect to a randomly selected node, and (5) add-node-with-edges, ie each time a new node will be added to the network and will connect to k randomly selected nodes, where k is the average node degree.

3) Networks of different diameters: We generate two different networks based on the BA model. The first network contains 90 nodes and 188 edges, with an average node degree of 4.2. Its diameter is $D = 8$. The second network contains 70 nodes and 197 edges, with an average node degree of 5.6. Its diameter is $D = 4$. To compute the supernode set and the memberships, three methods are compared, the classic global Guha and Khuller's algorithm, our proposed local algorithm with 3-hop neighbor information as well as 2-hop neighbor information. Fig. 6 and Fig. 7 show the detailed information of these two LNs, respectively. The measurement results are given in Table I, respectively.

4) Discussion: Based on Table I, we can conclude that, the more information an algorithm uses, the smaller supernode set it will compute. However, our proposed local algorithm shows a better performance on the cost of membership updates after topology changes, meaning that our pooling method produces a more stable supernode set and supernode-to-non-supernode matching. Meanwhile, we also observe that the network diameter is another factor affecting the performance the pooling methods. Usually, a network with a longer diameter will be affected more after topology changes.

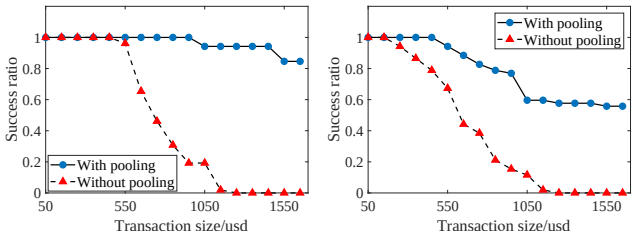
C. Pooling Performance

We generate a custom network based on the BA model with 100 nodes and 340 edges in Fig. 8 (a). Each node is colored based on its node degree, where yellow is the highest and the purple the lowest. After pooling, we obtain a set of 42 supernodes, red nodes in Fig. 8 (b), and the size of each pool



(a) Single transaction success ratio. (b) Transaction flow success ratio.

Fig. 9: Transactions of identical transfer over randomly balanced channels.

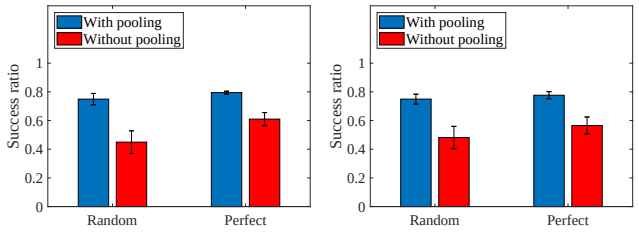


(a) Single transaction success ratio. (b) Transaction flow success ratio.

Fig. 10: Transactions of identical transfer over perfectly balanced channels.

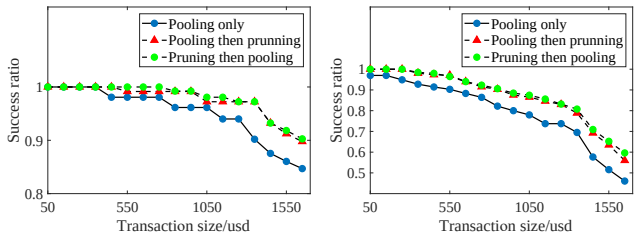
is shown in Fig. 8 (c). Since the supernode set size depends on ID distribution, we perform ID permutation on the network 100 times and obtain the set size varies from 35 to 49 with 43 as the mean. In the following simulation, ID assignment is fixed so we can focus on the other metrics.

1) Pooling only: We conduct experiments on the custom network of Fig. 4 and the network generated by pooling only. We specify an identical transfer amount for each transaction in the homogeneous setting. We generate a flow of 150 transactions with random parings. To see the impact of the transfer amount, we vary its value and re-run the experiment under the fixed network setting, including 150 sender-receiver pairs. Fig. 9 (a) shows the single transaction success ratio under different transfer amounts when all the channels are randomly balanced initially. Obviously, pooling improves the success ratio, especially when the transfer amount grows. We can observe from Fig. 9 (b) that pooling outperforms without pooling for the transaction flow, although its success ratio deteriorates quicker than the single transaction results. Figs. 10 (a) and (b) show success ratios for the single transaction and the transaction flow, respectively, when all of the channels are balanced initially. The performance without pooling stays close to the one with pooling for small transaction sizes. Like in



(a) Single transaction success ratio. (b) Transaction flow success ratio.

Fig. 11: Transactions of random transfer amounts.



(a) Single transaction success. (b) Transaction flow success ratio.

Fig. 12: Transactions of identical transfer over randomly balanced channels.

Fig. 9, pooling helps the success ratio as the transaction size grows, even under an elephant (*i.e.*, large) transaction flow.

Figs. 11 (a) and (b) show the results from the heterogeneous setting on the custom network, under the two different channel balance settings: random and perfect. We generate 17 transaction flows, each containing 150 transactions. On average, our pooling strategy could improve network liquidation: for the single transaction by 66% for random and by 33% for perfect, and for the transaction flow by 60% for random and by 34% for perfect, compared to without pooling.

2) Pooling and pruning: We conduct experiments under three different topologies generated by pooling only, pooling then pruning, and pruning then pooling, respectively, on the custom network. The pooling-only topology contains 255 links connecting 42 supernodes. Based on this topology, we remove some channels through link pruning, and obtain the pooling then pruning topology with 213 links. The pruning then pooling is generated by applying the reversing order and contains 43 supernodes with 226 links. Again, each supernode redistributes its fund equally to all of its external channels.

The performance comparison starts by specifying an identical transfer amount for each transaction in the homogeneous setting. We show the impact caused by the transfer amount in Fig. 12. As before, we generate a flow of 150 transactions with random pairs, then gradually increase the transfer amount. In Figs. 12 (a) and (b), we can observe that the network after pruning has higher success ratios for both cases. This is intuitive, as fewer links lead to more funds assigned to each external channel. Because relatively fewer links are pruned after pooling, the improvement of the success ratio for pruning is less obvious than pooling for the custom network.

In a separation simulation for the case of Fig. 12 (b) but with a 20% variation in the identical transfer size of the transaction flow, the success ratio increases by around 5% compared with Fig. 12 (b). Hence, we conduct simulations of transaction flow

$(\|V\|, \|E\|)$	**Operation**	STSR	TFSR	PL	ND
$(100, 340)$	W/O pooling	0.45	0.48	4.89	6.80
$(42, 255)$	W/ pooling	0.75	0.77	6.35	12.14
$(42, 213)$	Pooling, pruning	0.88	0.83	7.01	10.14
$(43, 226)$	Pruning, pooling	0.87	0.86	7.12	10.51

TABLE II: A comprehensive comparison of methods on the custom network.

Topo$(\|V\|, \|E\|)$	**Operation**	STSR	TFSR	PL	ND
ISP$(42, 66)$	W/O pooling	0.64	0.68	2.8	3.14
ISP$(12, 18)$	W pooling	0.85	0.84	3.2	3
ISP$(12, 15)$	Pooling, pruning	0.94	0.95	3.8	2.5
ISP$(10, 13)$	Pruning, pooling	0.98	1	3.4	2.6
WS$(100, 200)$	W/O pooling	0.52	0.49	4.2	4
WS$(81, 133)$	W pooling	0.61	0.66	6.7	3.28
WS$(81, 108)$	Pooling, pruning	0.69	0.76	7.1	2.67
WS$(82, 117)$	Pruning, pooling	0.67	0.74	6.9	2.85

TABLE III: Summary of performance comparison over ISP and WS topologies.

in the heterogeneous setting with random transfer amounts, a more realistic setting. We use the same set of flows for Fig. 11 and apply them to the following: (1) the custom network, (2) pooling only, (3) pooling then pruning, (4) pooling then pruning with $p = 0.5$, for probabilistic pruning, (5) pruning then pooling, and (6) pruning with $p = 0.5$ then pooling. All external channel capacities are assumed to be randomly balanced after pooling. The corresponding mean results are shown in Table II. STSR, TFSR, PL, ND are short for single transaction success ratio, transaction flow success ratio, average path length, and average node degree, respectively.

Based on Table II, success ratios of both single transaction and transaction flow improve after pooling and pruning. However, the impact of the ordering between pooling and pruning is less obvious for the custom network. Probabilistic pruning offers a desirable trade-off among success ratio, path length, and node degree. Note that the success ratio increases due to the pruning efficiency, but will cause a longer path on average, meaning the sender has to pay more routing fees since each intermediate node should be rewarded, according to [24]. The path length (PL) after pooling represents the hop count between sender and receiver, which includes internal channels. The node degree (ND) after pooling measures only supernodes, which includes external channels only. As the custom network is constructed based on the power-law model, a supernode tends to have a higher node degree, even after removing internal channels connecting pool members.

We also evaluate the algorithms on two popular topologies: an ISP topology [25] and a Watts-Strogatz (WS) topology [26]. The same setting used in Table II for channels and transactions is applied, but generate $1,000$ random transactions for ISP and $5,000$ for WS. We summarize the performance comparison in Table III, which shows the same trend as that of Table II, except for the relatively low ND after pooling. This is because ISP and WS do not follow the power-law distribution, supernodes usually do not have high node degrees as in the BA model, which are further reduced after discounting internal channels.

3) Summary: Our simulation on the custom network shows a node reduction between 51% to 65%, which is a desirable

Success ratio improvement	Custom		ISP		WS	
	STSR	TFSR	STSR	TFSR	STSR	TFSR
Pooling only	66%	60%	33%	24%	17%	35%
Pooling, pruning	95%	73%	47%	40%	32%	55%
Pruning, pooling	93%	79%	53%	47%	29%	51%

TABLE IV: A summary of transaction success ratio improvement.

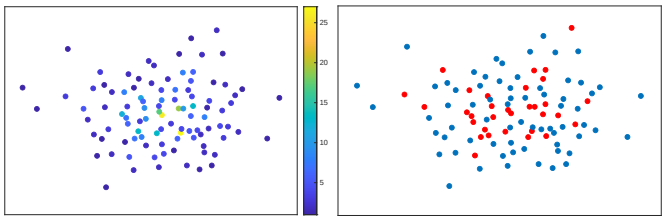
result for searching scalability. Simulations using the heterogeneous setting on three networks show promising results on network liquidation in terms of the success ratio improvement, as is summarized in Table IV. The improvement of pooling then pruning and pruning then pooling varies dependents primarily on the network topology. Note that link pruning comes at the cost of a longer routing path, as any two non-supernodes have to perform transactions through supernodes. Among all proposed algorithms, we find that pooling then pruning and pruning then pooling tend to perform better than pooling only in terms of transaction liquidation, as these two algorithms enlarge the channel capacity among supernodes.

D. Testbed

1) CLoTH testbed: In this part, we use CLoTH [27], a testbed for HTLC payment networks (of which LN is the best working example). It simulates input-defined payments on an input-defined HTLC network and produces performance measures in terms of payment-related statistics (such as the time needed to complete payments and probability of payment failure). CLoTH helps to predict issues and obstacles that might emerge in the development stages of an HTLC payment network and to estimate the effects of an optimisation action before deploying it.

2) Revive rebalancing scheme: Revive enables a set of members in a payment network to shift balances between their payment channels safely. Rather than to enact previously mandatory on-chain channel closing and re-opening, this allows participants to safely revive a channel by real-locating off-chain the funds they have assigned to their payment channels. To be more precise, a rebalancing cycle will be found if each channel along the cycle has the same direction of fund-shifting requirements. The original scheme computes the rebalancing cycles with an objective to maximize the amount of funds moved between channels while setting constraints to maintain the sanity and fairness of the generated transaction set. Here, we simplify the objective by removing the constraints.

3) Generation of network and transaction: We generate a network with 100 nodes and 224 edges, as is shown in Fig. 13. The distribution of the channel capacity and balance strictly follow the rules we mentioned in the setup. Each node's minimal htlc is set as 1000 millisatoshi and its timelock is set as 144ms. To get rid of the external effects, we set the routing fee charged by each routing node as an identical value of 10 millisatoshi, which is far less than the channel balances and the transaction amounts. We also generate a flow of 1200 heterogeneous transactions. We specify a start time for each transaction and ensure that the interval between any two



(a) Testbed network. (b) Pooling: supernodes marked red.

Fig. 13: Testbed network of 100 nodes and 224 edges (not shown).

TFSR	W/O pooling	W/ pooling	REVIVE	
			every 200 tx	every 400 tx
Random	0.718	0.967	0.932	0.921
Perfect	0.788	0.985	0.941	0.927

TABLE V: A summary of transaction flow success ratio under 4 different network mechanisms.

subsequent transactions is with in the range of $[50, 250)$ms. Here, we only apply pooling, which leads to a set of 32 supernodes. We compare the transaction flow success ratio under 4 different network mechanisms, *i.e.*, (1) a regular LN without pooling or rebalancing, (2) an LN with pooling, (3) an LN with rebalancing every 200 transactions, and (4) an LN with rebalancing every 400 transactions. The corresponding results are shown in Table V.

4) Discussion: Obviously, our pooling mechanism still shows good performance when running in the real-time testbed. Although increasing the rebalancing frequency can lead to a higher success ratio for Revive scheme, its performance is still lower than that of our pooling mechanism. Meanwhile, in the experiment, we ignore the rebalancing time which is quite time-consuming in reality. This means those involved channels cannot be used for routing during the rebalancing period, which may lead some transactions to fail due to the path being unavailable.

VI. Related Work

A. Blockchain scalability

The major challenge that hinders the full adoption of blockchain is its scalability. Many efforts have been made to improve the scalability of blockchain. These works can be classified into: on-chain and off-chain. On-chain solutions are also called layer-one solutions, which include increasing the block-size [28], introducing the sharding technique [29–31], using other lightweight consensus such as proof of stake (PoS) [32], or even exploring alternative ledger structures, such as utilizing a directed acyclic graph to organize blocks [33, 34]. Off-chain solutions, *i.e.*, layer-two solutions, focus on constructing a new network that operates on top of an underlying blockchain protocol so that a portion of on-chain transactional burden can be shifted to this adjacent system architecture. Examples of off-chain solutions include SegWit (short for Segregated Witness) [35], side/tree chain [36, 37], rollups [38], and payment channel network (PCN) [3, 4, 39]. Our paper focuses on LN, an exemplification of PCN.

B. Payment channel networks (PCNs)

There are several PCNs [40, 41], and duplex micropayment channels [42] is one of the first PCNs. This scheme can also support incremental channel capability at a relatively low cost [43]. The Raiden network [4] is similar to LN design, but based on the Etherum blockchain using smart contract. Sprites [39] uses merits of both LN and Raiden but a slightly different objective in reducing off-chain channel cost. Several others PCNs apply specific switching and routing techniques. Spider [44] uses packet-switching based routing with payment being split into several micro-payment to alleviate the channel depletion problem. Both SilentWhipser [45] and SpeedyMurmurs [46] utilizer landmark routing where landmarks are the center of the payments. Bolt [47] uses only one intermediary node between the sender and receiver to reduce path length and search complexity.

C. Routing in LNs

The original routing algorithm described in the LN white paper [48] applies a BGP-like protocol, where every node accumulates a global map of the network to perform routing. To support scalability, Flare [7] reduces the size of routing tables maintained by nodes, allowing them to only store neighbors within certain hops. Meanwhile, Flare introduces beacon nodes with a richer network information to supplement a node's local view, while violating the spirit of decentralization. Both SilentWhispers [45] and SpeedyMurmurs [46] propose landmark-based routing schemes. The above routing algorithms fall into static routing, without capturing the payment channel dynamics. Thus, Revive [8], Spider [44], and Flash [49] propose dynamic routing algorithms, leading to a higher throughput and success volume of an LN. [50] discusses tradeoffs between privacy and utility in routing cryptocurrency over payment networks. We focus on reducing the routing space itself, and hence, all of the above extensions can be directly applied.

D. Supernode selection

Supernode selection usually goes through a cluster formation, where a distinct IP address is used to select supernodes. In non-local solutions, an iterative process is applied to identify supernodes (also called clusterheads) such that all other non-clusterhead nodes are directly connected to at least one supernode. Clusterheads generated out of the iterative process usually have some desirable properties such as clusterheads forming a maximal independent set as in [9] that aims to reduce the number of clusters. However, this approach is hard to directly apply to LNs as clusterheads of adjacent clusters are not necessarily directly connected, making transactions among clusterheads more complex.

To better handle network dynamics, local solutions are used to identify self-connected clusterheads using local information and without label propagation as in [10, 11]. Our approach adopted from [13] is local and can also ensure that the derived clusterheads are connected. Additionally, we introduce the neighbor set reduction process to control network density.

VII. Conclusion

This paper introduces a new notion of local pooling to address two challenges in lightning networks: scalability and liquidation. The central idea of local pooling is local clustering with supernodes as clusterheads such that supernodes are self-connected. Supernodes pool all funds in the whole network and they form a smaller network for searching. Local pooling can also be extended by introducing multi-level clustering. Our simulation results show the effectiveness of the local pooling in terms of supporting routing scalability as well as overall network liquidation, especially for transactions that involves high transfer amounts. Our future work will focus on the impact of view (k), pruning probability (p), and routing fees on performance and cost trade-offs. The impact of the diversity of transfer amounts in transaction flows on the network liquidation is still an open research topic. Finally, a performance-centric (such as delay measurement) evaluation in a natural LN system will be part of our future work.

References

[1] J. Wu and S. Jiang, "Local pooling of connected supernodes in lightning networks for blockchains," in *Proc. of IEEE Int'l Conf. on Blockchain*, 2020, pp. 421–427.

[2] K. Croman *et al.*, "On scaling decentralized blockchains," in *Proc. of Int'l Conf. on Financial Cryptography and Data Security*, 2016, pp. 106–125.

[3] "The lightning network 2020." [Online]. Available: https://lightning.network/

[4] "Raiden network." [Online]. Available: http://raiden.network/

[5] "Bitcoin 2020." [Online]. Available: https://bitcoin.org/en/

[6] W. Aiello, F. Chung, and L. Lu, "A random graph model for power law graphs," *Experimental Mathematics*, 2001.

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

[8] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *Proc. of ACM CCS*, 2017.

[9] C.-C. Chiang and M. Gerla, "Routing and multicast in multihop, mobile wireless networks," in *Proc. of IEEE ICUPC*, 1997.

[10] P. Sinha, R. Sivakumar, and V. Bharghavan, "Enhancing ad hoc routing with dynamic virtual infrastructures," in *Proc. of IEEE INFOCOM*, 2001.

[11] J. Wu, "Extended dominating-set-based routing in ad hoc wireless networks with unidirectional links," *IEEE TPDS*, 2002.

[12] D. Simplot-Ryl, I. Stojmenovic, and J. Wu, "Energy-efficient backbone construction, broadcasting, and area coverage in sensor networks." 2005.

[13] J. Wu and F. Dai, "A generic distributed broadcast scheme in ad hoc wireless networks," *IEEE TC*, 2004.

[14] F. Dai and J. Wu, "On constructing k-connected k-dominating set in wireless networks," in *Proc. of IEEE IPDPS*, 2005.

[15] J. Wu and K. Huang, "The balanced hypercube: a cube-based system for fault-tolerant applications," *IEEE Transactions on Computers*, vol. 46, no. 4, pp. 484–490, 1997.

[16] M. Kuperberg, "Towards an analysis of network partitioning prevention for distributed ledgers and blockchains," in *Proc. of IEEE Int'l Conf. on Decentralized Applications and Infrastructures*, 2020, pp. 94–99.

[17] K. Karlsson *et al.*, "Vegvisir: A partition-tolerant blockchain for the internet-of-things," in *Proc. of IEEE ICDCS*, 2018, pp. 1150–1158.

[18] S. Guha and S. Khuller, "Approximation algorithms for connected dominating sets," *Algorithmica*, vol. 20, no. 4, pp. 374–387, 1998.
[19] Y. Pigné, A. Dutot, F. Guinand, and D. Olivier, "Graphstream: A tool for bridging the gap between complex systems and dynamic graphs," *arXiv preprint arXiv:0803.2093*, 2008.
[20] "Eclipse 2019-09." [Online]. Available: https://www.eclipse.org/downloads/packages/release/2019-09
[21] N. MathWorks Inc, "Ma. 2018. matlab r2018a."
[22] I. A. Seres, L. Gulyás, D. A. Nagy, and P. Burcsi, "Topological analysis of bitcoin's lightning network," *arXiv preprint arXiv:1901.04972*, 2019.
[23] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of Modern Physics*, 2002.
[24] "Lightning network daemon." [Online]. Available: https://github.com/lightningnetwork/lnd
[25] http://www.topology zoo.org/, "Isp topology zoo."
[26] "Watts–strogatz model." [Online]. Available: https://en.wikipedia.org/wiki/WattsStrogatz_model
[27] M. Conoscenti, A. Vetrò, J. C. De Martin, and F. Spini, "The CLoTH simulator for htlc payment networks with introductory lightning network performance results," *Information*, vol. 9, no. 9, p. 223, 2018.
[28] Q. Zhou, H. Huang, Z. Zheng, and J. Bian, "Solutions to scalability of blockchain: A survey," *IEEE Access*, vol. 8, pp. 16 440–16 455, 2020.
[29] L. Luu *et al.*, "A secure sharding protocol for open blockchains," in *Proc. of ACM CCS*, 2016, pp. 17–30.
[30] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *Proc. of IEEE S&P*, 2018, pp. 583–598.
[31] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proc. of ACM CCS*, 2018, pp. 931–948.
[32] F. Saleh, "Blockchain without waste: Proof-of-stake," *The Review of Financial Studies*, vol. 34, no. 3, pp. 1156–1190, 2021.
[33] Y. Lewenberg, Y. Sompolinsky, and A. Zohar, "Inclusive block chain protocols," in *Proc. of Int'l Conf. on Financial Cryptography and Data Security*. Springer, 2015, pp. 528–547.
[34] C. Li *et al.*, "A decentralized blockchain with high throughput and fast confirmation," in *Proc. of USENIX ATC*, 2020.
[35] E. Lombrozo, J. Lau, and P. Wuille, "Segregated witness (consensus layer)," *Bitcoin Core Develop. Team, Tech. Rep. BIP*, vol. 141, 2015.
[36] A. Back *et al.*, "Enabling blockchain innovations with pegged sidechains," *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains*, vol. 72, 2014.
[37] A. Dorri and R. Jurdak, "Tree-chain: A fast lightweight consensus algorithm for iot applications," in *Proc. of IEEE LCp*. IEEE, 2020, pp. 369–372.
[38] A. Gluchowski, "Zk rollup: scaling with zero-knowledge proofs," *Matter Labs*, 2019.
[39] A. Miller, I. Bentov, R. Kumaresan, and P. McCorry, "Sprites: Payment channels that go faster than lightning," *CoRR, abs/1702.05812*, 2017.
[40] S. Tripathy and S. K. Mohanty, "Mappcn: Multi-hop anonymous and privacy-preserving payment channel network," in *Proc. of Int'l Conf. on Financial Cryptography and Data Security*, 2020, pp. 481–495.
[41] E. Erdin, M. Cebe, K. Akkaya, E. Bulut, and S. Uluagac, "A scalable private bitcoin payment channel network with privacy guarantees," *Journal of Network and Computer Applications*, vol. 180, p. 103021, 2021.
[42] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Symposium on Self-Stabilizing Systems*. Springer, 2015, pp. 3–18.
[43] C. Burchert, C. Decker, and R. Wattenhofer, "Scalable funding of bitcoin micropayment channel networks," *Royal Society Open Science*, vol. 5, no. 8, p. 180089, 2018.
[44] V. Sivaraman, S. B. Venkatakrishnan, M. Alizadeh, G. Fanti, and P. Viswanath, "Routing cryptocurrency with the spider network," *arXiv preprint arXiv:1809.05088*, 2018.
[45] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Silentwhispers: Enforcing security and privacy in decentralized credit networks." in *NDSS*, 2017.
[46] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," *arXiv preprint arXiv:1709.05748*, 2017.
[47] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *Proc. of ACM S&P*, 2017, pp. 473–489.
[48] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016. [Online]. Available: https://www.bitcoinlightning.com/wp-content/uploads/2018/03/lightning-network-paper.pdf
[49] P. Wang, H. Xu, X. Jin, and T. Wang, "Flash: efficient dynamic routing for offchain networks," *arXiv preprint arXiv:1902.05260*, 2019.
[50] W. Tang, W. Wang, G. Fanti, and S. Oh, "Privacy-utility tradeoffs in routing cryptocurrency over payment channel networks," *Proc. of the ACM on Measurement and Analysis of Computing Systems*, vol. 4, no. 2, pp. 1–39, 2020.

Jie Wu is the Director of the Center for Networked Computing and Laura H. Carnell professor at Temple University. He also serves as the Director of International Affairs at College of Science and Technology. He served as Chair of Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and Associate Vice Provost for International Affairs from the fall of 2015 to the summer of 2017. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Mobile Computing, IEEE Transactions on Service Computing, Journal of Parallel and Distributed Computing, and Journal of Computer Science and Technology. Dr. Wu was general co-chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, ACM MobiHoc 2014, ICPP 2016, IEEE CNS 2016, and ICDCN 2022 as well as program co-chair for IEEE INFOCOM 2011, CCF CNCC 2013, and ICCCN 2021. He was an IEEE Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a Fellow of the IEEE and the AAAS.

Suhan Jiang received her B.S. degree in Computer Science and Engineering from University of Electronic Science and Technology of China, Chengdu, China, in 2016. He is currently a Ph.D. student in the Department of Computer and Information Sciences, Temple University, Philadelphia, Pennsylvania, USA. Her current research focuses on cloud computing and distributing systems, including blockchain techniques.