Local Pooling of Connected Supernodes in Lightning Networks for Blockchains

Jie Wu and Suhan Jiang
Department of Computer and Information Sciences, Temple University

***Abstract*—The lightning network (LN) is a special network in Bitcoin that 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 and liquidation for small nodes still remain major challenges for the LN. 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 without any 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 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, localized algorithms, pooling, supernodes.**

I. Introduction

Lightning networks (LNs) [1] recently emerged as a promising approach that addresses the scalability issue of the blockchain and its applications in Bitcoin [2]. 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.

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, 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 (\$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.

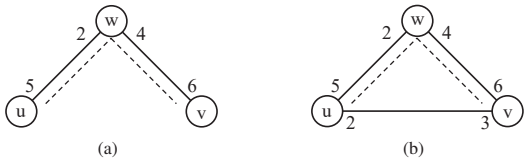


Fig. 1: A fund transfer of \$4 from u to v via w without (a) and with (b) the existence of $\{u, v\}$.

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 [3] 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 $N(S)$ in 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

This research was supported in part by NSF grants CNS 1824440, CNS 1828363, CNS 1757533, CNS 1629746, CNS 1651947, and CNS 1564128.

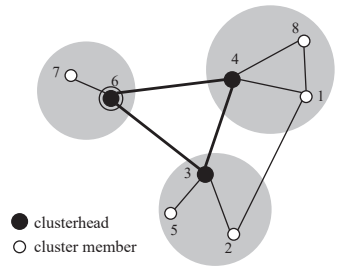


Fig. 2: Supernode-based pooling.

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 summarize 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 propose how to further reduce the size of S through a hierarchy of supernodes. (3) We discuss how S can be maintained and updated in the dynamic LN. (4) We evaluate the performance in terms of scalability and liquidation on different network settings.

II. Background

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.

Payment path in LNs: In the lifecycle of a payment channel, there are two transactions, *i.e.*, a creating transaction and a closing transaction, that have to be committed to the blockchain, thereby causing transaction fees and waiting time. Payment channels are a suitable choice for any two nodes with long-term and high-frequency mutual transactions. Another feature of LNs is the ability to 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.

Pooling via clustering: We assume that each node has a distinct ID and all nodes are initially colored white in coloring schemes for pooling. In pseudo local clustering [4], 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 [5], 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 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 [6] 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 2-hop view, *i.e.*, 2-hop subgraph: 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 $N_2(2)$.

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

1) **Supernode selection**. Using 2-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 "vir-

tually" 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 the supernode to manage their funds by using only channels connected to the supernode. The supernode 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 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 for connection among supernodes.

B. Supernode selection

We adopt a scheme proposed in [7]:

- **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, $pri(u) > pri(v)$.

The formation of supernodes in a given graph depends on topology, priority distribution, and the amount of local information. 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 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 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}).

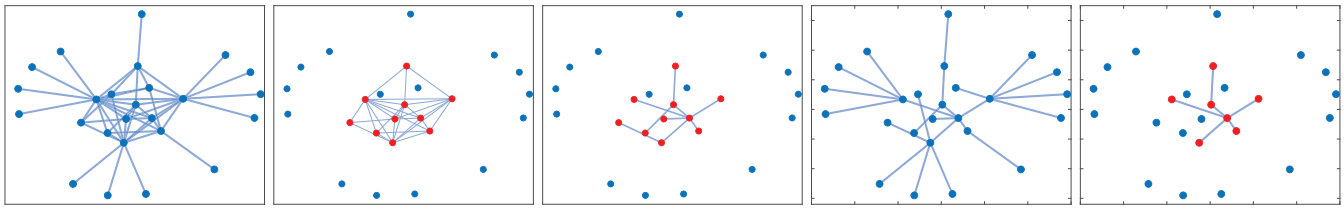
Given a graph, we can apply local pooling to generate the supernode set S, and then, apply link pruning on $G[S]$; we can also apply the link pruning on G first, followed by local pooling. Note that in the latter case, link pruning reduces the average node degree which will make it easier to reduce the supernode set in the second step. To maintain a certain node degree for network fault tolerance, link pruning can be controlled in a probabilistic way, *e.g.* the actual link removal is controlled by a pre-defined probability p on each link pruning.

Fig. 3 (a) shows a 25-node LN that follows the power-law distribution. Figs. 3 (b) and (c) apply supernode set selection of 10 nodes colored red (node reduction), followed by link pruning (link reduction). Figs. 3 (d) and (e) adopt link pruning first, followed by supernode set selection of 7 nodes.

D. 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



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

Fig. 3: Two different orders of processes using local pooling and neighbor set reduction on a 25-node LN (red nodes: supernodes).

$\{x', y'\}$ 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 $\{u, v\}$.

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 $(\{2, 3\}, \{3, 6\}, \{6, 4\}, \{4, 8\}, \{8, 1\})$. 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 nodes $\{4, 1\}$, $\{4, 1\}$, $\{3, 4\}$, 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. □

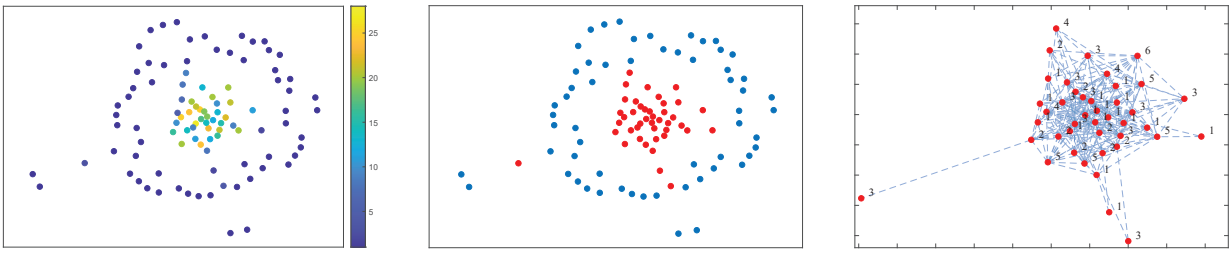
E. Node joining and leaving

When a node v joins or leaves, local pooling can perform 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.

IV. Related Work

Routing in LNs: The original routing algorithm described in the LN white paper [8] applies a BGP-like protocol, where every node accumulates a global map of the network to perform routing. To support scalability, Flare [9] 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 [10] and Speedy-Murmurs [11] propose landmark-based routing schemes. The above routing algorithms fall into static routing, without capturing the payment channel dynamics. Thus, Revive [12],



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

Fig. 4: A custom network of 100 nodes and 340 edges (not shown), where node degrees are represented in colors (yellow: high degree, blue: low degree).

Spider [13], and Flash [14] propose dynamic routing algorithms, leading to a higher throughput and success volume of an LN. We focus on reducing the routing space itself, and hence, all of the above extensions can be directly applied.

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 [4] that aims to reduce the number of clusters. However, this approach is hard to be directly applied 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 [5, 6]. Our approach adopted from [7] is local and can also ensure that the derived clusterheads are connected. Additionally, we introduce the neighbor set reduction process to control network density.

V. Performance Evaluation

A. Setup

In the simulation, we generate LN topologies using the GraphStream library [15] in Java [16] and implement routing algorithms using the Graph package in Matlab R2018a [17].

Topology: Based on [18], LNs can be approximated by the scale-free model where the node degree distribution follows the power law [3]. 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 [19] 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.

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 channel capacity.

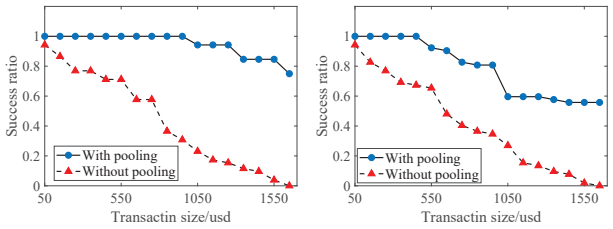
Transaction parameters: For each transaction, the sender-receiver pair is randomly selected. For the transaction size, we use two settings, (1) homogeneous: each transaction has an identical transfer amount, and (2) heterogeneous: 40% of them are micro, with the transfer amount from $(0, 200]$; 30% of 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.

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. Performance

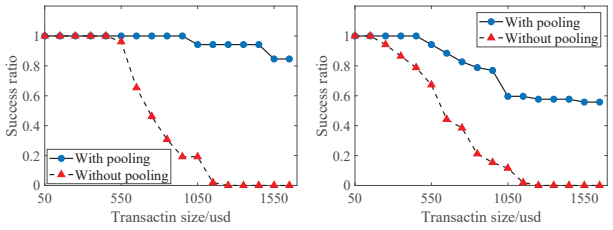
We generate a custom network based on the BA model with 100 nodes and 340 edges in Fig. 4 (a). Each node is colored based on its node degree, where yellow is the highest and the purple the lowest. After pooling, we obtain a set of 42 supernodes, red nodes in Fig. 4 (b), and the size of each pool is shown in Fig. 4 (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 other metrics.

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. 5 (a) shows the single transaction success ratio



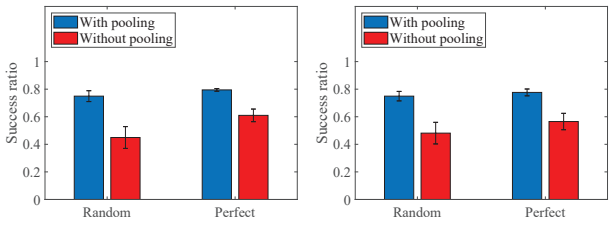
(a) Single transaction success ratio. (b) Transaction flow success ratio.

Fig. 5: Transactions of identical transfer over randomly balanced channels.



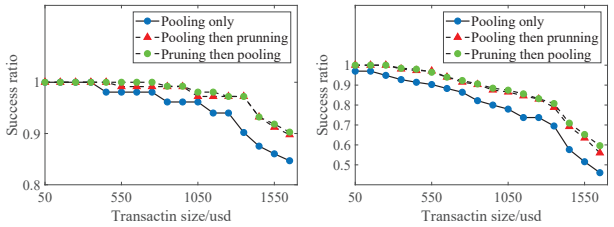
(a) Single transaction success ratio. (b) Transaction flow success ratio.

Fig. 6: Transactions of identical transfer over perfectly balanced channels.



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

Fig. 7: Transactions of random transfer amounts.



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

Fig. 8: Transactions of identical transfer over randomly balanced channels.

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. 5 (b) that pooling outperforms without pooling for the transaction flow, although its success ratio deteriorates quicker than the single transaction results. Figs. 6 (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 Fig. 5, pooling helps the success ratio as the transaction size grows, even under an elephant (*i.e.*, large) transaction flow.

Figs. 7 (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.

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 re-distributes its fund equally to all 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. 8. As before, we generate a flow of 150 transactions with random pairs, then gradually increase the transfer amount. In Figs. 8 (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. 8 (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. 8 (b). Hence, we conduct simulation of transaction flow in the heterogeneous setting with random transfer amounts, a more realistic setting. We use the same set of flows for Fig. 7 and apply 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 I. 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 I, 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 [20]. 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.

$(\|V\|, \|E\|)$	**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
$(42, 241)$	Pooling, 0.5pruning	0.79	0.79	6.77	11.48
$(43, 226)$	Pruning, pooling	0.87	0.86	7.12	10.51
$(43, 244)$	0.5Pruning, pooling	0.81	0.80	6.74	11.35

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

We also evaluate the algorithms on two popular topologies: an ISP topology [21] and a Watts-Strogatz (WS) topology [22]. The same setting used in Table I 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 II, which shows the same trend as that of Table I, 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.

Summary: Our simulation on the custom network shows a node reduction between 51% to 65%, which is a desirable result for searching scalability. Simulations using the heterogeneous setting on three networks show promising results on network liquidation in terms of success ratio improvement, as is summarized in Table III. 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.

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 II: Summary of performance comparison over ISP and WS topologies.

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 III: A summary of transaction success ratio improvement.

VI. 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. Finally, the impact of the diversity of transfer amounts in transaction flows on the network liquidation is still an open research topic.

References

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

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

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

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

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

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

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

[8] 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

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

[10] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Silentwhispers: Enforcing security and privacy in decentralized credit networks." in *Proc. of NDSS*, 2017.

[11] 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.

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

[13] V. Sivaraman, S. B. Venkatakrishnan, M. Alizadeh, G. Fanti, and P. Viswanath, "Routing cryptocurrency with the spider network," *arXiv preprint arXiv:1809.05088*, 2018.

[14] P. Wang, H. Xu, X. Jin, and T. Wang, "Flash: efficient dynamic routing for offchain networks," *arXiv preprint arXiv:1902.05260*, 2019.

[15] 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.

[16] "Eclipse 2019-09." [Online]. Available: https://www.eclipse.org/downloads/packages/release/2019-09

[17] N. MathWorks Inc, "Ma. 2018. matlab r2018a."

[18] I. A. Seres, L. Gulyás, D. A. Nagy, and P. Burcsi, "Topological analysis of bitcoin's lightning network," *arXiv preprint arXiv:1901.04972*, 2019.

[19] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of Modern Physics*, 2002.

[20] "Lightning network daemon." [Online]. Available: https://github.com/lightningnetwork/lnd

[21] http://www.topology zoo.org/, "Isp topology zoo."

[22] "Watts–strogatz model." [Online]. Available: https://en.wikipedia.org/wiki/WattsStrogatz_model