Boosting Payment Channel Network Liquidity with Topology Optimization and Transaction Selection

Krishnendu Chatterjee
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Jan Matyáš Křišťan
Faculty of Information Technology, Czech Technical University, Prague, Czech Republic

Stefan Schmid
TU Berlin, Germany
Weizenbaum Institute, Berlin, Germany

Jakub Svoboda
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Michelle Yeo[1]
National University of Singapore, Singapore

Abstract

Payment channel networks (PCNs) are a promising technology that alleviates blockchain scalability by shifting the transaction load from the blockchain to the PCN. Nevertheless, the network topology has to be carefully designed to maximise the transaction throughput in PCNs. Additionally, users in PCNs also have to make optimal decisions on which transactions to forward and which to reject to prolong the lifetime of their channels. In this work, we consider an input sequence of transactions over p parties. Each transaction consists of a transaction size, source, and target, and can be either accepted or rejected (entailing a cost). The goal is to design a PCN topology among the p cooperating parties, along with the channel capacities, and then output a decision for each transaction in the sequence to minimise the cost of creating and augmenting channels, as well as the cost of rejecting transactions. Our main contribution is an $\mathcal{O}(p)$ approximation algorithm for the problem with p parties. We further show that with some assumptions on the distribution of transactions, we can reduce the approximation ratio to $\mathcal{O}(\sqrt{p})$. We complement our theoretical analysis with an empirical study of our assumptions and approach in the context of the Lightning Network.

2012 ACM Subject Classification Networks → Network algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Approximation algorithms analysis

Keywords and phrases Blockchains, Cryptocurrencies, Payment Channel Networks, Throughput, Optimisation, Graph Algorithms, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.23

Related Version *Full Version*: `https://eprint.iacr.org/2025/1484.pdf`

Funding *Krishnendu Chatterjee*: European Research Council CoG 863818 (ForM-SMArt) and Austrian Science Fund 10.55776/COE12.
Jan Matyáš Křišťan: Czech Science Foundation Grant no. 24-12046S.
Stefan Schmid: German Research Foundation (DFG) project ReNO (SPP 2378) from 2023-2027.
Jakub Svoboda: European Research Council CoG 863818 (ForM-SMArt) and Austrian Science Fund 10.55776/COE12.
Michelle Yeo: MOE-T2EP20122-0014 (Data-Driven Distributed Algorithms).

[1] Corresponding author

1 Introduction

Blockchain scalability is one of the key bottlenecks in the mainstream adoption of cryptocurrencies [9, 12, 24]. For instance, Bitcoin can only process an average of 7 transactions per second which is paltry compared to Visa's 47,000 [26]. This makes it impractical as of yet for the widespread usage of cryptocurrencies in everyday situations.

Some promising solutions to blockchain scalability are layer 2 solutions like payment channel networks (PCNs) [13, 14, 25, 26], which allow users to bypass the blockchain and transact directly with each other. To do so, two users have to first open a payment channel between themselves in an initial funding transaction on the blockchain, thereby locking some funds only to be used in the channel. Thereafter, the two users transact with each other simply by updating the distribution of funds on each end of the channel to reflect the payment amount. As compared to the tedious process of waiting for consensus on the blockchain, these payments can be finalised almost instantaneously since they only involve exchanging signatures between the two parties. As such, with only a constant number of blockchain transactions, any two users can make an unbounded number of off-chain transactions in the payment channel, consequently increasing the overall transaction throughput of the blockchain.

A PCN is a network of payment channels and users, where two users that are not directly connected to each other with a payment channel can still transact with each other in a multi-hop fashion over a path of payment channels. To incentivize intermediary nodes on payment paths to forward payments, PCNs allow these intermediary nodes to charge a small transaction forwarding fee that is typically linear in the transaction amount. Two of the most notable examples of PCNs are Bitcoin's Lightning Network [26] and Ethereum's Raiden [2].

Apart from joining a PCN in order to transact efficiently with other users, a secondary consideration for users in PCNs is to whom they should establish channels and at which capacities, as well as which transactions then to select for forwarding. Aside from the opportunity cost of rejecting to forward transactions and thus missing out on the revenue from transaction fees, users in PCNs also incur a cost whenever channels are created (the on-chain transaction fee) as well as when additional capacity is injected into the channel. As such, although it is tempting to accept to forward all transactions and thus profit from the transaction fees, doing so would quickly deplete the nodes' capacity on their incident channels. A crucial optimisation problem that intermediary nodes in transaction paths would therefore have to consider is deciding which transactions to accept and which to reject, given a minimal injection of capacity into their channel.

In this work, we study the problem of both optimal topology design and transaction selection over a PCN. The input to the problem is a sequence of transactions and a set of p coordinating parties. Each transaction in the sequence is of a certain amount, and is associated with a source and target from the set of parties. The problem then proceeds in two stages: in the first stage, the goal is for the p parties to decide jointly on a PCN topology over themselves. Each created channel in this stage incurs both a fixed channel creation cost as well as a capacity cost that depends on the amount of funds injected into the channel. The second stage involves processing the transactions in the transaction sequence, either accepting or rejecting them (and in so doing incurring a rejection fee) along the created paths in the network. The goal is to create a network and inject enough capacity into each channel such as to maximise the number of transactions accepted while incurring the least amount of cost (see the cost model in Section 2 for more details).

1.1 Related work

To the best of our knowledge, our work is the first to consider the problem of designing an optimal PCN topology for optimising the problem of transaction selection in PCNs. Our work is closely related to the work of [28] which considers a similar problem over a single payment channel and shows that the problem of deciding which transactions to accept or reject over a single channel is already NP-hard. The main open question of [28] is how to extend their proposed algorithm from a single channel to an entire network. Our work addresses this problem by proposing a solution for network design as well as algorithms for optimally accepting or rejecting transactions along paths in a network. Our work is also related to the works of [4, 7] which consider a similar problem of transaction selection but in the online setting and also limited to a single channel. Additionally, the works of [16, 17] also study minimal-cost PCN topologies, but do not consider the complementary problem of transaction selection.

Another related line of research is network creation games on cryptocurrency networks [5, 6, 15]. These works study how a node should connect to an existing cryptocurrency network in a stable and optimal way, optimising the utility of the single node in a unilateral fashion. In contrast, our work proposes a method of creating a network among any number of nodes such as to *jointly minimise* the cost (over all nodes) of network creation and processing transactions.

A crucial assumption in our model is coordination between the interested parties in creating the network during the network creation stage. We stress that this is not an impractical assumption and several works have made similar assumptions [5, 6, 7] where optimal behaviour is analysed in the setting where some or all involved parties cooperate and create certain network topologies together.

Further upfield but also related are works analysing the consensus number of a cryptocurrency [20, 22]. The transactions studied in our work, and in general the sequential nature of layer 2 transactions on PCNs, have consensus number 1, implying that expensive consensus protocols are unnecessary in such settings. Additionally, our work is also related to the classic works on theory of contention management in transactional memory [3, 18, 19].

More generally, our work is related to optimising flows and throughput in typical capacitated communication networks [10, 27]. However, we stress a crucial difference between our problem and problem over traditional communication networks: in traditional communication networks, the capacity is usually independent in the two directions of the channel [21]. In our case, the amount of transactions u sends to v in a channel (u, v) directly affects v's capability to send transactions, as each transaction u sends to v increases v's capacity on (u, v).

1.2 Main challenges and our contributions

We generalise the model of processing transactions over a single channel proposed in [28] to an entire network. The first challenge in shifting from optimising processing transactions from a single channel to a network is deciding on the optimal network topology among the cooperating parties. Here, we show in Theorem 2 in Section 3.2 that the star graph of size p is a 2-approximation of the problem of creating an optimal PCN over p parties.

Once the optimal topology is decided, the next challenge would be to decide on which transactions to accept to forward, given that transactions have to be accepted along the entire payment path. In this setting, we leverage the star topology to generalise the existing single channel algorithm in [28] to get an algorithm for transaction selection optimisation in the star. Our main algorithm in this setting is described in Section 4 and we prove that it is an $\mathcal{O}(p)$ approximation of the weighted transaction selection problem in general graphs.

We describe another algorithm in Section 5 that has an improved approximation ratio of $\mathcal{O}(\sqrt{p})$ (see Theorem 10) for the transaction selection problem in general graphs. To achieve this approximation ratio, we adopt an additional realistic and reasonable assumption on the distribution of transactions in the transaction sequence. Namely, we assume that transactions are clustered, following a stochastic block model.

Finally, we augment our theoretical analysis with a case study of the Lightning Network in Section 6. Using simulations, we find that the network statistics correspond to and thus justify our transaction distribution assumptions.

2 Model

Payment channels. A unique aspect of our problem is that we optimise the selection of weighted transactions over payment channels. Payment channels can be formally modeled as edges that satisfy the following three properties:

1. The capacity or weight of a channel (u, v) is the sum of the initial capacities b_u and b_v injected into the channel by users (or nodes) u and v. The capacities b_u and b_v correspond to the initial amount of funds that users u and v choose to inject into the channel during channel creation.
2. Given a capacitated channel (u, v) with weight w, the weight or capacity can be arbitrarily split between both ends of the channel depending on the number and weight of transactions processed by u and v. That is, b_u and b_v can be arbitrary as long as $b_u + b_v = w$ and $b_u, b_v \geq 0$. For instance, given a payment channel (u, v) with an initial capacity split of b_u and b_v, if u sends a payment of amount x to v, the capacity on u's end of the channel drops to $b_u - x$ and the capacity of v's end increases to $b_v + x$ after processing the transaction. See Figure 1 for more examples of how the capacities on each end of a payment channel can vary in the course of processing transactions.
3. The total capacity of a payment channel is invariant throughout the lifetime of the channel. That is, it is impossible for nodes to add to or remove any part of the capacity in the channel. In particular, if a node is incident to more than one channel in the network, the node cannot transfer part of its capacity in one channel to "top up" the capacity in the other one.

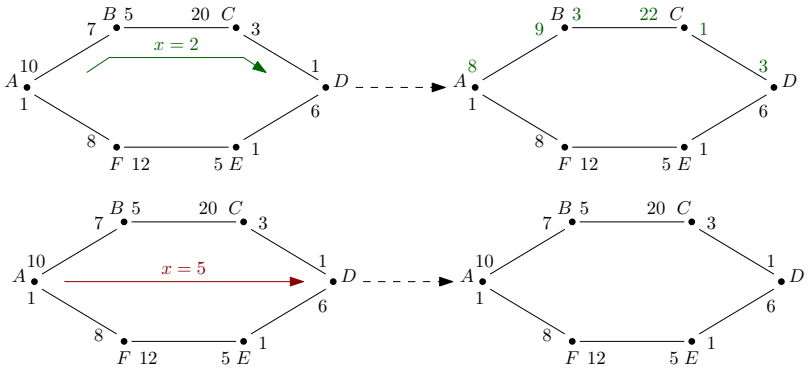

Figure 1 The diagram on the top shows the outcome of processing a transaction of size 2 going from A to D along the path (A, B, C, D) in the graph. The diagram on the bottom shows the outcome of processing a transaction of size 5 from A to D. The transaction has to be rejected as the size of the transaction is larger than the sum of the capacities of C in the channel (C, D) and either A or E in the channel (A, F) or (E, D) respectively.

Payment channel network. We model a PCN as a weighted, undirected graph $G = (V, E)$ where the vertices or nodes of G are users in the PCN and the edges of G are payment channels between nodes. We denote the capacity of a channel $(u, v) \in E$ as $w_{u,v}$, and the capacity of nodes u and v on the channel (u, v) as $b_{u,v}$ and $b_{v,u}$ respectively.

Transaction sequence. Let V denote a set of nodes (parties). We denote a finite ordered sequence of transactions over V by $X_n = ((s_1, t_1, x_1), \ldots, (s_n, t_n, x_n))$, where $s_i, t_i \in V$ represents the sender and target of the ith transaction respectively and $x_i \in \mathbb{R}^+$ represents the weight or size of the ith transaction.

Processing transactions in PCNs. Let $G = (V, E)$ be a connected PCN with some capacity over all of its channels. Let X_n be a sequence of transactions over V. Consider the ith element of the sequence (s_i, t_i, x_i) and suppose the transaction travels over some path $\pi_i = (s_i, \ldots, t_i)$ from the ith sender to the ith target. Let us call the first node along each channel in the path π_i the *forwarding node*. Each forwarding node in the path π_i can choose to do the following to the transaction:

- **Accept transaction.** For a forwarding node u in a channel $(u, v) \in \pi_i$, node u can accept to forward the transaction if their capacity in (u, v) is at least the size of the transaction.
- **Reject transaction.** Node u can also reject the transaction. This could happen if u's capacity is insufficient, or if accepting the transaction would incur a larger cost in the future. For a transaction of weight x, the cost of rejecting the transaction is $f \cdot x + m$ where $f, m \in \mathbb{R}^+$. Apart from incurring some cost, rejecting a transaction does not alter the capacity distribution on both ends of the channel (see the diagram on the bottom of Figure 1 for an example of rejecting a transaction).

We note that node v does not need to take any action nor incur any cost when transactions are going in the direction of u to v along a channel (u, v). Finally, we stress that decisions on each channel in a transaction path need to be the same.

Problem definition. Suppose we have p coordinating parties that want to come together to create a PCN that allows the processing of transactions between themselves. The input to the problem is a transaction sequence over the p coordinating parties. The problem then proceeds in 2 stages. The first stage of the problem is the *network creation stage*, where the p parties have to firstly decide on a PCN topology over themselves. Once the PCN has been created, the second stage of the problem is *transaction selection*, where the network topology created in stage 1 determines the path the transactions travel over. In this stage of the problem, the parties need to decide whether to accept or reject each transaction in the sequence.

Hardness. We know from [28] that the transaction selection problem over a single channel is NP-hard. Our problem includes $p > 2$ parties and we also need to find the optimal network topology over these p parties (for two parties, it is only a channel). This means that our problem is at least as hard as the problem on a single channel, and hence NP-hard as well.

Objective and costs. We consider three types of costs faced by the p parties:

1. **Channel creation costs.** We assume creating each channel incurs a fixed auxiliary cost of $k > 0$.

2. **Capacity costs.** For each channel (u, v) created between parties u and v, the capacity cost of (u, v) is the sum of the capacities $b_{u,v}, b_{v,u}$ injected by u, v into the channel.
3. **Rejection costs.** These are costs incurred during transaction selection when a party rejects a transaction.

The *total cost* incurred by the parties is the sum (over all channels in the PCN and transaction decisions) of channel creation, capacity, and rejection costs. The objective of the p parties is to create a PCN over themselves and output the decisions for each transaction in the sequence such that the total cost is minimised.

3 Accepting all transactions

We begin our investigation of transaction selection over general graphs by first considering the problem a restricted setting where all transactions in the input transaction sequence need to be accepted. In this setting, the total cost consists only of channel creation and capacity costs. We also assume here that we know the optimal PCN topology among the p parties, which we denote by $G = (V, E)$.

3.1 Linear program for a set of channels

We present a linear program (LP) that, for a given set of channels, computes the optimal capacity that the parties need to lock into the channels of the network in order to accept all transactions in the sequence. Indeed, if the channel creation cost k is small or if we want to know a lower bound for our problem, we can assume the optimal graph G is complete. For a node $v \in G$, let $Ne(v)$ denote its neighbours. Let S_v denote the *total balance* of node v in the graph, which is simply the sum of its balances on all its incident channels. For a channel $(u, v) \in E$, we denote the balance of node v (resp. u) after processing the ith transaction in the sequence by $b^i_{v,u}$ (resp. $b^i_{u,v}$), and we also denote the total balance of v after processing the transaction by S^i_v. Let $AC(\mathcal{S}, G)$ denote the optimal capacity cost for the graph G where every transaction in the transaction sequence $\mathcal{S} := ((\pi_1, x_1), \ldots, (\pi_n, x_n))$ over G is accepted. That is, the total capacity injected into the channels of the graph G.

Before describing the LP, we define a *difference function* $\delta_v(s, t)$ between a source s and target t in a transaction path:

$$\delta_v(s,t) = \begin{cases} 1 & v = t \text{ and } v \neq s \\ -1 & v = s \text{ and } v \neq t \\ 0 & \text{otherwise} \end{cases}$$

This function will be used in the LP to maintain the invariant that processing a transaction can only affect the total balances of the source and target nodes in the transaction path, but the total balance of every other node in the graph must not change.

Given G and a transaction sequence $\mathcal{S}$ over G, we construct the following LP.

$$\begin{aligned} \text{minimise} \quad & \sum_{(v,u)\in E} w_{v,u} \\ \forall v, u, i: \quad & w_{v,u} = b^i_{v,u} + b^i_{u,v} \\ \forall v, i: \quad & S^i_v = \sum_{u \in Ne(v)} b_{v,u} \\ \forall v, i, (s_i, t_i, x_i): \quad & S^i_v - \delta_v(s_i, t_i) x_i = S^{i-1}_v \end{aligned}$$

For the sequence of transactions $\mathcal{S}$ and graph G, we denote the solution of the LP with the respective parameters by $LP(\mathcal{S}, G)$. The following lemma (with proof in Section A.1) proves that the solution of the LP is a lower bound on the capacity cost of the problem in the setting where all transactions are to be accepted.

▶ **Lemma 1.** *Given the sequence of transactions $\mathcal{S}$, we have $LP(\mathcal{S}, K_p) \leq AC(\mathcal{S}, G)$.*

3.2 Star as a 2-approximation

Although the capacity cost for the setting where all transactions are accepted is lowest in the complete graph K_p, the complete graph has $\Theta(p^2)$ edges. Thus, depending on the auxiliary channel creation cost k, the cost of creating K_p as well as injecting sufficient capacity into all channels of K_p might be large.

As a useful heuristic, we observe from the proof of Lemma 1 that the capacity cost is related to the diameter of the graph: the smaller the diameter, the lower the cost. Thus, a useful class of graphs to look at for the network creation stage of the problem are graphs that have both a small diameter and a small number of channels. An example of a graph that simultaneously satisfies both of these requirements is a star. In a star, one central node is connected by a channel to all other nodes. Thus, the star has a small diameter of 2, and $p-1$ channels, which is minimal for a connected graph over p nodes. We denote the star on p nodes as $K_{1,p-1}$.

We first observe that for the star, it is easy to compute a lower bound on the amount of capacity required in the setting where all transactions are accepted. This is due to the fact that between any two nodes u, v in a star, there is only one unique path channeling u and v, rendering transactions unsplittable. The capacity on the channel between any node v and the centre of the star c, $w_{v,c}$, has to therefore be larger than $\max_{j,k \leq n} \quad |\sum_{s_i,t_i,x_i | k \leq i \leq j} \delta_v(s_i, t_i) \cdot x_i|$, as this quantity captures the largest difference in the balance between both users of the channel.

We now show in Theorem 2 that we can upper bound the optimal capacity cost on the star. The proof of Theorem 2 is in Section A.2.

▶ **Theorem 2.** *For every graph G over p nodes, and any graph G' with diameter d, the following inequality holds*

$$AC(\mathcal{S}, G') \leq d \cdot AC(\mathcal{S}, G)\,.$$

▶ **Corollary 3.** *For every graph G over p nodes, and graph $K_{1,p-1}$, the following inequality holds*

$$AC(\mathcal{S}, K_{1,p-1}) \leq 2 \cdot AC(\mathcal{S}, G)\,.$$

With these insights, we restrict the network created among the p parties during the network creation stage of the weighted transaction problem to a star in Section 4, and a star of stars (of diameter 4, see Figure 3 for an example) in Section 5.

4 A general setting approximation algorithm

We now present an approximation algorithm for the general setting where transactions can be either accepted or rejected. We first reiterate that from Theorem 2 we can restrict ourselves to the star graph at the network creation stage of the problem, and in doing so incur twice as much capacity cost as the optimal capacity cost. Given that the channel creation cost for

the star topology is fixed at $k(p-1)$, the focus of our algorithm in this section is to come up with capacities that should be injected on both ends of each channel as well as a sequence of transaction decisions so as to minimise the capacity and rejection costs.

4.1 LP for a star

We describe a LP that minimises the capacity cost as well as rejection cost over transactions going between any two leaf nodes in the star. This LP is a relaxation of the general problem, where we allow some transactions to be accepted fractionally. That is, for a transaction of size x_i, the LP does not have to accept the full transaction amount, but can accept some portion y_i such that $0 \leq y_i \leq x_i$. Let us call transactions for which $y_i = x_i$ in the LP solution *fully accepted* transactions.

Let c denote the central node of the star. For a leaf node v and $i \in \mathbb{N}$, we use $C_{v,i}$ and $C'_{v,i}$ to denote the capacity at the channel (v, c) after processing the i-th transaction on the side of v and c respectively. Let C_v denote the total capacity of the channel (v, c). We can now formulate the LP as follows:

$$\begin{aligned}
\text{minimise } & \sum_u C_u + \sum_i f(x_i - y_i) + \frac{m(x_i - y_i)}{x_i} && (1)\\
\forall i : & 0 \leq y_i \leq x_i \\
\forall (v, u, x_i) : & C_v, C_u \geq x_i \\
& C_{v,i} = C_{v,i-1} - y_i \\
& C'_{v,i} = C'_{v,i-1} + y_i \\
& C'_{u,i} = C'_{u,i-1} - y_i \\
& C_{u,i} = C_{u,i-1} + y_i
\end{aligned}$$

▶ **Lemma 4.** *The total cost of the star LP solution is a lower bound on the total cost of the optimal solution.*

Proof. The proof follows from the observation that the optimal integer solution is also a feasible solution for the LP. ◀

▶ Remark 5. The solution of the LP assumes that the capacity cost and the transaction rejection cost are equally large, i.e., we cannot aggressively optimise for one objective while neglecting the other. An implication of this which motivates the design of our algorithms in the following sections is that our algorithms should infrequently reject fully accepted transactions, as these are transactions that do not incur any rejection cost and also change the distribution of the capacity along channels in a more optimal way.

4.2 An $\mathcal{O}(p)$ approximation algorithm

Algorithm for a single channel. We summarize an algorithm from [28] that solves the problem for a single channel, which we will call the *channel algorithm*. The channel algorithm first approximates the capacity to be injected into the channel. Next, the channel algorithm solves a LP (similar to Equation (1)). Given the solution of the LP, we denote by C_u and C_v the minimal capacity needed on each end of a channel (u, v) to mimic the (fractional) transport of transactions as dictated by the LP. In addition to C_u and C_v, the channel algorithm allocates additional capacities on each end of the channel, which we call *reserves*, and denote them by R_u and R_v. These reserves are mainly used to ensure that all transactions that are fully accepted by the LP are also accepted by the channel algorithm (a consequence of Remark 5).

Extending the channel algorithm to a star. We describe our algorithm that extends the channel algorithm over 2 nodes to a star $K_{1,p-1}$ with central node c in Algorithm 1. The key idea behind our algorithm is to first use the LP as described in Equation (1) to fix initial capacities on each end of the channel for every channel in the network (for instance the capacity C_v for v in (v, c) in Figure 2). We also provide each leaf with $p-2$ additional reserves (to process transactions for each other leaf). For each leaf pair u and v, let M denote the smallest change in the capacity as specified by the output of the LP after all interactions between u and v. The additional reserve R_u^v of u for processing transactions of v would then be of size $\frac{\sqrt{3}}{2}M$ (similarly for v). We leave details of Algorithm 1 to Section B.1.

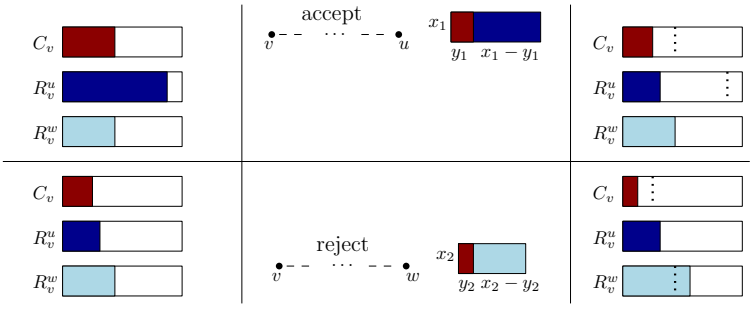

Figure 2 Movement of capacities and reserves from the perspective of node v in a star with centre node c and 2 other leaf nodes u and w. C_v represents the capacity as specified by the output of the LP. R_v^u and R_v^w denote the reserves stored by v to process transactions from u and w respectively. The rightmost column represents the capacities and reserves of v after processing the transaction.

The next theorem shows that Algorithm 1 is an $\mathcal{O}(p)$ approximation of the optimal cost on a star. The proof of Theorem 6 is in Section A.3

▶ **Theorem 6.** *Given the transaction sequence X_t, let $\mathcal{C}$ be the optimal capacity cost and $\mathcal{R}$ the optimal rejection cost on a star. Then Algorithm 1 run for every pair of nodes incurs a rejection cost of at most $(\sqrt{3}+1)\mathcal{R}$ and capacity cost of at most $(1+(p-1)\cdot\sqrt{3})\mathcal{C}$.*

In the next section, we show that if we impose some reasonable assumptions on the distribution of the input transaction sequence in our model, we can design an algorithm with a significantly lower approximation ratio of $\mathcal{O}(\sqrt{p})$.

5 An improved algorithm using the stochastic block model

Stochastic block model. In reality, users typically interact with each other in a structured manner, resulting in spatial and temporal locality. For example, if two nodes that previously transacted with each other are more likely to transact again in the near future. Our model for how nodes interact and send transactions to each other is the stochastic block model [23], in particular, the planted partition model [11]. The stochastic block model is a generative model to create random graphs where nodes are partitioned into clusters (blocks). See Section C.1 for more details on the stochastic block model.

Clustering. We now define the way nodes are partitioned into clusters given an input sequence of transactions. Given an input transaction sequence over a network, we say that the nodes are (m, k, t)-*clustered* if there exists a partition of nodes to m clusters $C_1, C_2, \ldots, C_m$ where each cluster has size at most k and the clusters satisfy the following *clustering conditions*:

Algorithm 1 Modified channel algorithm.

Input: nodes u and v, transaction sequence X_t, solution of LP indexed by i:C_u, C'_u, C_v, C'_v.
Output: decisions to accept or reject for transactions between u and v

$M \leftarrow \min(C_{u,0} + C'_{u,0}, C_{v,0} + C'_{v,0})$
$R_u = \frac{\sqrt{3}}{2}M$, $R_v = \frac{\sqrt{3}}{2}M$
$H \leftarrow$ CreateHeap()
for $i \in [t]$ **do**
  if x_i is transaction from u to v **then**
    if $R_u - (x_i - y_i) \geq \frac{\sqrt{3}-1}{2}M$ **then**
      $x_i \leftarrow$ **Accept**
      $R_u, R_v = R_u - (x_i - y_i), R_v + (x_i - y_i)$
    else if $\frac{y_i}{x_i} < \frac{\sqrt{3}}{\sqrt{3}+1}$ **then**
      $x_i \leftarrow$ **Reject**
      $R_u, R_v = R_u + y_i, R_v - y_i$
    else
      $x_i \leftarrow$ **Accept**
      $R_u, R_v = R_u - (x_i - y_i), R_v + (x_i - y_i)$
      AddToHeap(H, x_i)
      if $R_u < 0$ **then**
        while $R_u < \frac{\sqrt{3}-1}{2}M$ **do**
          $x_j \leftarrow$ PopHeap(H)
          $x_j \leftarrow$ **Reject**
          $R_u, R_v = R_u + x_j, R_v - x_j$
  if $R_u \geq \frac{\sqrt{3}-1}{2}M$ **then**
    EmptyHeap(H)

1. The volume (sum of sizes of transactions) of transactions inside the cluster is t-times the volume of transactions that go outside the cluster. That is,

$$\sum_{(s_j,t_j,x_j)|s_j,t_j \in C_i} x_j \geq t \cdot \sum_{\substack{(s_j,t_j,x_j)|s_j \in C_i \text{ and } t_j \notin C_i \\ \text{or } s_j \notin C_i \text{ and } t_j \in C_i}} x_j$$

2. For every $S \subset C_i$ such that $|S| \leq \frac{C_i}{2}$, the volume of transactions between S and $C_i \setminus S$ is at least $\frac{|S|}{4|C_i|}$ the volume in the cluster C_i. That is,

$$\frac{|S|}{4|C_i|} \sum_{x_j \text{ inside } C_i} x_j \leq \sum_{x_j \text{ between } C_i \setminus S \text{ and } S} x_j$$

3. For every $S \subset C_i$, the ratio of volumes of transactions from S and $C_i \setminus S$ is at least $\frac{1}{3}$. That is,

$$\frac{\sum_{x_j : C_i \setminus S \to S} x_j}{\sum_{x_j : S \to C_i \setminus S} x_j} \geq \frac{1}{3}.$$

These conditions need to be satisfied for every cluster C_i and any contiguous subsequence of transactions involving C_i (i.e., source or target in C_i) with volume at least $C_i \cdot x_{max}$, where x_{max} is the weight of the biggest transaction in the input transaction sequence. In the rest of this section, we assume that given an input sequence of transactions, the nodes can be

$(\sqrt{p}, \sqrt{p}, 24\sqrt{p})$-clustered. Looking ahead, this allows us to design an algorithm that gives us an $\mathcal{O}(\sqrt{p})$ approximation of the optimal cost. In the general case, for an (m, k, t)-clustering, for any $t \geq 24\sqrt{p}$, we have an $\mathcal{O}(m + k)$ approximation (by a similar analysis).

Double star. We define a double star as a graph of diameter $d = 4$ with one central node, some middle nodes, and some leaf nodes. The central and middle nodes create a star and the leaf nodes are connected to only one middle node, forming a cluster (see Figure 3 for an example with 6 clusters). In each cluster, the leaf nodes are connected to the middle node in a star topology. In our setting, given our aforementioned assumption on $(\sqrt{p}, \sqrt{p}, 24\sqrt{p})$-clustering, parties create a double star network topology during the network creation stage, where each cluster represents a group of nodes of size $\sqrt{p}$ that interact frequently with each other.

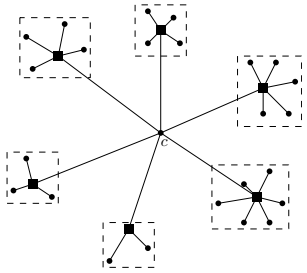

Figure 3 Double star graph with 6 clusters in the dashed boxes. The node labelled c is the centre node, and the middle nodes are the square nodes.

5.1 An $\mathcal{O}(\sqrt{p})$ approximation algorithm

Main ideas. We describe our algorithm that gives an $\mathcal{O}(\sqrt{p})$ approximation of the optimal cost of the problem in Algorithm 2. On a high level, Algorithm 2 consists of two main ideas. The first idea is to again formulate and solve an LP on a double star to get the amount of capacities nodes should inject on their ends of their incident links. This LP is similar to the LP for a star described in Equation (1) and we detail it in Section C.2. From Theorem 2, we know that the capacity cost of the double star is at worst a 4-approximation of the optimal capacity cost.

The second idea is to split the processing of transactions into processing transactions *between* clusters and processing transactions *within* clusters. Here, we use the specific double star topology to ensure that we can apply Algorithm 1 (for processing transactions on a single star) to process transactions between clusters as well as within each cluster. Finally, we use our clustering assumptions to allocate enough reserve capacity for each cluster to ensure that we can always shift the reserve capacities within a cluster from nodes with high to low reserves. This ensures that each node always has sufficient reserve capacity to align the decisions on their transactions (whether to accept or reject) as closely as possible to the ideal decisions as output from the linear program.

Processing transactions between clusters. After running the linear program for the double star, let us rewrite each transaction in the sequence as (s, t, x_i, y_i), where s is the source node, t is the target node, x_i is the weight of the transaction, and y_i denotes the fractional amount of the transaction accepted by the solution of the linear program. For processing transactions between clusters, we look at the star created by the centre and middle nodes of the double

Algorithm 2 Approximation algorithm for clustered nodes.

Input: Sequence of transactions X_t with LP solution, clustering $\mathcal{C} = \{C_1, C_2, \dots\}$
Output: Decisions to accept or reject transactions
1: $X_i \leftarrow \textsc{BetweenClusters}(X_t, \mathcal{C})$
2: $S_i \leftarrow \textsc{Solve}(X_i)$
3: **for** $C \in \mathcal{C}$ **do**
4: $\quad X_C \leftarrow \textsc{BalanceInCluster}(X_t, S_i, C)$
5: $\quad S_C \leftarrow \textsc{Solve}(X_C)$
6: **return** $S_i \cup S_b \cup (\bigcup_{C \in \mathcal{C}} X_C)$

star and treat these transactions as going only between these middle nodes. In Algorithm 2, this is handled by the function BetweenClusters which selects only transactions that go between clusters and treats clusters as a single node. As this becomes processing transactions on a star, this allows us to use Algorithm 1 to process transactions between different clusters. In Algorithm 2, this is handled by the function $\textsc{Solve}(X)$ in line 2 in Algorithm 2 which outputs the decisions using Algorithm 1 for every pair of nodes. Recall that Algorithm 1 also gives us an upper bound on the *reserve capacity* requirement for each cluster (middle node): for each middle node, we require reserves of size $\mathcal{O}(\sqrt{p}M)$, where M is the maximum of the capacity of the channel between the middle node and the centre of the double star c and the capacities of the channels between the middle node and the leaf nodes in its cluster. We call these reserves *cluster reserves*.

Processing transactions within clusters. Before processing transactions that go between nodes in a cluster, it is imperative that the cluster reserves are first shifted around within the cluster to ensure each leaf node in the cluster has enough reserves to process their between-cluster transactions (line 2 in Algorithm 2). This is handled by the function BalanceInCluster (described in Algorithm 3 in Section C.3), which modifies the solution of the double star LP such that cluster reserves are transferred from well-funded to depleted nodes. Finally, after balancing the reserve capacities, Algorithm 2 treats each cluster as an independent star graph and uses Algorithm 1 to process transactions within each cluster (function Solve in line 5). Using Algorithm 1 imposes a reserve capacity requirement of $\mathcal{O}(\sqrt{p}M')$ times the capacity for every leaf node, where M' is the capacity of the channel between the leaf node and the middle node.

Properties of balancing reserve capacities. We now prove several important properties of transporting reserve capacities within clusters as specified by BalanceInClusters, ensuring the correctness of BalanceInClusters. First, Lemma 7 states that we can modify the solution of the linear program (i.e., the fractional amount of transactions accepted by the linear program) to move the reserves capacities from the source s to target t, see Figure 4. From the statement of Lemma 7, we note that the only transactions that do not allow moving reserves are transactions that are fully accepted ($y_i = x_i$) or rejected ($y_i = 0$). Then, we quantify how much capacity we can move from one set of nodes to another in Lemma 8. Lastly, we show in Lemma 9 that we can always transport some reserves between any two sets of nodes. We defer the proofs of these lemmas to Sections A.4–A.6.

▶ **Lemma 7.** *If there is a transaction (s, t, x_i, y_i), we can change it to*
- *$(s, t, x_i, y_i + \alpha)$ to simulate moving α reserves from s to t, and*
- *$(s, t, x_i, y_i - \alpha)$ to simulate moving α reserves from t to s.*

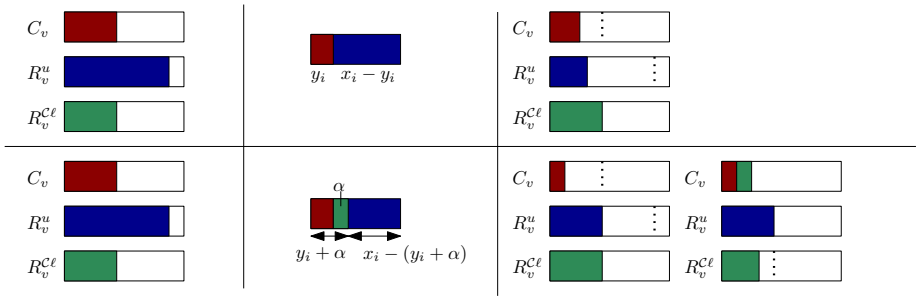

Figure 4 Movement of capacities, reserves and cluster reserves when accepting a transaction x_i from v to u where $v, u \in C_i$ for some cluster C_i. The case on the top represents the typical movement of capacities and reserves when accepting x_i. The case on the bottom represents the treatment of the reserves (and cluster reserves) when the change in cluster reserves ($R_v^{C\ell}$) is involved. The highlighted reserves are on v's side of the channel between v and the middle node. Similar changes are made on u's side of the channel between u and the middle node.

▶ **Lemma 8.** *Let S be a set of nodes that is connected to $C_i \setminus S$ for some cluster C_i by volume of at least $6|S| \cdot M$. Then at least $|S|M$ volume can be moved from S to $C_i \setminus S$.*

▶ **Lemma 9.** *Let X_t' be a subsequence of transactions where transactions incident to C_i have volume at least $tr \geq M$, then Algorithm 3 can transport at least tr from any set of nodes $S \subset C_i$ to any other set $T \subset C_i$. The LP solution is modified by volume at most $\mathcal{O}(\sqrt{p} \cdot tr)$.*

We now have the requisite ingredients to state and prove (in Section A.7) the main theorem of this section.

▶ **Theorem 10.** *Algorithm 2 is $\mathcal{O}(\sqrt{p})$ approximation of the problem where given a transaction sequence, the nodes are $(\sqrt{p}, \sqrt{p}, 24\sqrt{p})$-clustered.*

We also get the following corollary as a direct consequence of the proof of Theorem 10.

▶ **Corollary 11.** *Given (m, k, t)-clustered transaction sequence where $t \geq 24\sqrt{p}$, Algorithm 2 is $\mathcal{O}(\max(m, k))$-approximation of the problem.*

Finally, we end this section with two observations. The first observation effectively states that our algorithm just needs to focus on the total volume of partially accepted transactions and can ignore the volume of fully accepted and rejected transactions. The second observation allows us to further reduce the approximation ratio if we assume clustering of a larger depth.

Practical relaxations. We observe that the proof of Theorem 10 requires a condition on the partially accepted transactions (these are transactions which satisfy $0 < y_i < x_i$). However, we stress that the volume of fully accepted or rejected transactions can be arbitrary. Moreover, in every cluster, the middle node can interact with other clusters arbitrarily and this also does not affect the proof of Theorem 10.

Hierarchical clustering. We can use a similar idea to further reduce the approximation ratio if we assume that within each cluster the clusters themselves are also clustered. Namely, we first treat the top-level clusters as nodes and solve the problem as outlined by Algorithm 1. Then, we have some transactions pre-accepted or pre-rejected and we now shift the focus to the next level clusters. If these next-level clusters consist of other clusters, we treat them

again as nodes. We do this recursively until we reach the lowest-level clusters and then use Algorithm 2 to solve the problem within the lowest-level clusters. Note that we require the strength of the clustering to be bigger in order to achieve a similar approximation ratio in this setting. Nevertheless, assuming the nodes satisfy sufficiently strong clustering conditions, we can achieve an approximation ratio of $\mathcal{O}(p^{\frac{1}{3}})$ for two nested clusterings, and, in general, $\mathcal{O}(p^{\frac{1}{k+1}})$ for k nested clusterings (if the nodes in clusters are well distributed). We stress that although hierarchical clustering allows us to achieve a stronger approximation factor, it also comes with additional structural assumptions. However, we believe there are strong use cases of hierarchical clustering, for instance using a payment channel network to handle payments among users in a large company. The clusters would naturally follow the clustering of users into divisions and sub-divisions, with the head of each division being the centre of the star in their corresponding cluster.

6 Case study: the Lightning Network

To justify our transactions distribution assumption in Section 5, we perform an empirical case study of the largest and most commonly used PCN: Bitcoin's Lightning Network. Specifically, the research question we want to investigate is the following:

Are transactions in the Lightning Network clustered according to the clustering conditions stated in Section 5?

To address this question, we obtained the latest snapshot (dated September 2023) of the Lightning Network from the Lightning Network gossip repository [1]. We further restrict our examination to the largest connected component which contains $20,348$ users and $310,657$ channels. Figure 5 shows the distribution of degrees in the largest connected component, where a large number of nodes ($\sim 10\%$) are of degree ≤ 2.

Estimating the source and target of transactions. The main challenge in extracting transactions data from the Lightning Network is that the Lightning Network, by design, only provides information about the costs of transactions across channels, but does not provide information about transactions between users. Nevertheless, it is reasonable to assume that transactions between users are related to the channel capacities that connect them. Furthermore, we are mainly concerned with finding out if the clustering conditions hold for transactions between ordinary users with low degrees rather than between, e.g., large payment hubs with high degrees. We hence propose the following methodology: we first remove high degree nodes (i.e., nodes with degree ≥ 3) in the network and connect their neighbours to each other. Let us denote this resulting reduced network by G. In G, we assume nodes that are directly connected to each other transact with each other. Our methodology of generating the reduced network G is detailed in Algorithm 4 in our extended technical report [8].

Estimating the volume of transactions. Recall that a crucial clustering condition is that the volume of transactions that stay within a cluster is larger than the volume of the transactions that go between clusters. However, the privacy principles behind the design of the Lightning Network renders such transaction volume information hidden. Thus, a second challenge in our setting is extracting estimates of the size of transactions between users. In order to estimate transaction volume, we assume that the volume of transactions between two connected users in G is inversely proportional to the cost of a transaction. Namely, that the volume of transactions between users i and j is $1/(c_{i,j}+1)$ where $c_{i,j}$ is the minimum cost of sending a transactions of 100 milli-satoshi from i to j.

Note that the volume of transactions is estimated under the simplifying assumptions that all transactions have the same fixed volume, and therefore the cost for a transaction between a pair of users is fixed and specified using $cost_H$ on input.

We start with the largest strongly connected component of H as it is the largest set of users such that any user can send a transaction to any other. Also, to simplify the experiment, we assume that the volume of transactions from a user u to another user v is the same as the volume of transactions in the reverse order (from v to u).

This approach of computing costs for each pair of users runs in time linear with the number of edges of G. Note that the number of updates of a cost is at most the number of edges adjacent to the set $users$, which is bounded by $O(|users|)$ as the degree of each vertex is bounded.

Results. Figure 6 and Table 1 in our extended technical report [8] show that we obtain 18 clusters with average within-cluster transaction volume of 826.1, and between-cluster transaction volume of 4.27. This gives us a ratio of 193.2 of within-cluster and between-cluster transactions, which confirms our experimental hypothesis.

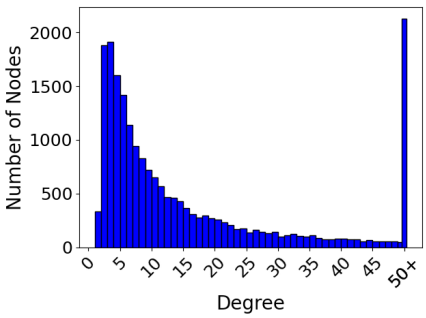

Figure 5 Degree distribution of the nodes in the largest connected component of the Lightning Network.

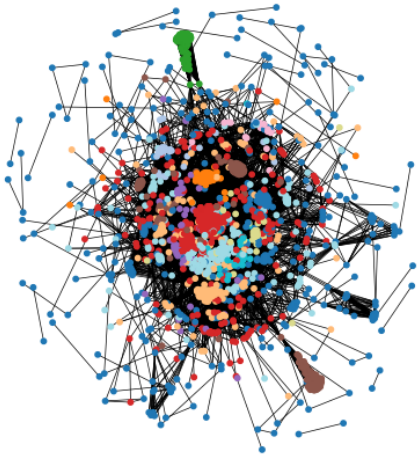

Figure 6 Clustering of the nodes in the reduced network G. Each color represents a different cluster.

7 Conclusion

In this work, we generalise a model from processing transactions over a single payment channel to an entire PCN. To this end, we first define a network creation stage of the problem which examines how to create a network with minimal capacity cost, and we show that the star is a 2-approximation of the optimal capacity cost. We then propose two algorithms to address weighted transaction processing over a network. The first algorithm assumes a star topology over the p parties, but apart from that works without any further assumptions on the transaction sequence, and gives an approximation ratio of $\mathcal{O}(p)$. The second algorithm assumes a double star topology and a clustering of the transaction sequence, and gives an approximation ratio of $\mathcal{O}(\sqrt{p})$. Finally, we also perform an empirical case study estimating transaction information from the topology of the Lightning Network, and our estimates suggest that our clustering assumptions regarding the input transaction sequence are close to reality.

We believe our work is an important first step in the direction of designing optimal networks and algorithms for transaction processing in PCN. We highlight two interesting directions for future work. First, it would be interesting to extend our model and algorithms to the online setting, which will also complement the analysis of [4, 7]. Secondly, developing better estimates regarding transaction information in real world PCNs would also be paramount to developing better optimisation algorithms and heuristics.

References

1 Lightning network topology datasets. URL: https://github.com/lnresearch/topology.

2 Raiden network. https://raiden.network/, 2017.

3 Hagit Attiya, Leah Epstein, Hadas Shachnai, and Tami Tamir. Transactional contention management as a non-clairvoyant scheduling problem. *Algorithmica*, 57(1):44–61, 2010. doi:10.1007/S00453-008-9195-X.

4 Georgia Avarikioti, Kenan Besic, Yuyi Wang, and Roger Wattenhofer. Online payment network design. In *DPM/CBT@ESORICS*, volume 11737 of *Lecture Notes in Computer Science*, pages 307–320. Springer, 2019. doi:10.1007/978-3-030-31500-9_20.

5 Zeta Avarikioti, Lioba Heimbach, Yuyi Wang, and Roger Wattenhofer. Ride the lightning: The game theory of payment channels. In *Financial Cryptography*, volume 12059 of *Lecture Notes in Computer Science*, pages 264–283. Springer, 2020. doi:10.1007/978-3-030-51280-4_15.

6 Zeta Avarikioti, Tomasz Lizurej, Tomasz Michalak, and Michelle Yeo. Lightning creation games. *CoRR*, abs/2306.16006, 2023. doi:10.48550/arXiv.2306.16006.

7 Mahsa Bastankhah, Krishnendu Chatterjee, Mohammad Ali Maddah-Ali, Stefan Schmid, Jakub Svoboda, and Michelle Yeo. R2: boosting liquidity in payment channel networks with online admission control. In *FC (1)*, volume 13950 of *Lecture Notes in Computer Science*, pages 309–325. Springer, 2023. doi:10.1007/978-3-031-47754-6_18.

8 Krishnendu Chatterjee, Jan Matyáš Křišťan, Stefan Schmid, Jakub Svoboda, and Michelle Yeo. Boosting payment channel network liquidity with topology optimization and transaction selection. Cryptology ePrint Archive, Paper 2025/1484, 2025. URL: https://eprint.iacr.org/2025/1484.

9 Anamika Chauhan, Om Prakash Malviya, Madhav Verma, and Tejinder Singh Mor. Blockchain and scalability. In *QRS Companion*, pages 122–128. IEEE, 2018. doi:10.1109/QRS-C.2018.00034.

10 Chandra Chekuri, Sanjeev Khanna, and F Bruce Shepherd. The all-or-nothing multicommodity flow problem. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 156–165, 2004. doi:10.1145/1007352.1007383.

11 Anne Condon and Richard M. Karp. Algorithms for graph partitioning on the planted partition model. *Random Struct. Algorithms*, 18(2):116–140, 2001. doi:10.1002/1098-2418(200103)18:2\%3C116::AID-RSA1001\%3E3.0.CO;2-2.

12 Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed E. Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains - (A position paper). In *Financial Cryptography Workshops*, volume 9604 of *Lecture Notes in Computer Science*, pages 106–125. Springer, 2016. doi:10.1007/978-3-662-53357-4_8.

13 Christian Decker, Rusty Russell, and Olaoluwa Osuntokun. eltoo: A simple layer2 protocol for bitcoin. https://blockstream.com/eltoo.pdf, 2018.

14 Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems (SSS)*, pages 3–18. Springer, 2015. doi:10.1007/978-3-319-21741-3_1.

15 Oguzhan Ersoy, Stefanie Roos, and Zekeriya Erkin. How to profit from payments channels. In *Financial Cryptography*, volume 12059 of *Lecture Notes in Computer Science*, pages 284–303. Springer, 2020. doi:10.1007/978-3-030-51280-4_16.

16 Paolo Guasoni, Gur Huberman, and Clara Shikhelman. Lightning network economics: Topology. *SSRN Electronic Journal*, 2023. doi:10.2139/ssrn.4439190.

17 Paolo Guasoni, Gur Huberman, and Clara Shikhelman. Lightning network economics: Channels. *Manag. Sci.*, 70(6):3827–3840, 2024. doi:10.1287/MNSC.2022.01664.

18 Rachid Guerraoui, Maurice Herlihy, and Bastian Pochon. Polymorphic contention management. In Pierre Fraigniaud, editor, *Distributed Computing*, pages 303–323, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/11561927_23.

19 Rachid Guerraoui, Maurice Herlihy, and Bastian Pochon. Towards a theory of transactional contention managers. In *PODC*, pages 316–317. ACM, 2006. doi:10.1145/1146381.1146429.

20 Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. The consensus number of a cryptocurrency. In *PODC*, pages 307–316. ACM, 2019. doi:10.1145/3293611.3331589.

21 Piyush Kumar Gupta and Panganamala Ramana Kumar. The capacity of wireless networks. *IEEE Trans. Inf. Theory*, 46:388–404, 2000. doi:10.1109/18.825799.

22 Maurice Herlihy. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 13(1):124–149, January 1991. doi:10.1145/114005.102808.

23 Paul Holland, Kathryn B. Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5:109–137, 1983. URL: https://api.semanticscholar.org/CorpusID:34098453.

24 Anurag Jain, Shoeb Siddiqui, and Sujit Gujar. We might walk together, but I run faster: Network fairness and scalability in blockchains. In *AAMAS*, pages 1539–1541. ACM, 2021. doi:10.5555/3463952.3464152.

25 Patrick McCorry, Surya Bakshi, Iddo Bentov, Sarah Meiklejohn, and Andrew Miller. Pisa: Arbitration outsourcing for state channels. In *AFT*, 2019. URL: 10.1145/3318041.3355461.

26 Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf, 2015.

27 Prabhakar Raghavan and Clark D Thompson. Provably good routing in graphs: regular arrays. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 79–87, 1985. doi:10.1145/22145.22154.

28 Stefan Schmid, Jakub Svoboda, and Michelle Yeo. Weighted packet selection for rechargeable links in cryptocurrency networks: Complexity and approximation. In *SIROCCO*, volume 13892 of *Lecture Notes in Computer Science*, pages 576–594. Springer, 2023. doi:10.1007/978-3-031-32733-9_26.

A Omitted proofs

A.1 Proof of Lemma 1

▶ **Lemma 1.** *Given the sequence of transactions $\mathcal{S}$, we have $LP(\mathcal{S}, K_p) \leq AC(\mathcal{S}, G)$.*

Proof. We first observe that adding channels to a graph can only decrease the capacity cost (adding channels can only make the shortest paths shorter). This means that the capacity cost is the lowest for the complete graph, no matter what is the optimal graph topology (as computed by minimising the *total* cost of both capacity and channel creation). The LP computes the optimal capacity for the complete graph K_p, which is a superset of every graph, including the optimal graph. ◀

A.2 Proof of Theorem 2

▶ **Theorem 2.** *For every graph G over p nodes, and any graph G' with diameter d, the following inequality holds*

$$AC(\mathcal{S}, G') \leq d \cdot AC(\mathcal{S}, G)\,.$$

Proof. Since for the complete graph K_n, we have $AC(\mathcal{S}, K_n) \leq AC(\mathcal{S}, G)$, we simply need to show that $AC(\mathcal{S}, G') \leq d \cdot AC(\mathcal{S}, G)$. We modify the solution of the LP for the complete graph by adding at most d-times the capacity and prove that this graph processes all transactions.

We start with graph G' with zero capacities. For all pairs of nodes v and u, let the capacity between them be $w_{v,u}$. We find the shortest path between them in G', since the diameter is at most d, the length of the path is at most d. Then, we increase the capacity of every channel on the path by $w_{v,u}$. We denote this capacity to a special bucket that will be used to process transactions that the LP sends along u, v. At the start at edge k, l, where k is closer to v and l is to u, we increase the balance at the k's side by $C^0_{v,u}$ (and l's balance is $C^0_{u,v}$), the balance at the edge v, u at the start as given by the LP.

After we increase the capacities of G' for every pair of nodes, we process the transactions. Any time the transaction of size s was forwarded along some edge u, v, we can transfer s on the shortest path between u and v from the bucket on the edge denoted u, v. If the bucket for some edge becomes negative, the solution of LP is infeasible.

By this, we can process all transactions. Since the diameter is at most d, we know that the increase in capacity is at most d-times. ◀

A.3 Proof of Theorem 6

▶ **Theorem 6.** *Given the transaction sequence X_t, let $\mathcal{C}$ be the optimal capacity cost and $\mathcal{R}$ the optimal rejection cost on a star. Then Algorithm 1 run for every pair of nodes incurs a rejection cost of at most $(\sqrt{3}+1)\mathcal{R}$ and capacity cost of at most $(1+(p-1)\cdot\sqrt{3})\mathcal{C}$.*

Proof. Every node has $p-1$ connections while the size of the channel is viewed as the minimum of the two sizes as denoted by the LP. The reserves are $\sqrt{3}$ times that value. Moreover, every channel in a star has funds amount as dictated by the LP. Altogether it gives $1+(p-1)\cdot\sqrt{3}$ times of $\mathcal{C}$.

The algorithm ensures that the in every channel on the right side, there is at least the amount prescribed by the LP. This means that all fully accepted ($x_i = y_i$) transactions are accepted. Moreover, from [28], we know that the rejection cost between two nodes is at most $\sqrt{3}+1$ times the rejection cost of the optimal rejection cost $\mathcal{R}$. Thus, Algorithm 1 incurs a rejection cost of at most $(\sqrt{3}+1)\mathcal{R}$ and capacity cost of at most $(1+(p-1)\cdot\sqrt{3})\mathcal{C}$. ◀

A.4 Proof of Lemma 7

▶ **Lemma 7.** *If there is a transaction (s, t, x_i, y_i), we can change it to*

- *$(s, t, x_i, y_i + \alpha)$ to simulate moving α reserves from s to t, and*
- *$(s, t, x_i, y_i - \alpha)$ to simulate moving α reserves from t to s.*

Proof. Every node has $p-1$ connections while the size of the channel is viewed as the minimum of the two sizes as denoted by the LP. The reserves are $\sqrt{3}$ times that value. Moreover, every channel in a star has funds amount as dictated by the LP. Altogether it gives $1+(p-1)\cdot\sqrt{3}$ times of $\mathcal{C}$.

The algorithm ensures that the in every channel on the right side, there is at least the amount prescribed by the LP. This means that all fully accepted ($x_i = y_i$) transactions are accepted. Moreover, from [28], we know that the rejection cost between two nodes is at most $\sqrt{3}+1$ times the rejection cost of the optimal rejection cost $\mathcal{R}$. Thus, Algorithm 1 incurs a rejection cost of at most $(\sqrt{3}+1)\mathcal{R}$ and capacity cost of at most $(1+(p-1)\cdot\sqrt{3})\mathcal{C}$. ◀

A.5 Proof of Lemma 8

▶ **Lemma 8.** *Let S be a set of nodes that is connected to $C_i \setminus S$ for some cluster C_i by volume of at least $6|S| \cdot M$. Then at least $|S|M$ volume can be moved from S to $C_i \setminus S$.*

Proof. From the condition 3 of the clustering conditions, we know that at least $2|S|M$ volume moves from the cluster and $2|S|M$ volume to the cluster. For the sake of contradiction, suppose that we can move at most $m < |S|M$ from S to $C_i \setminus S$.

From Lemma 7, we know that we cannot move only on fully accepted transactions from S and rejected transactions from $C_i \setminus S$. Let m' be the incoming volume to S that can be moved. Then in the subsequence in the solution of LP, S receives m' volume and gives up $2|S|M - (m - m')$ volume. That means on average it lost at least $2|S|M - (m - m') - m' = 2|S|M - m > |S|M$. From the Dirichlet principle, at least one node lost more than M. This is impossible since M is the highest capacity in C_i and the solution of the LP ensures that for any subsequence the difference is at most M. ◀

A.6 Proof of Lemma 9

▶ **Lemma 9.** *Let X'_t be a subsequence of transactions where transactions incident to C_i have volume at least $tr \geq M$, then Algorithm 3 can transport at least tr from any set of nodes $S \subset C_i$ to any other set $T \subset C_i$. The LP solution is modified by volume at most $\mathcal{O}(\sqrt{p} \cdot tr)$.*

Proof. Since there was tr transported, inside the cluster there was at least $24 \cdot tr \cdot |C_i|$ transported, and next to S, there was at least $6tr|S|$ transported. From Lemma 8, that means that at least $tr|S|$ goes away from S to other nodes.

Let $S_0 = S$ and S_i be the nodes that we can transport nonzero amount from S_{i-1}. Similarly, let $T_0 = T$ and T_i be the nodes that we can transport nonzero amount to T_{i-1}. Since the clustering conditions, we have that the size of S_i and T_i is increasing if their size is at most half of the cluster. That means they intersect at some point.

We look at the minimum mn that can be transported and we change the transactions such that in v the reserves are decreased by mn and in u they are increased by mn. After the change, we know that the clustering conditions still hold, now we just need to decrease the inequalities by mn. Therefore, we can find the path again. ◀

A.7 Proof of Theorem 10

▶ **Theorem 10.** *Algorithm 2 is $\mathcal{O}(\sqrt{p})$ approximation of the problem where given a transaction sequence, the nodes are $(\sqrt{p}, \sqrt{p}, 24\sqrt{p})$-clustered.*

Proof. We are using a double star, which means it is at least a 4 approximation of the optimal topology from Theorem 2.

The algorithm uses Algorithm 1 for every pair of clusters (between middle nodes) to accept and reject transactions between them. This incurs cost for additional capacity equal $\mathcal{O}(\sqrt{p})$ times the cost of the capacity cost of the LP for the capacity between clusters. The rejection cost is at most $(\sqrt{3}+1)$ times higher than the rejection cost between clusters. All of this follows from Theorem 6.

Next, from Lemma 9, we know that we can transport reserves such that everything between clusters is accepted or rejected from the previous algorithm. It changes the values of the LP by at most $\mathcal{O}(\sqrt{p})$ times the volume of rejected transactions between clusters.

Finally, again from Theorem 6, we can solve the problem inside the clusters. This incurs capacity cost $\mathcal{O}(\sqrt{p})$ times the capacity cost of the solution inside the clusters. Moreover, the cost for rejection is $(\sqrt{3}+1)$ times the rejection cost in the modified solution of LP (X'_t).

The modified rejection cost of solution of LP is the rejection cost of the original solution plus at most $\mathcal{O}(\sqrt{p})$ times the rejection cost of transactions between clusters (from Lemma 9). Altogether, we get that the algorithm is $\mathcal{O}(\sqrt{p})$ approximation algorithm. ◀

B Omitted details of star topology solution

B.1 Modified channel algorithm details

We now describe the decision making process in Algorithm 1 for transactions going from u to v (the other direction is symmetric). Algorithm 1 accepts a transaction as long as the reserve R_u is above a threshold (i.e., $R_u \geq \frac{\sqrt{3}-1}{2}M$, line 6). Accepting a transaction decreases the capacities C_u and R_u^v of u by y_i and $x_i - y_i$ respectively (refer to the "accept" case in Figure 2). If accepting the transaction would make R_u decrease by too much and the LP demands to accept the transaction *weakly* (i.e., $\frac{y_i}{x_i} < \frac{\sqrt{3}}{\sqrt{3}+1}$, line 9), the algorithm rejects the transaction. Rejecting a transaction from say u to v decreases both C_u and R_u^v of by y_i (refer to the "reject" case in Figure 2). If the reserve on u's side after accepting the transaction would be too small (i.e., $R_u < \frac{\sqrt{3}-1}{2}M$, line 17), the algorithm looks at all subsequent transactions *between* u and v in the transaction sequence. It accepts all transactions that increases R_u while rejecting weakly accepted transactions from u to v (rejecting them also increases R_u). From *strongly* accepted transactions (i.e., $\frac{y_i}{x_i} \geq \frac{\sqrt{3}}{\sqrt{3}+1}$) from u to v, the algorithm tries to accept all of them while simulating the movement of capacities between the reserves until either $R_u > \frac{\sqrt{3}-1}{2}M$ or $R_u < 0$. If $R_u > \frac{\sqrt{3}-1}{2}M$, then all transactions can be accepted. If $R_u < 0$, the algorithm rejects strongly accepted transactions until $R_u > \frac{\sqrt{3}-1}{2}M$. The proof in [28] shows that doing so maintains the approximation ratio at $\sqrt{3}+1$ for the rejection cost.

Algorithm details. The simulating of the reserves is implemented by a heap. The heap is empty while $R_u \geq \frac{\sqrt{3}-1}{2}M$, and when $R_u < \frac{\sqrt{3}-1}{2}M$ it starts to fill up. When $R_u < 0$, then the decision is changed for transactions from the top of the heap until $R_u \geq \frac{\sqrt{3}-1}{2}M$ again.

Observe that for Algorithm 1 the funds on both sides can shift and the reserves are not affected. This allows us to compose the algorithm from pairwise reserves with shared funds denoted by the linear program.

The algorithm describes decisions for the star, so a transaction goes from u to c (star center) and then from c to v. That means it needs to traverse two channels. The algorithm sees it as only pairwise interaction. Since the decisions for the two channels are always the same for both channels (a transaction is either accepted or rejected for both), we can have the reserves stored twice, once between u and c, the second time between v and c. The value R_u is then the reserves at u in channel u, c and at c in the channel c, v. Both of these reserves move in sync.

C Omitted details of clustering algorithm

C.1 Stochastic Block Model details

The planted partition model creates a random graph where nodes are partitioned into clusters. Inside a cluster, two nodes are connected with probability q_1, otherwise, they are connected with probability q_2. The planted partition model creates an unweighted and undirected graph. Our graph created by the packets is directed, weighted, and temporal. These changes call for modification of the planted partition model.

The biggest change to the model is brought by the temporality. Some edges follow after others, we require that all of the conditions hold for some continuous subsequence of edges incident to some cluster of bigger size.

To make the graph weighted, we view q_1 and q_2 as ratios between volumes of the packets between the clusters and inside the cluster. This view is expressed by the condition 1 of the clustering conditions.

To make the graph directed, we allow for arbitrary orientation of the edges with the condition that at least $\frac{1}{3}$ of the volume needs to be directed in both directions. It is expressed by condition 2 of the clustering conditions. Note that $\frac{1}{3}$ of the volume can be oriented in any direction, no matter the other packets.

Finally, the condition 2 of the clustering conditions ensures that some volume is incident to every node.

Note that all of the conditions would be satisfied with high probability in the stochastic block model. Moreover, there are other conditions that would be satisfied (for instance any two clusters would send packets between each other at the same rate) that we do not require. From the above the graph generated by the planted partition model with $\sqrt{p}$ partitions of size $\sqrt{p}$ and $q_2 = \frac{1}{p}$ and $q_1 = 1 - q_2$ satisfies all clustering conditions with high probability.

C.2 Linear program for double star

Let the set of leaf nodes be L and the set of middle nodes be M, and c denote the central node of the star. For a leaf node u, let $M(u)$ be the middle node corresponding to u's cluster.

For a leaf node v and natural number i, we use $C_{v,i}$ and $C'_{v,i}$ to denote the capacity at the channel $(v, M(v))$ after processing the i-th transaction on the side of v and $M(v)$ respectively. Moreover let C_v denote the total capacity of the channel $(v, M(v))$, i.e., $C_v = C_{v,i} + C'_{v,i}$. For a middle node v and i, we use $M_{v,i}$ and $M'_{v,i}$ to denote the capacity at the channel (v, c) after processing the i-th transaction on the side of v and c respectively. Moreover, let M_v denote the total capacity of the channel (v, c).

$$\text{minimise} \sum_{u \in L} C_u + \sum_{u \in M} M_u + \tag{2}$$

$$\sum_i f(x_i - y_i) + \frac{m(x_i - y_i)}{x_i} \tag{3}$$

$$\begin{aligned}
\forall i &: 0 \le y_i \le x_i \\
\forall (v, u, x_i) &: C_v, C_u \ge x_i \\
&\quad C_{v,i} = C_{v,i-1} - y_i \\
&\quad C'_{v,i} = C'_{v,i-1} + y_i \\
&\quad C'_{u,i} = C'_{u,i-1} - y_i \\
&\quad C_{u,i} = C_{u,i-1} + y_i \\
\forall (v, u, x_i) \text{ between clusters:} &\ M_{M(u)}, M_{M(v)} \ge x_i \\
&\quad M_{M(v),i} = M_{M(v),i-1} - y_i \\
&\quad M'_{M(v),i} = M'_{M(v),i-1} + y_i \\
&\quad M'_{M(u),i} = M'_{M(u),i-1} - y_i \\
&\quad M_{M(u),i} = M_{M(u),i-1} + y_i
\end{aligned}$$

Algorithm 3 Transporting reserves between nodes.

Input: Subsequence of transactions X_t inside one cluster, two nodes v, u, amount A.
Output: Modified X_t such that A in reserves is transported from v to u.

```
1: procedure TRANSPORT(X_t, v, u, A)
2:     M ← PATHBETWEEN(v, u)
3:     mn ← MINIMUMONPATH(M)
4:     mn ← min(mn, A)
5:     for (dir, x, y, x_i, y_i) ∈ M do
6:         if dir from v to u then
7:             y_i ← y_i − mn
8:         else
9:             y_i ← y_i + mn
10:    if A − mn > 0 then
11:        TRANSPORT(X_t, v, u, A − mn)
```

C.3 Algorithm to transport cluster reserves

Here we detail Algorithm 3, which transfers cluster reserves from well-funded to depleted nodes *within* a cluster, so that the cluster nodes have sufficient reserves to handle transactions that go *between* the clusters.