

High Throughput Cryptocurrency Routing in Payment Channel Networks

Vibhaalakshmi Sivaraman¹, Shaileshh Bojja Venkatakrishnan¹, Kathy Ruan²,
Parimarjan Negi¹, Lei Yang¹, Radhika Mittal³,
Mohammad Alizadeh¹, and Giulia Fanti²

¹Massachusetts Institute of Technology

²Carnegie Mellon University

³University of Illinois at Urbana-Champaign

Abstract

Despite growing adoption of cryptocurrencies, making fast payments at scale remains a challenge. Payment channel networks (PCNs) such as the Lightning Network have emerged as a viable scaling solution. However, completing payments on PCNs is challenging: payments must be routed on paths with sufficient funds. As payments flow over a single channel (link) in the same direction, the channel eventually becomes depleted and cannot support further payments in that direction; hence, naive routing schemes like shortest-path routing can deplete key payment channels and paralyze the system. Today’s PCNs also route payments atomically, worsening the problem. In this paper, we present Spider, a routing solution that “packetizes” transactions and uses a multi-path transport protocol to achieve high-throughput routing in PCNs. Packetization allows Spider to complete even large transactions on low-capacity payment channels over time, while the multi-path congestion control protocol ensures balanced utilization of channels and fairness across flows. Through extensive simulations, we show that Spider requires less than 25% of the funds needed by state-of-the-art approaches to successfully route over 95% of the transactions on balanced traffic demands, and requires only one on-chain transaction for every 10K transactions routed to achieve full throughput on imbalanced demands.

1 Introduction

Despite their growing adoption, cryptocurrencies suffer from poor scalability. For example, the Bitcoin [4] network processes 7 transactions per second, and Ethereum [12] 15 transactions/second, which pales in comparison to the 1,700 transactions per second achieved by the VISA network [53]. Scalability thus remains a major hurdle to the adoption of cryptocurrencies for retail and other

The lead author can be contacted at vibhaa@mit.edu

large-scale applications. The root of the scalability challenge is the inefficiency of the underlying consensus protocol: every transaction must go through full consensus to be confirmed, which can take anywhere from several minutes to hours [41].

A leading proposal for improving the scalability of cryptocurrencies relies on so-called *payment channels*. A payment channel is a cryptocurrency transaction that escrows or dedicates money on the blockchain for exchange with a prespecified user for a predetermined duration. For example, Alice can set up a payment channel with Bob in which she escrows 10 tokens for a month. Now Alice can send Bob (and only Bob) signed transactions from the escrow account, and Bob can validate them privately in a secure manner without mediation on the blockchain (§2). If Bob or Alice want to close the payment channel at any point, they can broadcast the most recent signed transaction message to the blockchain to finalize the transfer of funds.

The versatility of payment channels stems from **payment channel networks** (PCNs), in which users who do not share direct payment channels can route transactions through intermediaries for a nominal fee. PCNs enable fast, secure transactions without requiring consensus on the blockchain for every transaction. PCNs have received a great deal of attention in recent years, and many blockchains are looking to PCNs to scale throughput without overhauling the underlying consensus protocol. For example, Bitcoin has deployed the Lightning network [13, 9], and Ethereum uses Raiden [16].

For PCNs to be economically viable, the network must be able to support high *transaction throughput*. This is necessary for intermediary nodes (routers) to profitably offset the opportunity cost of escrowing funds in payment channels, and for encouraging end-user adoption by providing an appealing quality of payment service. But, a transaction is successful only if all channels along its route have sufficient funds. This makes payment channel *routing*, the protocol by which a path is chosen for a transaction, of paramount importance.

Existing payment channel routing protocols achieve poor throughput, for two main reasons. First, they attempt to route each incoming transaction atomically and instantaneously, in full. This approach is harmful, particularly for larger transactions, because a transaction fails completely if there is no path to the destination with enough funds. Second, existing protocols fail to keep payment channels *balanced*. A payment channel becomes imbalanced when the transaction rate across it is higher in one direction than the other; the party making more transactions eventually runs out of funds and cannot send further payments without “refilling” the channel via an on-chain transaction (i.e., committing a new transaction to the blockchain). Most PCNs today route transactions naively on shortest paths with no consideration for channel balance; this can leave many channels depleted, reducing throughput for everyone in the network. We describe a third problem, the creation of *deadlocks* in certain scenarios, in §3.

In this paper we present *Spider*, a multi-path transport protocol that achieves balanced, high-throughput routing in PCNs, building on concepts in an earlier position paper [22]. Spider’s design centers on two ideas that distinguish it from existing approaches. First, Spider sends “packetize” transactions, splitting them into transaction-units that can be sent across different paths at different rates. By enabling congestion-control-like mechanisms for PCNs, this packet-switched approach makes it possible to send large payments on low-capacity payment channels over a period of time. Second, Spider develops a simple multi-path congestion control algorithm that promotes balanced channels while maximizing throughput. Spider’s senders use a simple one-bit congestion signal from the routers to adjust window sizes, or the number of outstanding transaction-units, on each of their paths.

Spider’s congestion control algorithm is similar to multi-path congestion control protocols like MPTCP [56] developed for Internet congestion control. But the routing problem it solves in PCNs differs from standard networks in fundamental ways. Payment channels can only route transactions by moving a finite amount of funds from one end of the channel to the other. Because of this, the capacity of a payment channel — the transaction rate that it can support — varies depending on how it is used; a channel with balanced demand for routing transactions in both directions can support a higher rate than an imbalanced one. Surprisingly, we find that a simple congestion control protocol can achieve such balanced routing, despite not being designed for that purpose.

We make the following contributions:

1. We articulate challenges for high-throughput routing in payment channel networks (§3), and we formalize the balanced routing problem (§5). We show that the maximum throughput achievable in a PCN depends on the nature of the transaction pattern: circulation demands (participants send on average as much as they receive) can be routed entirely with sufficient network capacity, while demands that form Directed Acyclic Graphs (DAGs) where some participants send more than they receive cannot be routed entirely in a balanced manner. We also show that introducing DAG demands can create deadlocks that stalls all payments.
2. We propose a packet-switched architecture for PCNs (§4) that splits transactions into transaction-units and multiplexes them across paths and time.
3. We design Spider (§6), a multi-path transport protocol that (i) maintains balanced channels in the PCN, (ii) uses the funds escrowed in a PCN efficiently to achieve high throughput, and (iii) is fair to different payments.
4. We build a packet-level simulator for PCNs and validate it with a small-scale implementation of Spider on the LND Lightning Network codebase [13]. Our evaluations (§7) show that (i) compared to the state-of-the-art, on circulation demands where 100% throughput is achievable, Spider completes over 95% of the transactions with 25% of the funds, and can complete 1.3-1.8x more of the largest 25% of transactions based on a credit card transactions dataset [32]; (ii) on DAG demands where 100% throughput is not achievable, Spider recovers 100% throughput with on-chain rebalancing as infrequent as every 10K transactions. In contrast, currently deployed approaches, only recover 85% throughput even when rebalancing every 10 transactions.

2 Background

Bidirectional payment channels are the building blocks of a payment channel network. A bidirectional payment channel allows a sender (Alice) to send funds to a receiver (Bob) and vice versa. To open a payment channel, Alice and Bob jointly create a transaction that escrows money for a fixed amount of time [44]. Suppose Alice puts 3 units in the channel, and Bob puts 4 (Fig. 1). Now, if Bob wants to transfer one token to Alice, he sends her a cryptographically-signed message asserting that he approves the new balance. This message is not committed to the blockchain; Alice simply holds on to it. Later, if Alice wants to send two tokens to Bob, she sends a signed message to Bob approving the new balance (bottom left, Fig. 1). This continues until one party decides to close the channel, at which point they publish the latest message to the blockchain asserting the channel balance. If one party tries to cheat by publishing an earlier balance, the cheating party loses all the money they escrowed to the other party [44].

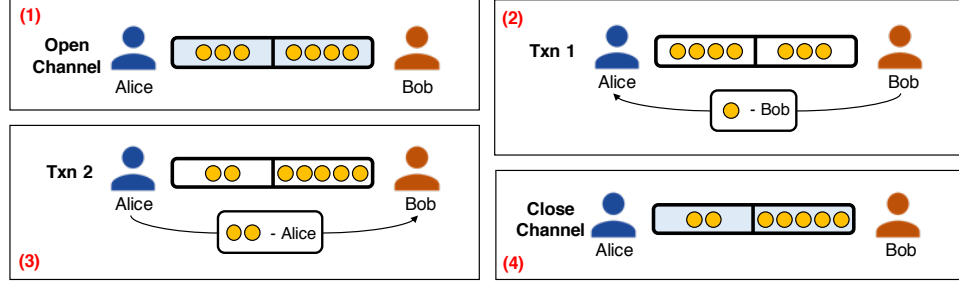


Figure 1: Bidirectional payment channel between Alice and Bob. A blue shaded block indicates a transaction that is committed to the blockchain.

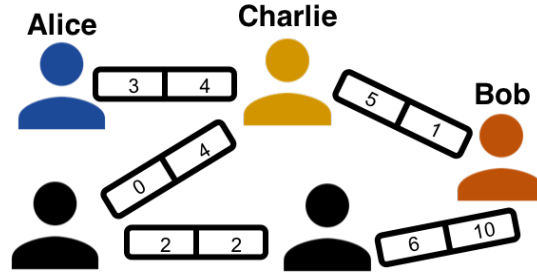


Figure 2: In a payment channel network, Alice can transfer money to Bob by using intermediate nodes' channels as relays. There are two paths from Alice to Bob, but only the path (Alice, Charlie, Bob) can support 3 tokens.

A payment channel network is a collection of bidirectional payment channels (Fig. 2). If Alice wants to send three tokens to Bob, she first finds a path to Bob that can support three tokens of payment. Intermediate nodes on the path (Charlie) will relay payments to their destination. Hence in Fig. 2, two transactions occur: Alice to Charlie, and Charlie to Bob. To incentivize Charlie to participate, he receives a routing fee. To prevent him from stealing funds, a cryptographic hash lock ensures that all intermediate transactions are only valid after a transaction recipient knows a private key generated by Alice [16]. Once Alice is ready to pay, she gives that key to Bob; he can either broadcast it (if he decides to close the channel) or pass it to Charlie. Charlie is incentivized to relay the key upstream to Alice so that he can also get paid. Note that Charlie's payment channels with Alice and Bob are independent: Charlie cannot take move funds between them without going through the blockchain.

3 Challenges in Payment Channel Networks

A major cost of running PCNs is the collateral needed to set up payment channels. As long as a channel is open, that collateral is locked up, incurring an opportunity cost for the owner. For PCNs to be financially viable, this opportunity cost should be offset by routing fees, which are charged on each transaction that passes through a router. To collect more routing fees, routers try to process as many transactions as possible for a given amount of collateral. A key performance metric is therefore the *transaction throughput per unit collateral* where throughput itself is measured either

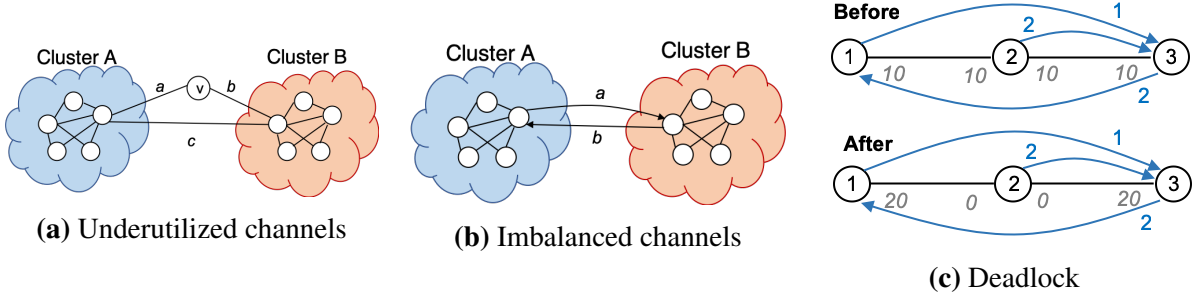


Figure 3: Example illustrating the problems with state-of-the-art PCN routing schemes.

in number of transactions per second or transaction value per second.

Current PCN designs exhibit poor throughput due to naive design choices in three main areas: (1) *how* to route transactions, (2) *when* to send them and, (3) *deadlocks*.

Challenge #1: How to route transactions? A central question in PCNs is what route(s) to use for sending a transaction from sender to destination. PCNs like the Lightning and Raiden networks are source-routed.¹ Most clients by default pick the shortest path from the source to the destination.

However, shortest-path routing degrades throughput in two key ways. The first is to cause underutilization of the network. To see this, consider the PCN shown in Fig. 3a. Suppose we have two clusters of nodes that seek to transact with each other at roughly the same rate on average, and the clusters are connected by two paths, one consisting of channels $a - b$, and the other channel c . If the nodes in cluster A try to reach cluster B via the shortest path, they would all take channel c , as would the traffic in the opposite direction. This leads to congestion on channel c , while channels a and b are under-utilized.

A second problem is more unique to PCNs. Consider a similar topology in Figure 3b, and suppose we fully utilize the network by sending all traffic from cluster $A \rightarrow B$ on edge a and all traffic from cluster $B \rightarrow A$ on edge b . While the rate on both edges is the same, as funds flow in one direction over a channel, the channel becomes *imbalanced*: all of the funds end up on one side of the channel. Because of this, cluster A can no longer send payments until it receives funds from cluster B on the edge a or it deposits new funds into the channel a via an on-chain rebalancing transaction. The same applies to cluster B on edge b . Since on-chain transactions are expensive and slow, it is desirable to avoid them. Naive routing schemes like shortest-path routing do not account for this problem, thereby leading to reduced throughput (§7). Hence, to exhibit good throughput, it is important to choose routes that actively prevent channel imbalance. For example, in Figure 3b, we could send half of the $A \rightarrow B$ traffic on edge a , and half on edge b , and the same for the $B \rightarrow A$ traffic. The challenge is making these decisions in a fully decentralized way.

Challenge #2: When to send transactions? Another central problem is *when* to send transactions. Most existing PCNs are circuit-switched: transactions are processed instantaneously and atomically upon arrival [44, 16]. This causes a number of problems. If a transaction’s value exceeds the available balance on each path from the source to the destination, the transaction fails. Since transaction values in the wild tend to be heavy-tailed [32, 28], either a substantial fraction of real transactions will fail as PCN usage grows, or payment channel operators will need to provision

¹This was done in part for privacy reasons: transactions in the Lightning network use onion-routing, which is easy to implement with source routing [31].

higher collateral to satisfy demand.

Even when transactions do not fail outright, sending transactions instantaneously and atomically exacerbates the imbalance problem by transferring the full transaction value to one side of the channel. A natural idea to alleviate these problems is to “packetize” transactions: transactions can be split into smaller transaction units that can be multiplexed over space (by traversing different paths) and in time (by being sent at different rates). Versions of this idea have been proposed before; atomic multi-path payments (AMP) enable transactions to traverse different paths in the Lightning network [3], and the Interledger protocol uses a similar packetization to conduct cross-ledger payments [51]. However, a key observation is that it is not enough to subdivide transactions into smaller units: to achieve good throughput, it is also important to multiplex in *time* as well, by performing congestion control. If there are large bursts of transactions in one direction on a channel, simply sending them out immediately in small units doesn’t imply transaction success, but throttling smaller units to match the rate in the opposite direction on the same payment channel allows balance to be replenished and transactions to complete.

Challenge #3: Deadlocks. The third challenge in PCNs is the idea that the introduction of certain flows can actively harm the throughput achieved by other flows in the network. To see this, consider the topology and demand rates in Figure 3c. Suppose nodes 1 and 2 want to transmit 1-unit transactions to node 3 at rates of 1 and 2 units/second, respectively, and node 3 wants to send to node 1 at rate 2 units/sec.² Notice that the specified transaction rates are imbalanced: there is a net flow of funds out of node 2 and into nodes 1 and 3. Suppose the payment channels are initially balanced, with 10 units on each side and we only start out with flows between nodes 1 and 3. For this demand and topology, the system can achieve a total throughput of 2 units/sec by only having nodes 1 and 3 to send to each other at a rate of 1 unit/second.

However, once transactions from node 2 are introduced, this example achieves zero throughput at steady-state. The reason is that node 2 sends transactions to node 3 faster than its funds are being replenished, which reduces its funds to 0. Slowing down 2’s transactions would only delay this outcome. Since node 2 needs a positive balance to route transactions between nodes 1 and 3, the transactions between 1 and 3 cannot be processed, despite the endpoints having sufficient balance. The network finds itself in a *deadlock* that can only be resolved by node 2 replenishing its balance with an on-chain transaction.

Why these problems are difficult to solve. The above problems are difficult to solve in part because their effects are closely intertwined. For example, because poor routing and rate-control algorithms can cause channel imbalance, which in turn degrades throughput, it is difficult to isolate the effects of each. Similarly, simply replacing circuit switching with packet-switching gives limited benefits without a corresponding rate control and routing mechanism.

From a networking standpoint, PCNs are very different from traditional communication networks. In particular, payment channels do not behave like a standard communication link with a certain capacity, say in transactions per second. Instead the capacity of a channel in a certain direction depends on two factors normally not seen in communication networks. First, the rate that transactions are received in the reverse direction on that channel, because tokens cannot be sent faster on average in one direction than they arrive in the other direction. Second, the delay it takes for the final receiver of a transaction to receive it and send back the secret key unlocking the funds

²For simplicity, we show three nodes, but a node in this example could represent a cluster of many users who wish to transact at the rates shown in aggregate.

at a router (§2). Tokens that are “in flight”, i.e. for which a router is waiting for the key, cannot be used to service new transactions. Therefore the network’s capacity effectively depends on its delay, and queued up transactions at a depleted link can hold up funds from channels in other parts of the network until they time out. This leads to cascading effects that make congestion control particularly critical.

4 Packet-Switched PCN

Spider uses a packet-switched architecture that splits transactions into a series of independently routed *transaction-units*. Each transaction-unit transfers a small amount of money bounded by a *maximum-transaction-unit (MTU)* value. Packetizing transactions is inspired by packet switching for the Internet, which is more effective than circuit switching [39]. Note that splitting transactions does not compromise the security of payments; each transaction-unit can be created with an independent secret key. As receivers receive and acknowledge transaction-units, senders can selectively reveal secret keys only for acknowledged transaction-units (§2). Senders can also use proposals like Atomic Multi-Path Payments (AMP) [3] if they desire atomicity of transactions.

In Spider, payments transmitted by source *end-hosts* are forwarded to their destination end-hosts by *routers* within the PCN. Spider routers queue up transaction-units at a payment channel whenever the channel lacks the funds to forward them immediately. As a router receives funds from the other side of its payment channel, it uses these funds to forward transaction-units waiting in its queue. Current PCN implementations [13] do not queue transactions at routers—a transaction fails immediately if it encounters a channel with insufficient balance on its route. Thus, currently, even a temporary lack of channel balance can cause many transactions to fail, which Spider avoids.

5 Modeling Routing

A good routing protocol must satisfy the following objectives:

1. **Efficiency.** For a PCN with a fixed amount of escrowed capital, the aggregate transaction throughput achieved must be as high as possible.
2. **Fairness.** The throughput allocations to different users must be fair. Specifically, the system should not starve transactions of some users if there is capacity.

Low latency, a common goal in communication networks, is desirable but not a first order concern, as long as transaction latency on the PCN is significantly less than an on-chain transaction (which can take minutes to hours today). However, as mentioned previously (§3), very high latency could hurt the throughput of a PCN, and must therefore be avoided.

To formalize the routing problem, we consider a fluid model of the system in which payments are modeled as continuous “fluid flows” between users. This allows us to cast routing as an optimization problem and derive decentralized algorithms from it, analogous to the classical Network Utility Maximization (NUM) framework for data networks [43]. More specifically, for the fluid model we consider a PCN modeled as a graph $G(V, E)$ in which V denotes the set of nodes (i.e., end-hosts or routers), and E denotes the set of payment channels between them. For a path p , let x_p denote the (fluid) rate at which payments are sent along p from a source to a destination. The fluid rate captures the long-term average rate at which payments are made on the path.

For maximizing throughput efficiency, routing has to be done such that the total payment flow through each channel is as high as possible. However, routers have limited capital on their payment channels, which restricts the maximum rate at which funds can be routed (Fig. 3a). In particular, when transaction units are sent at a rate $x_{u,v}$ across a payment channel between u and v , and it takes Δ time units on average to receive the secret key from a destination once a payment is forwarded, then $x_{u,v}\Delta$ credits are locked (i.e., unavailable for use) at any point in time in the channel. This implies that the average rate of transactions (across both directions) on a payment channel cannot exceed $c_{u,v}/\Delta$. This leads to *capacity constraints* on channels.

Sustaining a flow in one direction through a payment channel requires funds to be regularly replenished from the other direction. This requirement is a key difference between PCNs and traditional data networks. In PCNs if the long-term rates $x_{u,v}$ and $x_{v,u}$ are mismatched on a channel (u, v) , say $x_{u,v} > x_{v,u}$, then over time all the funds $c_{u,v}$ will accumulate at v deeming the channel unusable in the direction u to v (Fig. 3b). This leads to *balance constraints* which stipulate that the total rate at which transaction units are sent in one direction along a payment channel matches the total rate in the reverse direction.

Lastly, for enforcing fairness across flows we assume sources have an intrinsic *utility* for making payments, which they seek to maximize. A common model for utility at a source is the logarithm of the total rate at which payments are sent from the source [37, 36, 30]. A logarithmic utility ensures that the rate allocations are proportionally fair [37]—no individual sender’s payments can be completely throttled. Maximizing the overall utility across all source-destination pairs subject to the capacity and balance constraints discussed above, can then be computed as

$$\text{maximize} \quad \sum_{i,j \in V} \log \left(\sum_{p \in \mathcal{P}_{i,j}} x_p \right) \quad (1)$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{P}_{i,j}} x_p \leq d_{i,j} \quad \forall i, j \in V \quad (2)$$

$$x_{u,v} + x_{v,u} \leq \frac{c_{u,v}}{\Delta} \quad \forall (u, v) \in E \quad (3)$$

$$x_{u,v} = x_{v,u} \quad \forall (u, v) \in E \quad (4)$$

$$x_p \geq 0 \quad \forall p \in \mathcal{P}, \quad (5)$$

where for a source i and destination j , $\mathcal{P}_{i,j}$ is the set of all paths from i to j , $d_{i,j}$ is the demand from i to j , $x_{u,v}$ is the total flow going from u to v for a channel (u, v) , $c_{u,v}$ is the total amount of funds escrowed into (u, v) , Δ is the average round-trip time of the network taken for a payment to be completed, and \mathcal{P} is the set of all paths. Equation (2) specifies *demand constraints* which ensures that the total flow for each sender-receiver pair across all of their paths, is no more than their demand.

5.1 Implications for Throughput

A consequence of the balance constraints is that certain traffic demands are more efficient to route than certain others. In particular, demands that have a *circulation* structure (total outgoing demand matches total incoming demand at a router) can be routed efficiently. The cyclic structure of such demands enables routing along paths such that the rates are naturally balanced in channels. However, for demands without a circulation structure, i.e., if the demand graph is a directed acyclic

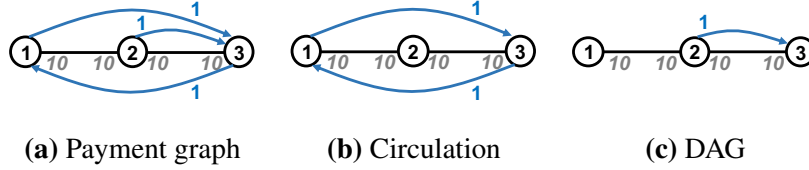


Figure 4: Payment graph (denoted by blue lines) for a 3 node network (left). It decomposes into a maximum circulation and DAG components as shown in (b) and (c).

graph (DAG), balanced routing is impossible to achieve in the absence of periodic replenishment of channel credits, regardless of how large the channel capacities are.

For instance, Fig. 4a shows the traffic demand graph for a PCN with nodes $\{1, 2, 3\}$ and payment channels between nodes 1 – 2 and 2 – 3. The weight on each blue edge denotes the demand in transaction-units per second between a pair of users. The underlying black lines denote the topology and channel sizes. Fig. 4b shows the circulation component of the demand in Fig. 4a. The entire demand contained in this circulation can be routed successfully as long as the network has sufficient capacity. In this case, if the confirmation latency for transaction-units between 1 and 3 is less than 10s, then the circulation demand can be satisfied indefinitely. The remaining component of the demand graph, which represents the DAG, is shown in Fig. 4c. This portion cannot be routed indefinitely since it shifts all tokens onto node 3 after which the 2 – 3 channel becomes unusable.

In Appendix A we formalize this notion of circulation and show that the maximum throughput achievable by any balanced routing scheme is at most the total demand contained within the circulation graph.

5.2 A Primal-Dual Decomposition Based Approach

We now describe a decentralized algorithm based on standard primal-dual decomposition techniques used in utility-maximization-based rate control and routing literature (e.g. [36]). §5.2.1 and §5.2.2 discuss the protocol at router nodes and end-hosts in order to facilitate this approach. A detailed derivation of the algorithm using in the fluid model that also considers the cost of on-chain rebalancing is discussed in Appendix D. However, §5.2.3 outlines the difficulties with this approach that lead us to the design of the practical protocol discussed in §6.

To arrive at this algorithm, we consider the optimization problem described in §5 for a generic utility function $U(\sum_{p \in \mathcal{P}_{i,j}} x_p)$. The structure of the Lagrangian of the LP allows us to naturally decompose the overall optimization problem into separate subproblems for each sender-receiver pair.

A solution to this LP can be computed via a decentralized algorithm in which each sender maintains rates at which payments are sent on each of its paths. Each payment channel has a *price* in each direction. Routers locally update these prices depending on both congestion and imbalance across the payment channels, while end-hosts adjust their rates by monitoring the total price on the different paths. The primal variables of the LP represent the rate of payments on each path, and the dual variables represent the channel prices. While this approach has been used before [36], a key difference from prior work is the presence of price variables for link balance constraints in addition to the price variables for capacity constraints. This ensures that the price of a channel having a skewed balance is different in each direction, and helps steer the flow rates to counter the

skew.

5.2.1 Router Design

Routers in each payment channel under this algorithm maintain price variables, which are updated periodically based on the current arrival rate of transaction units in the channel, available channel balance, and the number of transaction units queued up at the routers. The price variables at the routers determine the path prices, which in turn affect the rates at which end-hosts transmit transaction units (we discuss more in §5.2.2).

In a payment channel $(u, v) \in E$, routers u and v hold estimates for three types of price variables: $\lambda_{u,v}$, $\mu_{u,v}$ and $\mu_{v,u}$. These are dual variables corresponding to the capacity and imbalance constraints in Equation (4) and (3) respectively. The *capacity price* $\lambda_{u,v}$ signals congestion in the channel if the total rate at which transactions arrive exceeds its capacity; the *congestion prices* $\mu_{u,v}$ and $\mu_{v,u}$ are used to signal imbalance in the channel in the two directions. These variables are updated periodically to ensure that the capacity and imbalance conditions are not violated in the channel. Prices are updated every τ seconds according to the rules described next.

Imbalance Price. For a channel (u, v) , let n_u, n_v denote the total amount of transactions that have arrived at u and v respectively, in the τ seconds since the last price update. Let q_u, q_v be the total amount of payments that are currently queued up at u and v respectively. The price variable for imbalance $\mu_{(u,v)}$ is updated as

$$\mu_{u,v}(t+1) = \left[\mu_{u,v}(t) + \kappa \left(n_u(t) - n_v(t) + \frac{q_u(t)}{T} \tau - \frac{q_v(t)}{T} \tau \right) \right]_+, \quad (6)$$

where κ is a positive step-size parameter for controlling the rate at which the price varies and T is a parameter for controlling the sensitivity of the prices to queue size.³ Intuitively, if more funds arrive in the u - v direction compared to the v - u direction (i.e., $n_u(t) > n_v(t)$), the price $\mu_{u,v}$ increases while the price $\mu_{v,u}$ decreases. The higher price in the u - v direction signals end-hosts that are routing along (u, v) to throttle their rates, and signal end-hosts routing along (v, u) to increase their rates. Similarly, if there is queue buildup on one side due to an imbalance in the rates, Equation (6) causes the price to increase in the direction where there is greater queue buildup. This in turn would reduce the arrival rate in that direction, and cause the queue to drain.

Capacity Price. The price variable for capacity $\lambda_{u,v}$ is also updated every τ seconds as follows:

$$\lambda_{u,v}(t+1) = [\lambda_{u,v}(t) + \eta (m_u(t) + m_v(t) - c_{u,v} + \beta \min(q_u(t), q_v(t)))]_+. \quad (7)$$

For the current rates of transaction arrival at u and v , $m_u(t)$ and $m_v(t)$ are estimates of the amount of funds required to sustain those rates at u and v respectively. Since $c_{u,v}$ is the total amount of funds available in the channel, any excess in required amount of funds compared to $c_{u,v}$ would cause $\lambda_{u,v}$ to rise and vice-versa. An increase in $\lambda_{u,v}$ signals end-hosts routing via u, v , on either direction, to reduce their rates. We estimate the demands $m_u(t)$ and $m_v(t)$ for tokens by measuring the arrival and service rates of transactions, and the current amount of locked funds in the channel. $\lambda_{u,v}$ also increases if there is queue buildup simultaneously on both u and v , as it implies the total

³The price update for $\mu_{v,u}$ is analogous to Eq. (6), but with u and v interchanged. This equation can be modified to include on-chain rebalancing rates on both ends.

demand placed on the channel is exceeding the available channel capacity. β is a positive parameter for controlling the sensitivity of prices to queue buildup, and η is a positive step-size parameter.

5.2.2 End-host Design

Spider-hosts run a multi-path transport protocol with pre-determined paths which controls the rates at which payments are transferred, based on observations of the channel prices or router feedback. End-hosts here use probe messages to evaluate the channel prices on each path. The total price of a path p is given by

$$z_p = \sum_{(u,v):(u,v) \in p} (2\lambda_{u,v} + \mu_{u,v} - \mu_{v,u}), \quad (8)$$

which captures the aggregate amount of imbalance and excess demand, as signaled by the corresponding price variables, in the path. We refer to Appendix D for a mathematical intuition behind Equation (8). Probes are sent periodically every τ seconds (i.e., the same frequency at which channel prices are updated §6.2) on each path. A probe sent out on path p sums the price $2\lambda_{u,v} + \mu_{u,v} - \mu_{v,u}$ of each channel (u, v) it visits, until it reaches the destination host on p . The destination host then transmits the probe back to the sender along the reverse path. The rate to send x_p on each path p is updated using the path price z_p from the most recently received probe as

$$x_p(t+1) = x_p(t) + \alpha(U'(x) - z_p(t)), \quad (9)$$

where α is a positive step-size parameter. Thus the rate to send on a path decreases if the path price is high—indicating a large amount of imbalance or capacity deficit in the path—and increases otherwise.

5.2.3 Challenges

There are a number of challenges in making this algorithm work in practice. Firstly, iterative algorithms for adjusting path prices and sending rates suffer from slow convergence. An algorithm that is slow to converge may not be able to adapt the routing to changes in the transaction arrival pattern. If the transaction arrival patterns change frequently, this may result in a perpetually sub-optimal routing. Secondly, in order to compute the imbalance prices, the two routers in a payment channel need to exchange information about their arrival patterns and their respective queue states to calculate n_u and m_u in Eq.6–7. This implies that routers cannot deploy this in isolation. Further, we found that the scheme was extremely sensitive to the many parameters involved in the algorithm, making it hard to tune for a variety of topologies and capacity distributions. Lastly, a pure pacing based approach could cause bursts in transaction-units sent that lead to large queue buildups much before the prices react appropriately. To account for this, the algorithm needs to be augmented with windows [35] and active queue control to work in practice. Due to these difficulties, we propose a more practical and simpler protocol described in §6.

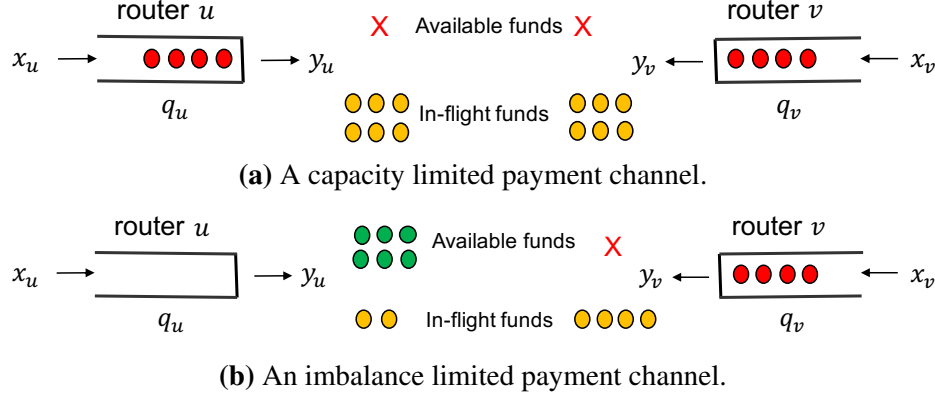


Figure 5: Example of queue growth in a payment channel between routers u and v , under different scenarios of transaction arrival rates at u and v . (a) If the rate of arrival at v , x_v , and the rate of arrival at u , x_u , are such that their sum is greater than the capacity of the channel, then there are no available funds at either router and a queue builds up at both u and v . (b) If the arrival rates are imbalanced, e.g., if $x_v > x_u$, then u has excess available funds while v has none, and therefore a queue builds up at v .

6 Design

6.1 Intuition

Spider routers queue up transactions at a payment channel whenever the channel lacks the funds to forward the transactions immediately (§5). Thus, queue buildup is a sign that either transaction-units are arriving faster (in both directions) than the channel can process (Fig. 5a) or that there are insufficient funds at one of the two ends of a payment channel (Fig. 5b). In other words, it suggests that either the capacity constraint (Equation 3) or the balance constraint (Equation 4) is being violated and that the sender should adjust its sending rate accordingly.

Therefore, if senders use a congestion control protocol that controls queues, it could in principle detect both capacity and imbalance violations and react to them. For example, in Fig. 5a, the congestion control protocol would throttle both x_u and x_v , and in Fig. 5b, it would throttle x_v . In the latter case, x_v would decrease until it matches the rate at which queue q_v drains, which is precisely x_u , the rate at which new funds become available at router v .

This example illustrates that a congestion controller that satisfies two basic properties can achieve both efficiency and balanced rates:

1. *Keeping queues non-empty*, which ensures that any available capacity is being utilized, i.e., there are no unused tokens at any router.
2. *Keeping queues stable (bounded)*, which ensures that (a) the flow rates do not exceed a channel's capacity, (b) the flow rates are balanced. If either condition is violated, then at least one of the channel's queues would grow.

Congestion control algorithms that satisfy these properties abound (e.g., Reno [17], Cubic [33], DCTCP [21], Vegas [26], etc.) and could be adapted for PCNs.

To optimize throughput, it is desirable in PCNs to transmit transaction-units along multiple paths to better utilize available network capacity. Thus in the design of Spider, we are inspired by multipath transport protocols like MPTCP [56]. These protocols couple rate control decisions

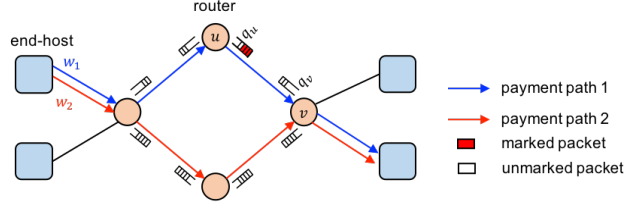


Figure 6: Routers queue up transaction-units and schedule them across payment channels based on available capacity and transaction priorities. If the delay through the queue for a packet exceeds a threshold, they mark the packet. End-hosts maintain windows for each path to a receiver which are adjusted based on whether acked transaction-units are marked.

for multiple paths to achieve both high throughput and fairness among competing flows [55]. We describe an MPTCP-like protocol for PCNs in §6.2–6.3. In §6.4 we show that the rates found by Spider’s protocol for parallel network topologies, match the solution to the optimization problem in §5.

6.2 Spider Router Design

Fig. 6 shows a schematic diagram of the various components in the Spider PCN. Spider routers monitor the time that each packet spends in their queue and mark the packet if the time spent is larger than a pre-determined threshold T . If the transaction-unit is already marked, routers leave the field unchanged and merely forward the transaction-unit. When the end-host sends back an acknowledgement with the *marked* field appropriately set, Spider routers forward it back to the sender which interprets it accordingly.

6.3 Spider Transport Layer at End-Hosts

Spider-hosts run a multi-path transport protocol which controls the rates at which payments are transferred, based on observations of congestion in the network. For each destination host, a sender chooses a set of k paths along which to route transaction-units. The route a transaction-unit takes is decided upfront at the sender before transmitting the unit. It is written into the transaction-unit using onion encryption, to hide the full route from intermediate routers and provide privacy [31, 15]. In §7.5, we provide a detailed evaluation of the impact of different path choices on Spider’s performance. Based on these results, we propose using edge-disjoint widest paths [20] between each sender and receiver in Spider.

In order to control the rate at which payments are sent on a path, end-hosts maintain a window size w_p for every candidate path to a destination. This window size denotes the maximum number of transaction-units that can be outstanding on path p at any point in time. End-hosts track the transaction-units that have been sent out on each path but have not been acked or canceled yet. A new transaction-unit is transmitted on a path p only if the total amount pending does not exceed w_p .

End-hosts adjust w_p based on router feedback on congestion and imbalance. In particular, on a

path p between source i and receiver j the window changes as

$$w_p \leftarrow w_p - \beta, \quad \text{on every marked packet and,} \quad (10)$$

$$w_p \leftarrow w_p + \frac{\alpha}{\sum_{p': p' \in \mathcal{P}_{i,j}} w_{p'}}, \quad \text{on every unmarked packet.} \quad (11)$$

We expect the application to specify a deadline for every transaction. If the transport layer fails to complete the payment within the deadline, the sender chooses to give up and cancels the payment, clearing all of its state from the PCN. Such canceled packets are considered marked and Spider decreases its window in response to them. Here, α and β are both positive constants that denote the aggressiveness with which the window size is increased and decreased respectively. Eq. (10)–(11) are similar to MPTCP, but with a multiplicative decrease factor that depends on the fraction of packets marked on a path (similar to DCTCP [21]).

6.4 Optimality of Spider

Under a fluid approximation model for Spider’s dynamics, we can show that the rates computed by Spider are an optimal solution to the routing problem in Equations (1)–(5) for parallel networks (such as Fig. 19 in Appendix B). In the fluid model, we let $x_p(t)$ denote the rate of flow on a path p at time t ; for a channel (u, v) , $f_{u,v}(t)$ denotes the fraction of packets that are marked at router u as a result of excessive queueing. The dynamics of the flow rates $x_p(t)$ and marking fractions $f_{u,v}(t)$ can be specified using differential equations to approximate the window update dynamics in Equations (10) and (11). We elaborate more on this fluid model, including specifying how the queue sizes and marking fractions evolve, in Appendix B.

Now, consider the routing optimization problem (Equations (1)–(5)) written in the context of a parallel network. If Spider is used on this network, we can show that there is a mapping from the rates $\{x_p\}$ and marking fractions $\{f_{u,v}\}$ values after convergence, to the primal and dual variables of the optimization problem, such that the Karush-Kuhn-Tucker (KKT) conditions for the optimization problem are satisfied. This proves that the set of rates found by Spider is an optimal solution to the optimization problem [25]. The complete and formal mathematical proof showing the above is presented in Appendix B.

7 Evaluation

We develop an event-based simulator for PCNs, and use it to extensively evaluate Spider across a wide range of scenarios. We begin by describing our simulation setup (§7.1), followed by presenting detailed results for circulation demands (§7.3). We then show the effect of adding DAG components to circulation payment graphs (§7.4), and study the impact Spider’s design choices (§7.5).

7.1 Experimental Setup

Simulator. We extend the OMNET++ simulator (v5.4.1) [1] to model a PCN. Our simulator accurately models the network-wide effects of transaction processing, by explicitly passing messages

between PCN nodes (endhosts and routers).⁴ Each endhost (i) generates transactions destined for other endhosts as per the specified workload, and (ii) determines when to send a transaction and along which path, as per the specified routing scheme. All endhosts maintain a view of the entire PCN topology, to compute suitable source-routes. The endhosts do not have access to channel balances, but they do know each channel’s size.⁵ Endhosts also split generated transactions into MTU-sized segments (or transaction-units) before routing, if required by the routing scheme (e.g. by Spider). Each generated transaction has a *timeout* value and is marked as a failure if it fails to reach its destination by then. Upon receiving a transaction, an endhost generates an acknowledgment that is source-routed along its reverse path.

A router forwards incoming transactions and acknowledgments along the payment channels specified in their source-routes, while correspondingly decrementing or incrementing the channel balances. Funds consumed by a transaction in a channel are *inflight* and unavailable between when the transaction is processed and its acknowledgment is received. A transaction is forwarded on a payment channel only if the channel has sufficient balance; otherwise the transaction is stored in a *per-channel queue*. The queue is serviced in a last in first out (LIFO) order when balance becomes available. If the queue is full, an incoming transaction is dropped, and a message is sent to the sender, indicating that the transaction has failed.

Simulator Validation We validate our simulator via a prototype implementation on Lightning Network Daemon (LND) that we detail in §7.2.

Routing Schemes. We implement and evaluate five different routing schemes in our simulator.

(1) *Spider*: Every Spider sender maintains a set of up to k edge-disjoint widest paths to each destination and a window size per path. The sender splits transactions into transaction-units and sends a transaction-unit on a path if the path’s window is larger than the path’s amount of inflight units. If a transaction-unit cannot be sent, it is placed in a per-destination queue at the sender that is served in LIFO order. Spider routers mark transaction-units experiencing delays higher than a pre-determined threshold. Spider receivers echo the mark back to senders who adjust the window size according to the equations in §6.3.

(2) *Waterfilling*: We also develop an explicit balance-aware routing scheme to contrast against Spider’s simpler 1-bit feedback. As with Spider, a sender splits transactions into transaction-units and picks up to k edge-disjoint widest paths per destination. It maintains one outstanding probe along each path that computes the bottleneck (minimum) channel balance along the path. When a path’s probe is received, the sender subtracts the number of in-flight transaction-units along the path from its bottleneck balance, to obtain the available path balance. A transaction-unit is sent along the path with the highest available balance among the k choices. If the available balance for all of the k paths is zero (or less), the transaction segment is queued and retried the next time a probe comes back.

(3) *Shortest Path*: This naïve baseline sends transactions along the shortest path to the destination without transaction splitting.

(4) *Landmark Routing*: Landmark routing is used in many prior PCN routing schemes [45, 40, 48]. This scheme chooses k well-connected *landmark* nodes in the topology. Every sender computes its shortest path to each landmark and concatenates it with the shortest path from that landmark to the

⁴We intend to make our simulator publicly available.

⁵We distinguish channel *size*, denoting the number of tokens in € in a payment channel, from *capacity* in €/s , denoting the maximum rate at which transaction-units can be successfully transmitted across a payment channel.

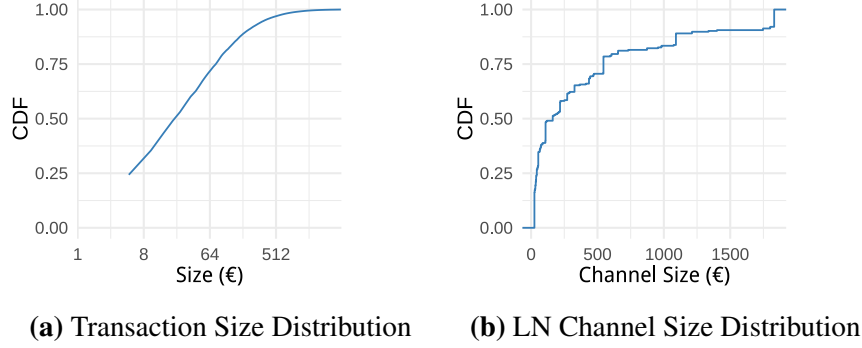


Figure 7: Transaction dataset and channel size distribution used for real-world evaluations.

destination to obtain k distinct paths. Then, for every transaction, the sender sends out a probe on each of the k paths to obtain its bottleneck channel balance, and partitions each transaction across the k paths such that each path can support its share of the total transaction. If such a partition does not exist or if any of the partitions fail to complete, the transaction fails.

(5) *LND*: We also compare against a PCN scheme currently deployed in the Lightning Network Daemon (LND) [13]. A transaction is first sent along the shortest path to its destination. If the transaction fails due to insufficient balance at a channel, the sender removes that channel from its local view, recomputes the shortest path, and retries the transaction on the new path. This process is repeated until the destination becomes unreachable or the transaction times out. Both are treated as failures. A channel is added back to the local view 5 seconds after its removal.

Workload. We generate two forms of payment graphs to specify the rate at which a sender transacts with every other receiver: (i) pure circulations, with a fixed total sending rate x per sender. The traffic demand matrix for this is generated by adding x random permutation matrices; (ii) circulations with a DAG component, having a total rate y . This type of demand is generated by sampling y different sender-receiver pairs where the senders and receivers are chosen from two separate exponential distributions (so that some nodes are more likely than others to be picked as a sender or receiver). The scale β of the distribution is set proportional to the desired percentage of DAG component in the total traffic matrix: the greater the fraction of DAG component desired, the more skewed the distribution becomes. We translate the rates specified in the payment graphs to discrete transactions with Poisson inter-arrival times. The transaction size distribution is drawn from credit card transaction data [32], and has a mean of 88€ and median 25€ with the largest transaction being 3930€. The distribution of transaction sizes is shown in Fig. 7a. We keep the sending rates constant at an average of 30 tx/sec per sender that is shared among 10 destinations throughout all of our experiments. Note that a sender represents a router in these experiments, sending transactions to other routers on behalf of many users.

Topology. We set up an LND node [13] to retrieve the the Lightning Network topology on July 15, 2019. We snowball sample [34] the full topology (which has over 5000 nodes and 34000 edges), resulting in a PCN with 106 nodes and 265 payment channels. For compatibility with our transaction dataset, we convert LND payment channel sizes from Satoshis to €, and cap the minimum channel size to the median transaction size of 25€. The distribution of channel sizes for this topology has a mean and median size of 421€ and 163€ respectively (Fig. 7b). This distribution is highly skewed, resulting in a mean that is much larger than the median or the smallest

payment channels. We refer to this distribution as the Lightning Channel Size Distribution (LCSD). We draw channel propagation delays based on ping times from our LND node to all reachable nodes in the Lightning Network, resulting in transaction minimum RTTs of about a second.

We additionally simulate two synthetic topologies: a Watts-Strogatz small world topology [18] with 50 nodes and 200 edges, and a scale-free Barabasi-Albert graph [19] with 50 nodes and 336 edges. We set the per-hop delay to 30ms in both cases, resulting in an end-to-end minimum RTT of 200-300ms.

For payment channel sizes, we use real capacities in the Lightning topology and sample capacities from LCSD for synthetic topologies. We vary the mean channel size across different experiments by proportionally scaling up the size of each payment channel. All payment channels are perfectly balanced at the start of the experiment.

Parameters. We set the MTU as 1€. Every transaction has a timeout of 5 seconds. Schemes with router queues enabled have a per-channel queue size of 12000€. The number of path choices is set to $k = 4$ for schemes that use multiple paths. We vary both the number of paths and the nature of paths in §7.5. For Spider, we set α (window increase factor) to 10, β (multiplicative decrease factor) to 0.1, and the marking threshold for the queue delay to 300ms. For the experiments in §7.4, we set this threshold to 75ms so as to respond to queue buildup faster than per-RTT.

Metrics. We use the following three evaluation metrics: (i) *transaction success ratio*: the number of completed transactions over the number of generated transactions. A transaction which is split at the source is complete when all of its split pieces successfully reach the destination, and (ii) *normalized throughput*: the total amount of payments (in €) completed over the total amount of payments generated. All of these metrics are computed over a measurement interval, set such that all algorithms are in their steady-state. Unless otherwise specified, we use a measurement interval of 800-1000s when running an experiment for 1010s.

7.2 Prototype Implementation

To support Spider, we modify the Lightning Network Daemon (LND) [13] which is currently deployed on the live Bitcoin Network. We repurpose the router queues to queue up transactions (or HTLCs) that cannot be immediately serviced. When a transaction spends more than 75ms in the queue, Spider marks it. The marking is echoed back via an additional field in the transaction acknowledgement (`FulfillHTLC`) to the sender. We maintain a per-receiver state at the sender to capture the window and number in-flight on each path, as well as the queue of unattempted transactions. Each sender finds 4 edge-disjoint shortest paths to every destination. We do not implement transaction-splitting.

We deploy our modified LND implementation [13] on Amazon EC2’s `c5d.4xlarge` instances with 16 CPU cores, 16 GB of RAM, 400 GB of NVMe SSD, and a 10 Gbps network interface. Each instance hosts one end-host and one router. Every LND node is run within a docker container with a dedicated bitcoin daemon [5]. We create our own regtest [7] blockchain for all the nodes. Channels are created on this regtest corresponding to a scale-free graph with 10 nodes and 25 edges. We vary the mean channel size from 25€ to 400€. Five circulation payment graphs with are generated at an average rate of 100 tx/s (each 1€) per sender. Receiving nodes communicate invoices via `etcd` [11] to sending nodes who then complete them using the appropriate protocol. We run the LND baseline and Spider under this setting on the implementation, and measure the observed transaction RTTs. We use this measurement to set the propagation delay on

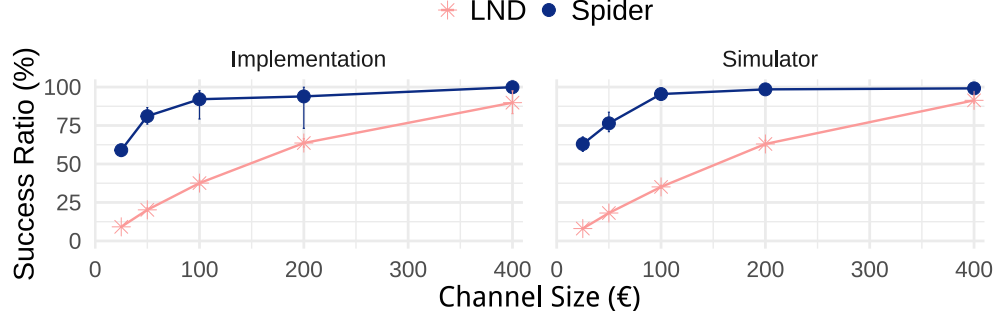


Figure 8: Comparison of performance on simulator and implementation for LND and Spider on a 10 node scale-free topology with 1€ transactions. Spider outperforms LND in both settings. Further, the average success ratio on the simulator and implementation for both schemes are within 5% of each other. Error-bars denote the maximum and minimum success ratio across runs.

the simulator, and then run the same experiments on the implementation.

Fig. 8 shows the average success ratio that Spider and LND achieve on the implementation and the simulator. There are two important takeaways: (i) Spider outperforms LND in both settings and, (ii) the average success ratio on the simulator is within 5% of the implementation for both schemes. Our attempts at running experiments at larger scale showed that the LND codebase is not optimized to sustain high throughput. For example, IO bottlenecks caused by persisting every HTLC state on disk causes variations on the order of few 10s of seconds in transaction completion times even on small topologies. Given the fidelity and flexibility of the simulator, we chose to use it for the rest of the evaluations.

7.3 Circulation Payment Graph Performance

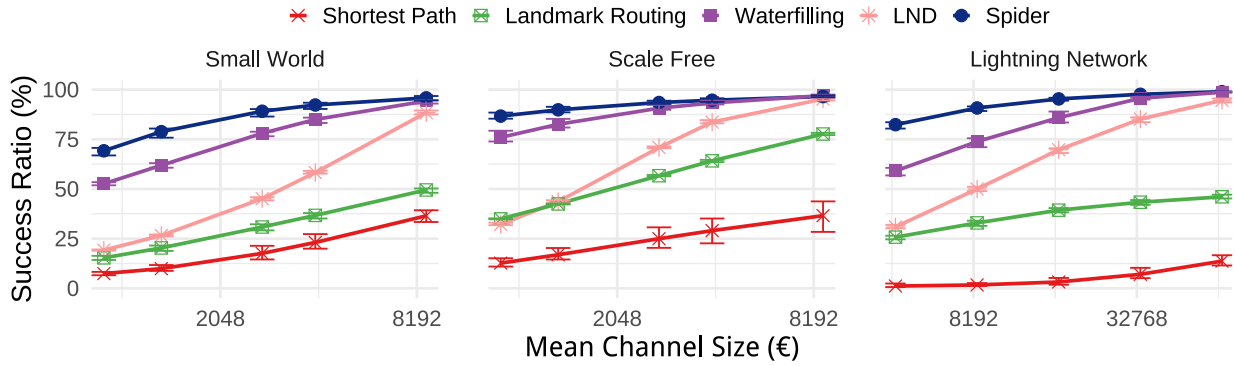


Figure 9: Performance of different algorithms on small-world, scale-free and Lightning Network topologies, for different per sender transaction arrival rates. Spider consistently outperforms all other schemes achieving near 100% average success ratio. Error-bars denote the maximum and minimum success ratio across five runs. Note the log scale of the x-axes.

Recall that on circulation payment graphs, *all* the demand can theoretically be routed if there is sufficient capacity (§5.1 and Appendix A). However, the capacity at which a routing scheme

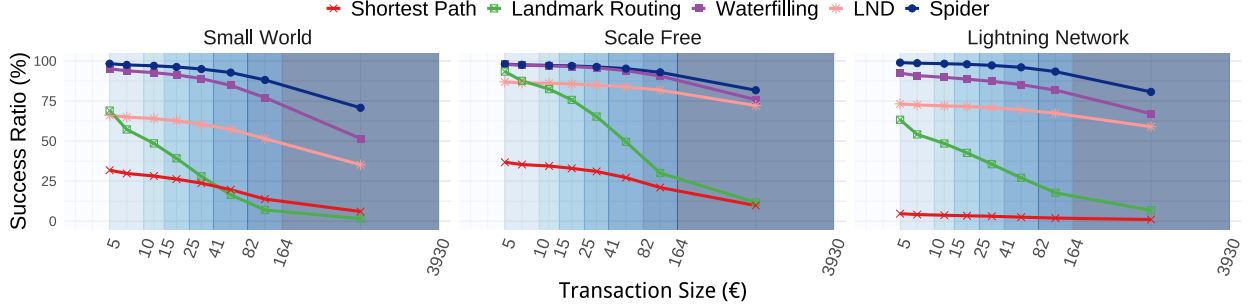


Figure 10: Breakdown of performance of different schemes by size of transactions completed. Each point reports the success ratio for transactions whose size belongs to the interval denoted by the shaded region. Each interval corresponds roughly to a 12.5% weight in the transaction size CDF shown in Fig. 7a. The graphs correspond to the midpoints of the corresponding Lightning sampled channel sizes in Fig. 9.

attains 100% throughput depends on the scheme’s ability to balance channels: the more balanced a scheme is, the less capacity it needs for high throughput.

Efficiency of Routing Schemes. We run five circulation traffic matrices on our three topologies (§7.1). Notice that the channel sizes are much larger on the Lightning Topology compared to the other two due to the highly skewed nature of capacities Fig. 7b. We measure the average success ratio for the transactions across different payment channel sizes. Fig. 9 shows that across all topologies, Spider outperforms the state-of-the-art schemes. Spider is able to successfully route more than 95% of the transactions with less than 25% of the capacity required by LND. At lower capacities, Spider successfully routes 2-3 \times more transactions than LND. This is because Spider maintains balance in the network by responding quickly to queue buildup at payment channels, thus making better use of network capacity. As the Lightning Network is still in early stages of deployment, channels are likely to be small in size and a 50% increase in success ratio marks a very significant improvement for PCNs. Note that the explicit balance-aware scheme, Waterfilling, is also able to route many more transactions than LND.

Size of Successful Payments. Spider’s benefits are most pronounced at larger transaction sizes, where transaction splitting and congestion control enables more successful transactions. To demonstrate this, Fig. 10 shows success ratio as a function of transaction size. We use mean channel sizes of 4000€ and 16880 € for the synthetic and real topologies, respectively. Each shaded region denotes a different range of transaction sizes, and a workload contains roughly an equal number of transactions under each range. A point within a range represents the average success ratio for transactions in that interval across 5 runs. Spider outperforms LND across all sizes, and is able to route 5-30% more of the largest transactions compared to LND.

7.4 Effect of DAGs

Real transaction demands are often not pure circulations: consumer nodes spend more, and merchant nodes receive more. To simulate this, we add 5 DAG payment graphs (§7.1) to circulation payment graphs, varying the relative weight to generate effectively 5%, 20% and 40% DAG in the total demand matrix.

We run all schemes on the Lightning topology with a mean channel size of 16880€; results on the synthetic topologies are in Appendix G.

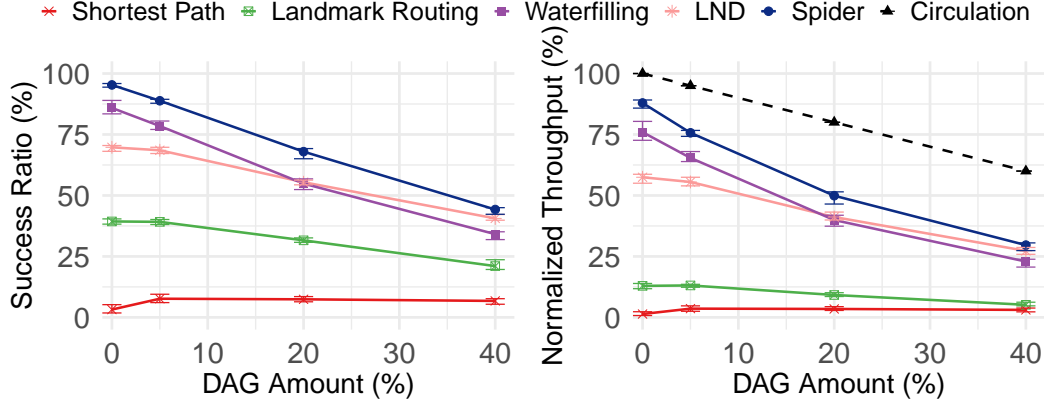


Figure 11: Performance of different algorithms on the Lightning topology as the DAG component in the transaction demand matrix is varied. As the DAG amount is increased, the normalized throughput achieved is further away from the expected optimal circulation throughput. Error-bars denote the maximum and minimum success ratio across five runs.

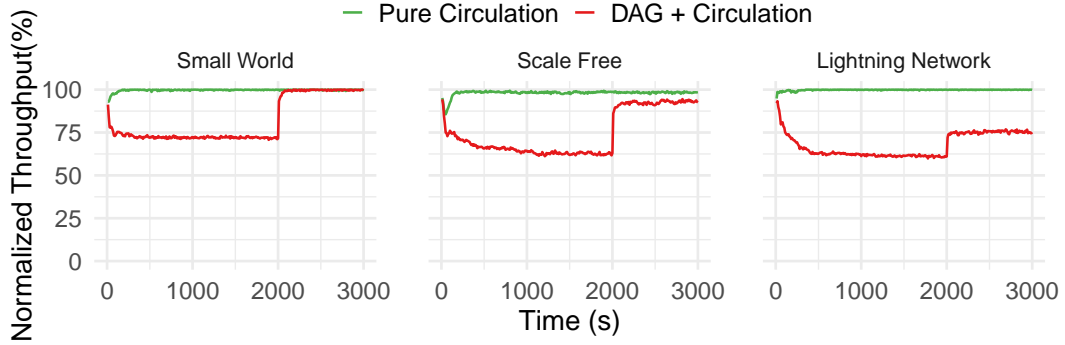


Figure 12: Comparing throughput when a pure circulation demand is run for 3000s to a scenario where a circulation demand is restored for 1000s after 2000s of a demand with 20% DAG. The throughput achieved on the last 1000s of circulation is not always the expected 100% even after the DAG is removed.

Fig. 11 shows the success ratio and normalized throughput. We immediately notice that no scheme achieves the theoretical upper bound on throughput (i.e., the % circulation demand). However, throughput is closer to the bound when there is a smaller DAG component in the demand matrix. This suggests that not only is the DAG itself unroutable, it also alters the PCN balances in a way that prevents the circulation from being fully routed. Further, the more DAG there is, the more affected the circulation is. This is because the DAG causes a deadlock (§3).

To illustrate this, we run two scenarios: (i) a pure circulation demand X for 3000s, and (ii) a traffic demand $(X + Y)$ containing 20% DAG for 2000s followed by the circulation X for 1000s after that. Here, each sender sends 200€/s of unit-sized transactions in X . We observe a time series of the normalized throughput over the 3000s. The mean channel size is 4000€ and 16990€ for the synthetic and real topologies respectively.

Fig. 12 shows that Spider achieves 100% throughput (normalized by the circulation demand) at steady state for the pure circulation demand on all topologies. However, when the DAG component is introduced to the demand, it affects the topologies differently. Firstly, we do not observe the

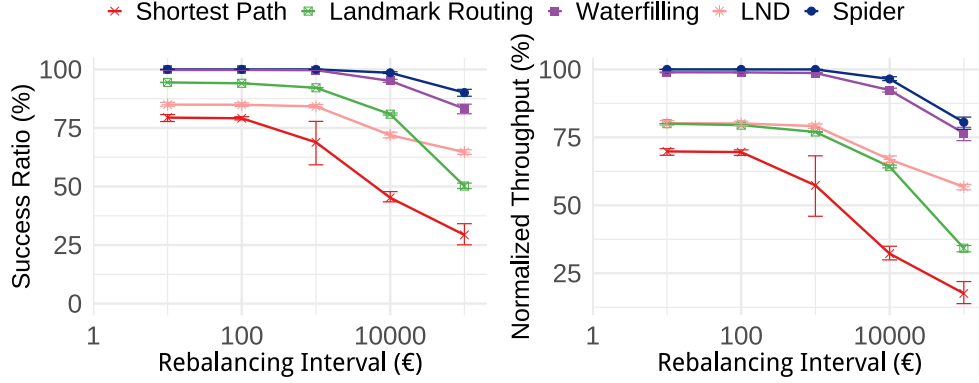


Figure 13: Performance of different algorithms on the Lightning Topology when augmented with a rebalancing scheme that moves funds from channels with unused capacity to channels needing them. Spider continues to outperform other schemes, needing less frequent rebalancing to sustain high throughput. Error-bars denote the maximum and minimum statistic across five runs.

expected 80% throughput for the circulation in the presence of the DAG workload suggesting that the DAG affects the circulation. Further, even once the circulation demand is restored for the last 1000s, in the scale free and Lightning Network topology, the throughput achieved is no longer 100%. In other words, in these two topologies, the DAG causes a deadlock that affects the circulation even after the DAG is removed.

As described in §3, the solution to this problem involves replenishing funds via on-chain rebalancing, since DAG demands continuously move money from sources to sinks. We therefore implement a simple rebalancing scheme where every router periodically reallocates funds between its payment channels to equalize their *available balance*. The frequency of rebalancing for a router, is defined by the number of successful transaction-units (in €) between consecutive rebalancing events. In this model, the frequency captures the on-chain rebalancing cost vs. routing fee tradeoff for the router.

Fig. 13 shows the success ratio and normalized throughput achieved by different schemes when rebalancing is enabled for the traffic demand with 20% DAG from Fig. 11, or Fig. 12. Spider is able to achieve 90% success ratio even when its routers rebalance only every 10,000€ routed while LND is never able to sustain more than 85% success ratio even when rebalancing for every 10€ routed. This is because LND deems a channel unusable for 5 seconds every time a transaction fails on it due to lack of funds and this is further worsened by its lack of transaction splitting. This implies that when using Spider, routers need to pay for only one on-chain transaction typically costing under 1€ [6] for every 10,000€ routed. Thus, for a router to break even, it would have to charge 1€ for every 10000€ routed. This translates into significantly lower routing fees for end-users than today’s payment systems [10].

7.5 Spider’s Design Choices

In this section, we investigate Spider’s design choices with respect to the number of paths, type of paths, and the scheduling algorithm that services transaction-units at Spider’s queues. We evaluate these on both the real and synthetic topologies with channel sizes sampled from the LCSD, and

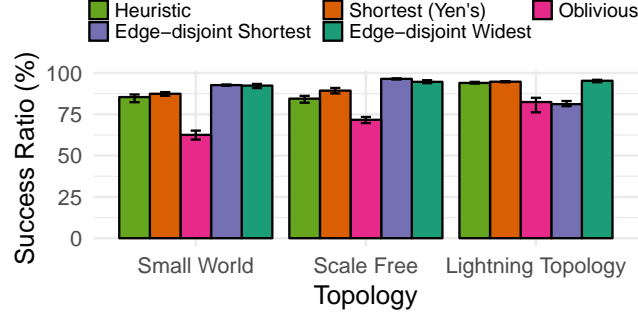


Figure 14: Performance of Spider as the type of paths considered per sender-receiver pair is varied. Edge-disjoint widest outperforms others by 1-10% on the Lightning Topology without being much worse on the synthetic topologies. Error-bars denote the maximum and minimum success ratio across five runs.

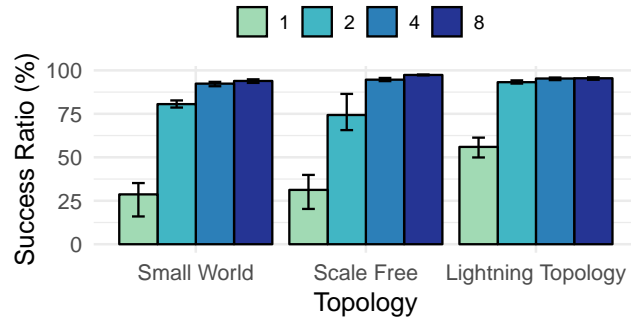


Figure 15: Performance of Spider as the number of edge-disjoint widest paths considered per sender-receiver pair is varied on different topologies. Increasing the number of paths increases success ratio, but the gains are low in going from 4 to 8 paths. Error-bars denote the maximum and minimum success ratio across five runs.

scaled to have mean of 16880€ and 4000 € respectively .

Choice of Paths. We vary the type of paths that Spider uses by replacing edge-disjoint widest paths with edge-disjoint shortest paths, Yen’s shortest paths [57], oblivious paths [46] and a heuristic approach. For the widest and oblivious path computations, the channel size acts as the edge weight. The heuristic picks 4 paths for each flow with the highest bottleneck balance/RTT value. Fig. 14 shows that edge-disjoint widest paths outperforms other approaches by 1-10% on the Lightning Topology while being only 1-2% worse than edge-disjoint shortest paths on the synthetic topologies. This is because widest paths are able to utilize the capacity of the network better when there is a large skew (Fig. 7b) in payment channel sizes.

Number of Paths. We vary the maximum number of edge-disjoint widest paths Spider allows from 1 to 8. Fig. 15 shows that, as expected, the success ratio increases with an increase in number of paths, as more paths allow Spider to better utilize the capacity of the PCN. While moving from 1 to 2 paths results in upto 50% improvement in success ratio, moving from 4 to 8 paths has negligible benefits (<5%). This is because the sparseness of the three PCN topologies causes most flows to have at most 5-6 edge-disjoint widest paths. Further, Spider prefers paths with smaller RTTs since they receive feedback faster resulting in the shortest paths contributing most to the overall rate for the flow. As a result, we use 4 paths for Spider.

Scheduling Algorithms. We modify the scheduling algorithm at the per-destination queues at

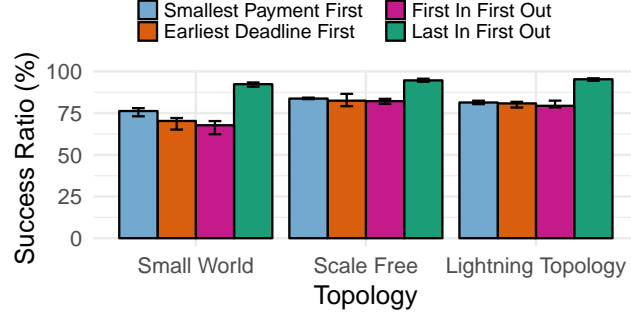


Figure 16: Performance of Spider as the scheduling algorithm at the sender and router queues is varied. Last in first out outperforms all other approaches by over 10% on all topologies. Error-bars denote the maximum and minimum success ratio across five runs.

the sender as well as the router queues in Spider to process transactions as per first in first out (FIFO), earliest deadline first (EDF) and smallest payments first (SPF) in addition to the LIFO baseline. Fig. 16 shows that LIFO achieves a success ratio that is 10-28% higher than its counterparts. This is because LIFO prioritizes transactions that are furthest from their deadlines and thus, most likely complete. Spider’s rate control results in long wait times in the sender queues themselves. This causes FIFO and EDF that send out transactions closest to their deadlines to time out immediately in the network resulting in poor throughput. When SPF deprioritizes large payments at router queues, they consume funds from other payment channels for longer, reducing the effective capacity of the network.

7.6 Additional Results

In addition to the results described so far, we run additional experiments that are described in the Appendices.

1. We evaluate the schemes on the synthetic and real topologies with a simpler channel size distribution where all channels have equal numbers of tokens. Even in this scenario, Spider is able to successfully route more than 95% of the transactions with less than 25% of the capacity required by LND (Appendix F.2).
2. We evaluate the schemes for their fairness across multiple payments and show that Spider does not hurt small payments to gain on throughput (Appendix F.1).
3. We show the effect of DAG workloads on synthetic topologies. In particular, we identify deadlocks with those topologies too and show that Spider requires rebalancing only every 10,000€ successfully routed to sustain high success ratio and normalized throughput (Appendix G).

8 Related Work

PCN Improvements. As described in §7.1, in current Lightning Network implementations, each node maintains a local view of the network topology and source-routes transactions [44]. Paths are chosen based on shortest path algorithms [13, 2]. Classical max-flow-based alternatives are impractical for the Lightning Network that has over 5000 nodes and 30,000 channels [14, 8] due

to their computational complexity. Recent proposals have used a modified version of max-flow that differentiates based on the size of transactions [54]. However, inferring the size of payments is hard in an onion-routed network like Lightning.

Two main alternatives to max-flow routing have been proposed: landmark routing and embedding-based routing. In *landmark routing*, select routers (landmarks) store routing tables for the rest of the network, and nodes need only route transactions to a landmark [52]. This approach is used in Flare [45] and SilentWhispers [40, 42]. *Embedding-based* or *distance-based* routing learns a vector embedding for each node, such that nodes that are close in network hop distance are also close in embedded space. Each node relays each transaction to the neighbor whose embedding is closest to the destination’s embedding. VOUTE [47] and SpeedyMurmurs [48] use embedding-based routing. Computing and updating the embedding dynamically as the topology and link balances change is a primary challenge of these approaches. Our experiments show that Spider outperforms landmark routing and prior work [22] has shown that embedding-based approaches tend to underperform even more.

A different class of PCN improvements instead considers ways to facilitate adding funds to existing payment channels. These techniques, including channel factories [27] and rebalancing techniques [38], are complementary to Spider and can be used to enhance overall performance.

Utility Maximization and Congestion Control. Kelly’s Network Utility Maximization (NUM) is a popular framework for decentralizing transport protocols in data networks that optimize a fairness objective [36]. NUM uses link “prices” derived from the solution to the utility maximization problem, and senders compute rates based on these router prices. Congestion control algorithms that use router signals based on buffer sizes or queueing delays [29, 50, 21] are closely related. While the Internet congestion control literature has focused on links with fairly stable capacities, this paper shows that they can be effective even in networks with capacities dependent on the input rates themselves. Such problems have also been explored in the context of ridesharing, for instance [23, 24], and require new innovation in both formulating and solving routing problems.

9 Discussion

We motivate the need for efficient routing on PCNs and propose Spider, a protocol for balanced, high-throughput routing in PCNs. Spider uses a packet-switched architecture, multi-path congestion control, and in-network scheduling. Spider achieves nearly 100% throughput on circulation payment demands across both synthetic and real topologies. We show how the presence of DAG payments causes deadlocks that hurt the circulation throughput in addition to DAG throughput. These can be resolved only via on-chain rebalancing. In such cases, Spider requires rebalancing only every 10K transactions to recover 100% throughput while state-of-the-art schemes only achieve 85-92% even when rebalancing every 10 transactions.

Future work includes further investigating the effect of DAG payments on circulation throughput. Not all DAGs cause deadlocks that hurt circulation throughput. Further, this effect isn’t uniform across different topologies. Rebalancing schemes that explicitly account for demand imbalance at routers could be more effective. Such rebalancing schemes should be consensus-protocol-aware: the longer it takes for transactions to be confirmed, the more proactive rebalancing needs to be.

10 Acknowledgements

We thank Andrew Miller, Thaddeus Dryja and Vikram Nathan for their detailed feedback. We also thank the Distributed Technologies Research Foundation, Input-Output Hong Kong Inc, the National Science Foundation under grants CNS-1718270 and CNS-1617702, and the Army Research Office under grant W911NF1810332 for their support.

References

- [1] <http://omnetpp.org/>.
- [2] Amount-independent payment routing in Lightning Networks. <https://medium.com/coinmonks/amount-independent-payment-routing-in-lightning-networks-6409201ff5ed>.
- [3] AMP: Atomic Multi-Path Payments over Lightning. <https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html>.
- [4] Bitcoin Core. <https://bitcoin.org/en/bitcoin-core/>.
- [5] Bitcoin Core Daemon. <https://bitcoin.org/en/full-node#other-linux-daemon>.
- [6] Bitcoin historical fee chart. <https://bitinfocharts.com/comparison/bitcoin-median-transaction-fee.html>.
- [7] Bitcoin Regtest Mode. <https://bitcoin.org/en/developer-examples#regtest-mode>.
- [8] Blockchain caffe. <https://blockchaincaffe.org/map/>.
- [9] c-lightning: A specification compliant Lightning Network implementation in C. <https://github.com/ElementsProject/lightning>.
- [10] Credit Card Merchant Processing Fees. <https://paymentdepot.com/blog/average-credit-card-processing-fees/>.
- [11] etcd: A distributed, reliable key-value store for the most critical data of a distributed system. <https://github.com/etcd-io/etcd>.
- [12] Ethereum. <https://www.ethereum.org/>.
- [13] Lightning Network Daemon. <https://github.com/lightningnetwork/lnd>.
- [14] Lightning Network Search and Analysis Engine. <https://lml.com>.
- [15] Onion Routed Micropayments for the Lightning Network. <https://github.com/lightningnetwork/lightning-onion>.
- [16] Raiden network. <https://raiden.network/>.
- [17] The NewReno Modification to TCP's Fast Recovery Algorithm. <https://tools.ietf.org/html/rfc6582>.
- [18] Watts Strogatz Graph. https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.generators.directed.scale_free_graph.html.
- [19] Watts Strogatz Graph. https://networkx.github.io/documentation/networkx-1.9/reference/generated/networkx.generators.random_graphs.watts_strogatz_graph.html.

- [20] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, 1993.
- [21] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). *ACM SIGCOMM computer communication review*, 41(4):63–74, 2011.
- [22] Anonymous. Anonymized for double-blind reviewing.
- [23] S. Banerjee, R. Johari, and C. Riquelme. Pricing in ride-sharing platforms: A queueing-theoretic approach. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, pages 639–639. ACM, 2015.
- [24] S. Banerjee, R. Johari, and C. Riquelme. Dynamic pricing in ridesharing platforms. *ACM SIGecom Exchanges*, 15(1):65–70, 2016.
- [25] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- [26] L. S. Brakmo, S. W. O’Malley, and L. L. Peterson. *TCP Vegas: New techniques for congestion detection and avoidance*, volume 24. ACM, 1994.
- [27] C. Burchert, C. Decker, and R. Wattenhofer. Scalable funding of bitcoin micropayment channel networks. *Royal Society open science*, 5(8):180089, 2018.
- [28] C. N. Cordi. *Simulating high-throughput cryptocurrency payment channel networks*. PhD thesis, 2017.
- [29] N. Dukkipati. *Rate Control Protocol (RCP): Congestion control to make flows complete quickly*. Citeseer, 2008.
- [30] A. Eryilmaz and R. Srikant. Joint congestion control, routing, and mac for stability and fairness in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24(8):1514–1524, 2006.
- [31] D. Goldschlag, M. Reed, and P. Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.
- [32] U. M. L. Group. Credit card fraud detection, 2018. <https://www.kaggle.com/mlg-ulb/creditcardfraud>.
- [33] S. Ha, I. Rhee, and L. Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.
- [34] P. Hu and W. C. Lau. A survey and taxonomy of graph sampling. *arXiv preprint arXiv:1308.5865*, 2013.
- [35] V. Jacobson and M. J. Karels. Congestion avoidance and control. In *SIGCOMM 1988*, Stanford, CA, Aug. 1988.
- [36] F. Kelly and T. Voice. Stability of end-to-end algorithms for joint routing and rate control. *ACM SIGCOMM Computer Communication Review*, 35(2):5–12, 2005.
- [37] F. P. Kelly, A. K. Maulloo, and D. K. Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, 49(3):237–252, 1998.
- [38] R. Khalil and A. Gervais. Revive: Rebalancing off-blockchain payment networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 439–453. ACM, 2017.

- [39] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff. A brief history of the internet. *SIGCOMM Comput. Commun. Rev.*, 39(5):22–31, Oct. 2009.
- [40] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei. SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks. *IACR Cryptology ePrint Archive*, 2016:1054, 2016.
- [41] R. McManus. Blockchain speeds & the scalability debate. *Blockspain*, February 2018.
- [42] P. Moreno-Sanchez, A. Kate, M. Maffei, and K. Pecina. Privacy preserving payments in credit networks. In *Network and Distributed Security Symposium*, 2015.
- [43] D. P. Palomar and M. Chiang. A tutorial on decomposition methods for network utility maximization. *IEEE Journal on Selected Areas in Communications*, 24(8):1439–1451, 2006.
- [44] J. Poon and T. Dryja. The Bitcoin Lightning Network: Scalable Off-chain Instant Payments. *draft version 0.5*, 9:14, 2016.
- [45] P. Prihodko, S. Zhigulin, M. Sahno, A. Ostrovskiy, and O. Osuntokun. Flare: An approach to routing in lightning network. 2016.
- [46] H. Racke. Minimizing congestion in general networks. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 43–52. IEEE, 2002.
- [47] S. Roos, M. Beck, and T. Strufe. Anonymous addresses for efficient and resilient routing in f2f overlays. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.
- [48] 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.
- [49] R. Srikant. *The mathematics of Internet congestion control*. Springer Science & Business Media, 2012.
- [50] C.-H. Tai, J. Zhu, and N. Dukkupati. Making large scale deployment of rcp practical for real networks. In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, pages 2180–2188. IEEE, 2008.
- [51] S. Thomas and E. Schwartz. A protocol for interledger payments. URL <https://interledger.org/interledger.pdf>, 2015.
- [52] P. F. Tsuchiya. The landmark hierarchy: a new hierarchy for routing in very large networks. In *ACM SIGCOMM Computer Communication Review*, volume 18, pages 35–42. ACM, 1988.
- [53] Visa. Visa acceptance for retailers. <https://usa.visa.com/run-your-business/small-business-tools/retail.html>.
- [54] P. Wang, H. Xu, X. Jin, and T. Wang. Flash: efficient dynamic routing for offchain networks. *arXiv preprint arXiv:1902.05260*, 2019.
- [55] D. Wischik, M. Handley, and M. B. Braun. The resource pooling principle. *ACM SIGCOMM Computer Communication Review*, 38(5):47–52, 2008.

- [56] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In *NSDI*, volume 11, pages 8–8, 2011.
- [57] J. Y. Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.

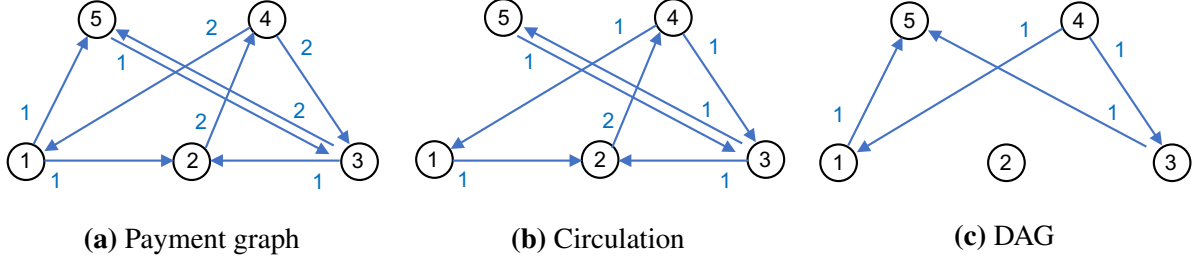


Figure 17: Example payment graph (denoted by blue lines) for a five node network (left). It decomposes into a maximum circulation and DAG components as shown in (b) and (c).

Appendices

A Circulations and Throughput Bounds

For a network $G(V, E)$ with set of routers V , we define a *payment graph* $H(V, E_H)$ as a graph that specifies the payment demands between different users. The weight of any edge (i, j) in the payment graph is the average rate at which user i seeks to transfer funds to user j . A *circulation graph* $C(V, E_C)$ of a payment graph is any subgraph of the payment graph in which the weight of an edge (i, j) is at most the weight of (i, j) in the payment graph, and moreover the total weight of incoming edges is equal to the total weight of outgoing edges for each node. Of particular interest are *maximum circulation graphs* which are circulation graphs that have the highest total demand (i.e., sum of edge weights), among all possible circulation graphs. A maximum circulation graph is not necessarily unique for a given payment graph.

Proposition 1. *Consider a payment graph H with a maximum circulation graph C^* . Let $\nu(C^*)$ denote the total demand in C^* . Then, on a network in which each payment channel has at least $\nu(C^*)$ units of escrowed funds, there exists a balanced routing scheme that can achieve a total throughput of $\nu(C^*)$. However, no balanced routing scheme can achieve a throughput greater than $\nu(C^*)$ on any network.*

Proof. Let $w_{C^*}(i, j)$ denote the payment demand from any user i to user j in the maximum circulation graph C^* . To see that a throughput of $\nu(C^*)$ is achievable, consider routing the circulation demand along the shortest paths of any spanning tree T of the payment network G . In this routing, for any pair of nodes $i, j \in V$ there exists a unique path from i to j in T through which $w_{C^*}(i, j)$ amount of flow is routed. We claim that such a routing scheme is perfectly balanced on all the links. This is because for any partition $S, V \setminus S$ of C^* , the net flow going from S to $V \setminus S$ is equal to the net flow going from $V \setminus S$ to S in C^* . Since the flows along an edge e of T correspond precisely to the net flows across the partitions obtained by removing e in T , it follows that the flows on e are balanced as well. Also, for any flow (i, j) in the demand graph C^* , the shortest path route from i to j in T can cross an edge e at most once. Therefore the total amount of flow going through an edge is at most the total amount of flow in C^* , which is $\nu(C^*)$.

Next, to see that no balanced routing scheme can achieve a throughput greater than $\nu(C^*)$, assume the contrary and suppose there exists a balanced routing scheme SCH with a throughput

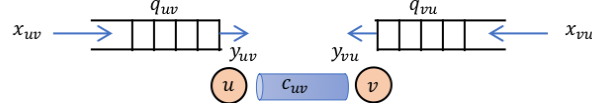


Figure 18: Model of queues at a payment channel between nodes u and v . x_{uv} and y_{uv} denote the rates at which transaction-units for v arrive into and get serviced at the queue at u respectively. c_{uv} is the capacity of the payment channel and q_{uv} denotes the total number of transaction-units waiting in u 's queue to be serviced.

greater than $\nu(C^*)$. Let $H_{\text{SCH}} \subseteq H$ be a payment graph where the edges represent the portion of demand that is actually routed in SCH. Since $\nu(H_{\text{SCH}}) > \nu(C^*)$, H_{SCH} is not a circulation and there exists a partition $S, V \setminus S$ such that the net flow from S to $V \setminus S$ is strictly greater than the net flow from $V \setminus S$ to S in H_{SCH} . However, the net flows routed by SCH across the same partition $S, V \setminus S$ in G are balanced (by assumption) resulting in a contradiction. Thus we conclude there does not exist any balanced routing scheme that can achieve a throughput greater than $\nu(C^*)$. \square

B Optimality of Spider

B.1 Fluid Model

In this section we describe a fluid model approximation of the system dynamics under Spider's protocol. Following a similar notation as in §5, for a path p we let $x_p(t)$ denote the rate of flow on it at time t . For a channel (u, v) and time t , let $q_{u,v}(t)$ be the size of the queue at router u , $f_{u,v}(t)$ be the fraction of incoming packets that are marked at u , $x_{u,v}(t)$ be the total rate of incoming flow at u , and $y_{u,v}(t)$ be the rate at which transactions are serviced (*i.e.*, forwarded to router v) at u . All variables are real-valued. We approximate Spider's dynamics via the following system of equations

$$\dot{x}_p(t) = \left[\frac{x_p(t)}{\sum_{p' \in \mathcal{P}_{i_p, j_p}} x_{p'}(t)} - \sum_{(u,v) \in p} f_{u,v}(t) x_p(t) \right]_{x_p(t)}^+ \quad \forall p \in \mathcal{P} \quad (12)$$

$$\dot{q}_{u,v}(t) = [x_{u,v}(t) - y_{u,v}(t)]_{q_{u,v}(t)}^+ \quad \forall (u, v) \in E \quad (13)$$

$$\dot{f}_{u,v}(t) = [q_{u,v}(t) - q_{\text{thresh}}]_{f_{u,v}(t)}^+ \quad \forall (u, v) \in E, \quad (14)$$

where $y_{u,v}(t) = y_{v,u}(t) =$

$$\begin{cases} \frac{c_{u,v}}{2\Delta} & \text{if } q_{u,v}(t) > 0 \ \& \ q_{v,u}(t) > 0 \\ \min\{\frac{c_{u,v}}{2\Delta}, x_{v,u}(t)\} & \text{if } q_{u,v}(t) > 0 \ \& \ q_{v,u}(t) = 0 \\ \min\{\frac{c_{u,v}}{2\Delta}, x_{u,v}(t)\} & \text{if } q_{u,v}(t) = 0 \ \& \ q_{v,u}(t) > 0 \\ \min\{\frac{c_{u,v}}{2\Delta}, x_{u,v}(t), x_{v,u}(t)\} & \text{if } q_{u,v}(t) = 0 \ \& \ q_{v,u}(t) = 0 \end{cases} \quad (15)$$

for each $(u, v) \in E$. Let i_p and j_p denote the source and destination nodes for path p respectively. Then, \mathcal{P}_{i_p, j_p} denotes the set of all paths i_p uses to route to j_p . Equation (12) models how the

rate on a path p increases upon receiving successful acknowledgements or decreases if the packets are marked, per Equations (10) and (11) in §6.3. If the fraction of packets marked at each router is small, then the aggregate fraction of packets that return marked on a path p can be approximated by the sum $\sum_{(u,v) \in p} f_{u,v}$ [49]. Hence the rate which marked packets arrive for a path p is $\sum_{(u,v) \in p} f_{u,v} x_p$. Similarly, the rate which successful acknowledgements are received on a path p is $x_p(1 - \sum_{(u,v) \in p} f_{u,v})$, which can be approximated as simply x_p if the marking fractions are small. Since Spider increases the window by $1/(\sum_{p' \in \mathcal{P}_{i_p, j_p}} w_{p'})$ for each successful acknowledgement received, the average rate at which x_p increases is $x_p/(\sum_{p' \in \mathcal{P}_{i_p, j_p}} x_{p'})$. Lastly, the rate x_p cannot become negative; so if $x_p = 0$ we disallow \dot{x}_p from being negative. The notation $(x)_y^+$ means x if $y > 0$ and 0 if $y = 0$.

Equations (13) and (14) model how the queue sizes and fraction of packets marked, respectively, evolve at the routers. For a router u in payment channel (u, v) , by definition $y_{u,v}$ is the rate at which transactions are serviced from the queue $q_{u,v}$, while transactions arrive at the queue at a rate of $x_{u,v}$ (Figure 18). Hence the net rate at which $q_{u,v}$ grows is given by the difference $x_{u,v} - y_{u,v}$. The fraction of packets marked at a queue grows if the queue size is larger than a threshold q_{thresh} , and drops otherwise, as in Equation (14). This approximates the marking model of Spider (§6.2) in which packets are marked at a router if their queuing delay exceeds a threshold.

To understand how the service rate $y_{u,v}$ evolves (Equation (15)), we first make the approximation that the rate at which transactions are serviced from the queue at a router u is equal to the rate at which tokens are replenished at the router, *i.e.*, $y_{u,v} = y_{v,u}$ for all $(u, v) \in E$. The precise value for $y_{u,v}$ at any time, depends on both the arrival rates and current occupancy of the queues at routers u and v . If both $q_{u,v}$ and $q_{v,u}$ are non-empty, then there are no surplus of tokens available within the channel. A token when forwarded by a router is unavailable for Δ time units, until its acknowledgement is received. Therefore the maximum rate at which tokens on the channel can be forwarded is $c_{u,v}/\Delta$, implying $y_{u,v} + y_{v,u} = c_{u,v}$ or $y_{u,v} = y_{v,u} = c_{u,v}/(2\Delta)$ in this case. If $q_{u,v}$ is non-empty and $q_{v,u}$ is empty, then there are no surplus tokens available at u 's end. Router v however may have tokens available, and service transactions at the same rate at which they are arriving, *i.e.*, $y_{v,u} = x_{v,u}$. This implies tokens become available at router u at a rate of $x_{v,u}$ and hence $y_{u,v} = x_{v,u}$. However, if the transaction arrival rate $x_{v,u}$ is too large at v , it cannot service them at a rate more than $c_{v,u}/(2\Delta)$ and a queue would start building up at $q_{v,u}$. The case where $q_{u,v}$ is empty and $q_{v,u}$ is non-empty follows by interchanging the variables u and v in the description above. Lastly, if both $q_{u,v}$ and $q_{v,u}$ are empty, then the service rate $y_{u,v}$ can at most be equal to the arrival rate $x_{v,u}$. Similarly $y_{v,u}$ can be at most $x_{u,v}$. Since $y_{u,v} = y_{v,u}$ by our approximation, we get the expression in Equation (15).

We have not explicitly modeled delays, and have made simplifying approximations in the fluid model above. Nevertheless this model is useful for gaining intuition about the first-order behavior of the Spider protocol. In the following section, we use this model to show that Spider finds optimal rate allocations for a parallel network topology.

B.2 Proof of Optimality

Consider a PCN comprising of two sets of end-hosts $\{e_1, \dots, e_m\}$ and $\{e'_1, \dots, e'_n\}$ that are connected via k parallel payment channels $(r_1, r'_1), \dots, (r_k, r'_k)$ as shown in Figure 19. The end-hosts from each set have demands to end-hosts on the other set. The end-hosts within a set, however, do

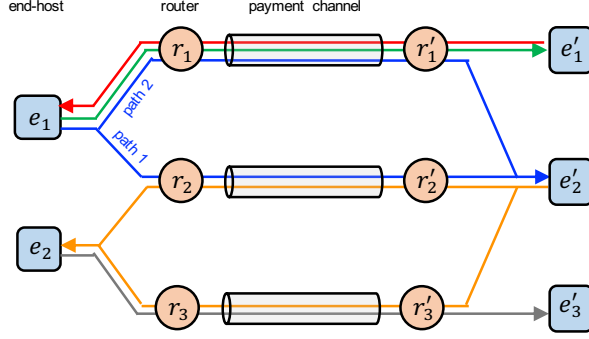


Figure 19: Example of a parallel network topology with bidirectional flows on each payment channel.

not have any demands between them. Let the paths for different source-destination pairs be such that for each path p , if p contains a directed edge (r_i, r'_i) for some i then there exists another path (for a different source-destination pair) that contains the edge (r'_i, r_i) . We will show that running Spider on this network results in rate allocations that are an optimal solution to the optimization problem in Equations (1)–(5). Under a fluid model for Spider as discussed in §B.1, assuming convergence, we observe that in the steady-state the time derivatives of the rate of flow of each path (Equation (12)) must be non-positive, *i.e.*,

$$\frac{1}{\sum_{p' \in \mathcal{P}_{i_p, j_p}} x_{p'}^*} - \sum_{(u,v) \in p} f_{u,v}^* \begin{cases} = 0 & \text{if } x_p^* > 0 \\ \leq 0 & \text{if } x_p^* = 0 \end{cases} \quad \forall p \in \mathcal{P}, \quad (16)$$

where the superscript $*$ denotes values at convergence (*e.g.*, x_p^* is the rate of flow on path p at convergence). Similarly, the rate of growth of the queues must be non-positive, or

$$x_{u,v}^* \begin{cases} = y_{u,v}^* & \text{if } q_{u,v}^* > 0 \\ \leq y_{u,v}^* & \text{if } q_{u,v}^* = 0 \end{cases} \quad \forall (u,v) \in E. \quad (17)$$

Now, consider the optimization problem in Equations (1)–(5) for this parallel network. For simplicity we will assume the sender-receiver demands are not constrained. From Equation (17) above, the transaction arrival rates $x_{u,v}^*$ and $x_{v,u}^*$ for a channel (u,v) satisfy the capacity constraints in Equation (3). This is because $x_{u,v}^* \leq y_{u,v}^*$ from Equation (17) and $y_{u,v}(t)$ is at most $\frac{c_{u,v}}{2\Delta}$ from Equation (15). Similarly the transaction arrival rates also satisfy the balance constraints in Equation (4). To see this, we first note that the queues on all payment channels through which a path (corresponding to a sender-receiver pair) passes must be non-empty. For otherwise, if a queue $q_{u,v}^*$ is empty then the fraction of marked packets on a path p through (u,v) goes to 0, and the rate of flow x_p^* would increase as per Equation (12). Therefore we have $x_{u,v}^* = y_{u,v}^*$ (from Equation (17)) for every channel. Combining this with $y_{u,v}(t) = y_{v,u}(t)$ (Equation (15)), we conclude that the arrival rates are balanced on all channels. Thus the equilibrium rates $\{x_p^* : p \in \mathcal{P}\}$ resulting from Spider are in the feasible set for the routing optimization problem.

Next, let $\lambda_{u,v} \geq 0$ and $\mu_{u,v} \in \mathbb{R}$ be the dual variables corresponding to the capacity and balance constraints, respectively, for a channel (u,v) . Consider the following mapping from $f_{u,v}^*$ to $\lambda_{u,v}$

and $\mu_{u,v}$

$$\lambda_{u,v}^* \leftarrow (f_{u,v}^* + f_{v,u}^*)/2 \quad \forall (u,v) \in E \quad (18)$$

$$\mu_{u,v}^* \leftarrow f_{u,v}^*/2 \quad \forall (u,v) \in E, \quad (19)$$

where the superscript $*$ on the dual variables indicate that they have been derived from the equilibrium states of the Spider protocol. Since $f_{u,v}(t)$ is always non-negative (Equation (14)), we see that $\lambda_{u,v}^* \geq 0$ for all (u,v) . Therefore $\{\lambda_{u,v}^* : (u,v) \in E\}$ and $\{\mu_{u,v}^* : (u,v) \in E\}$ are in the feasible set of the dual of the routing optimization problem.

Next, we have argued previously that the queues on all payment channels through which a path (corresponding to a sender-receiver pair) passes must be non-empty. While we used this observation to show that the channel rates $x_{u,v}^*$ are balanced, it also implies that the rates are at capacity, *i.e.*, $x_{u,v}^* = c_{u,v}/(2\Delta)$, or $x_{u,v}^* + x_{v,u}^* = c_{u,v}/\Delta$ for all (u,v) . This directly follows from Equation (17) and the first sub-case in Equation (15). It follows that the primal variables $\{x_p^* : p \in \mathcal{P}\}$ and the dual variables $\{\lambda_{u,v}^* : (u,v) \in E\}$, $\{\mu_{u,v}^* : (u,v) \in E\}$ satisfy the complementary slackness conditions of the optimization problem.

Last, the optimality condition for the primal variables on the Lagrangian defined with dual variables $\{\lambda_{u,v}^* : (u,v) \in E\}$ and $\{\mu_{u,v}^* : (u,v) \in E\}$ stipulates that

$$\frac{1}{\sum_{p' \in \mathcal{P}_{i_p, j_p}} x_{p'}} - \sum_{(u,v) \in p} (\lambda_{u,v}^* + \mu_{u,v}^* - \mu_{v,u}^*) \begin{cases} = 0 & \text{if } x_p > 0 \\ \leq 0 & \text{if } x_p = 0 \end{cases}, \quad (20)$$

for all $p \in \mathcal{P}$. However, note that for any path p

$$\begin{aligned} \sum_{(u,v) \in p} (\lambda_{u,v}^* + \mu_{u,v}^* - \mu_{v,u}^*) &= \sum_{(u,v) \in p} \frac{f_{u,v}^* + f_{v,u}^*}{2} + \frac{f_{u,v}^*}{2} - \frac{f_{v,u}^*}{2} \\ &= \sum_{(u,v) \in p} f_{u,v}^*, \end{aligned} \quad (21)$$

where the first equation above follows from our mapping for $\lambda_{u,v}^*$ and $\mu_{u,v}^*$ in Equations (18), (19). Combining this with Equation (16), we see that $x_p \leftarrow x_p^*$ for all $p \in \mathcal{P}$ is a valid solution to the Equation (20). Hence we conclude that $\{x_p^* : p \in \mathcal{P}\}$ and $\{\lambda_{u,v}^* : (u,v) \in E\}$, $\{\mu_{u,v}^* : (u,v) \in E\}$ are optimal primal and dual variables, respectively, for the optimization problem. The equilibrium rates found by Spider for the parallel network topology are optimal.

C Modeling on-chain rebalancing

We model on-chain payment channel rebalancing using variables $b_{(u,v)}$, which is the average rate at which payment channel (u,v) receives new funds in the $u \rightarrow v$ direction via on-chain transactions. These funds would typically be withdrawn from another payment channel on which node u is receiving more than it is sending. A positive rebalancing rate on some payment channels ($b_{(u,v)} > 0$) can improve network throughput beyond the upper bound presented in Proposition 1. However, rebalancing channels on chain is expensive for the routers both in time (due to transaction confirmation delays) and in transaction fees (paid to the miners). Thus routers may be unwilling

to actively rebalance their payment channels unless the profit obtained from the higher utility is sufficient to offset the rebalancing costs.

We leave a full analysis of incentives for routers to future work, but we now consider a simple modification to the previous optimization problem to understand the impact of on-chain rebalancing:

$$\text{maximize} \quad \sum_{i,j \in V} U\left(\sum_{p \in \mathcal{P}_{i,j}} x_p\right) - \gamma \sum_{(u,v) \in E} b_{(u,v)} \quad (22)$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{P}_{i,j}} x_p \leq d_{i,j} \quad \forall i, j \in V \quad (23)$$

$$\sum_{p \in \mathcal{P}: (u,v) \in p} x_p + \sum_{p \in \mathcal{P}: (v,u) \in p} x_p \leq \frac{c_{(u,v)}}{\Delta} \quad \forall (u,v) \in E \quad (24)$$

$$\sum_{p \in \mathcal{P}: (u,v) \in p} x_p - \sum_{p \in \mathcal{P}: (v,u) \in p} x_p \leq b_{(u,v)} \quad \forall (u,v) \in E \quad (25)$$

$$x_p \geq 0 \quad \forall p \in \mathcal{P} \quad (26)$$

$$b_{(u,v)} \geq 0 \quad \forall (u,v) \in E. \quad (27)$$

The objective in the problem above consists of two terms. The first term is the total utility, which we would like to maximize. The second term is the “cost” of on-chain rebalancing at a total rate of $\sum_{(u,v) \in E} b_{(u,v)}$, which we would like to minimize. The parameter γ dictates how we weight these two conflicting goals. We can interpret γ as the increase in utility required to offset one unit of on-chain rebalancing rate. In practice, γ would depend on a variety of factors, such as the ratio of blockchain transaction fees to routing fees and how the transaction fees are computed. For example, if blockchain transaction fees do not depend on transaction size, routers can reduce the cost of rebalancing by using larger transactions, thereby effectively reducing γ in the above model.

We now analyze how on-chain rebalancing impacts utility. Let $B \triangleq \sum_{(u,v) \in E} b_{(u,v)}$ be the total rate of on-chain rebalancing, and let’s assume for simplicity that the payment channel capacities are unlimited. For large γ , B will be close to 0 in the optimal solution. From Proposition 1, we know that the maximum achievable throughput in this case is $\nu(C^*)$. On the other hand, for $\gamma \approx 0$, B can become arbitrarily large. It is not difficult to see that the network throughput can satisfy all of the demand $\sum_{i,j} d_{i,j}$ in this case, provided that the links have sufficient capacity (i.e, if the constraints (24) are not tight). In general, as γ decreases, the total utility and total rebalancing rate both increase in the optimal solution, until the utility reaches the maximum possible utility given the capacity constraints.

We can show that the maximum achievable utility is a non-decreasing concave function of the total rebalancing rate. The argument is as follows. Let $t(B)$ be the maximum achievable utility for a bound B on the total rebalancing rate. $t(B)$ is the solution of the following optimization

problem:

$$\text{maximize} \quad \sum_{i,j \in V} U\left(\sum_{p \in \mathcal{P}_{i,j}} x_p\right) \quad (28)$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{P}_{i,j}} x_p \leq d_{i,j} \quad \forall i, j \in V \quad (29)$$

$$\sum_{p \in \mathcal{P}: (u,v) \in p} x_p + \sum_{p \in \mathcal{P}: (v,u) \in p} x_p \leq \frac{c_{(u,v)}}{\Delta} \quad \forall (u,v) \in E \quad (30)$$

$$\sum_{p \in \mathcal{P}: (u,v) \in p} x_p - \sum_{p \in \mathcal{P}: (v,u) \in p} x_p \leq b_{(u,v)} \quad \forall (u,v) \in E \quad (31)$$

$$\sum_{(u,v) \in E} b_{(u,v)} \leq B \quad (32)$$

$$x_p \geq 0 \quad \forall p \in \mathcal{P} \quad (33)$$

$$b_{(u,v)} \geq 0 \quad \forall (u,v) \in E. \quad (34)$$

$t(\cdot)$ is non-decreasing because the set of feasible solutions is non-decreasing in B . To see that $t(\cdot)$ is concave, consider arbitrary B_1 and B_2 and $\theta \in (0, 1)$. Let $\{(x_p^1, b_{(u,v)}^1) : p \in \mathcal{P}, (u,v) \in E\}$ and $\{(x_p^2, b_{(u,v)}^2) : p \in \mathcal{P}, (u,v) \in E\}$ be the optimal solutions corresponding to B_1 and B_2 . Then $\{(\theta x_p^1 + (1-\theta)x_p^2, \theta b_{(u,v)}^1 + (1-\theta)b_{(u,v)}^2) : p \in \mathcal{P}, (u,v) \in E\}$ is a feasible solution for bound $\theta B_1 + (1-\theta)B_2$, and it achieves a utility of $\theta t(B_1) + (1-\theta)t(B_2)$. Hence, $t(\theta B_1 + (1-\theta)B_2) \geq \theta t(B_1) + (1-\theta)t(B_2)$, and $t(\cdot)$ is concave.

D Primal-Dual Algorithm Derivation

In this section, we present a formal derivation of the decentralized algorithm for computing the optimum solution of the fluid-model LP (Eq. (22)–(27)). Consider the partial lagrangian:

$$\begin{aligned} L(\mathbf{x}, \mathbf{b}, \lambda, \mu) = & \sum_{i,j \in V} U\left(\sum_{p \in \mathcal{P}_{i,j}} x_p\right) - \gamma \sum_{(u,v) \in E} b_{(u,v)} \\ & - \sum_{(u,v) \in E} \lambda_{(u,v)} \left[\sum_{p \in \mathcal{P}: (u,v) \in p} x_p + \sum_{p \in \mathcal{P}: (v,u) \in p} x_p - \frac{c_{(u,v)}}{\Delta} \right] \\ & - \sum_{(u,v) \in E} \mu_{(u,v)} \left[\sum_{p \in \mathcal{P}: (u,v) \in p} x_p - \sum_{p \in \mathcal{P}: (v,u) \in p} x_p - b_{(u,v)} \right], \quad (35) \end{aligned}$$

where $\lambda_{(u,v)}$ and $\mu_{(u,v)}$ are lagrange variables corresponding to the capacity and rate-imbalance constraints in Eq. (24) and Eq. (25) respectively. The partial lagrangian can be rewritten as:

$$L(\mathbf{x}, \mathbf{b}, \lambda, \mu) = \sum_{i,j \in V} \left[U \left(\sum_{p \in \mathcal{P}_{i,j}} x_p \right) - \sum_{p \in \mathcal{P}_{i,j}} x_p \sum_{(u,v) \in p} (\lambda_{(u,v)} + \lambda_{(v,u)} + \mu_{(u,v)} - \mu_{(v,u)}) \right] + \sum_{(u,v) \in E} (\mu_{(u,v)} - \gamma) b_{(u,v)} + \sum_{(u,v) \in E} \lambda_{(u,v)} \frac{c_{(u,v)}}{\Delta}. \quad (36)$$

Define $z_{(u,v)} \triangleq \lambda_{(u,v)} + \lambda_{(v,u)} + \mu_{(u,v)} - \mu_{(v,u)}$ to be the price for edge $(u,v) \in E$, and $z_p \triangleq \sum_{(u,v) \in p} z_{(u,v)}$ to be price for path $p \in \mathcal{P}$. The important fact about the above partial lagrangian is that it decomposes into separate terms for the rate variables of each source/destination pair $\{x_p : p \in \mathcal{P}_{i,j}\}$, and the rebalancing rate of each edge $b_{(u,v)}$. This suggests the following primal-dual algorithm for solving the optimization problem in Eqs. (22)–(27):

- **Primal step.** Given the edge prices $z_{(u,v)}(t)$ at time t , for each source/destination pair (i,j) , update the sending rate on paths $p \in \mathcal{P}_{i,j}$ as follows:

$$\begin{aligned} x_p(t) &= x_p(t-1) + \alpha (U'(x) - z_p(t)), \\ x_p(t) &= \text{Proj}_{\mathcal{X}_{i,j}}(x_p(t)), \end{aligned} \quad (37)$$

where $\text{Proj}_{\mathcal{X}_{i,j}}(\cdot)$ stands for projection onto the convex set $\mathcal{X}_{i,j} = \{x_p \text{ for } p \in \mathcal{P}_{i,j} | x_p \geq 0, \sum_{p \in \mathcal{P}_{i,j}} x_p \leq d_{i,j}\}$.

At time t , each edge (u,v) also updates its on-chain rebalancing rate according to:

$$b_{(u,v)}(t) = [b_{(u,v)}(t-1) + \beta (\mu_{(u,v)}(t) - \gamma)]_+, \quad (38)$$

- **Dual step.** Similarly, for the dual step let $x_p(t)$ denote the flow rate along path p at time t and

$$w_{(u,v)}(t) = \sum_{p \in \mathcal{P}: (u,v) \in p} x_p(t) + \sum_{p \in \mathcal{P}: (v,u) \in p} x_p(t) - \frac{c_{(u,v)}}{\Delta} \quad (39)$$

$$y_{(u,v)}(t) = \sum_{p \in \mathcal{P}: (u,v) \in p} x_p(t) - \sum_{p \in \mathcal{P}: (v,u) \in p} x_p(t) - b_{(u,v)} \quad (40)$$

be the slack in the capacity and balance constraints respectively for a payment channel (u,v) . Then, each channel $(u,v) \in E$ updates its prices as

$$\lambda_{(u,v)}(t+1) = [\lambda_{(u,v)}(t) + \eta w_{(u,v)}(t)]_+ \quad (41)$$

$$\mu_{(u,v)}(t+1) = [\mu_{(u,v)}(t) + \kappa y_{(u,v)}(t)]_+ \quad (42)$$

$$\mu_{(v,u)}(t+1) = [\mu_{(v,u)}(t) - \kappa y_{(u,v)}(t)]_+. \quad (43)$$

The parameters α, η, κ are positive "step size" constants, that determine the rate at which the algorithm converges. Using standard arguments we can show that for small enough step sizes, the algorithm would converge to the optimal solution of the LP in Eq. (22)–(27).

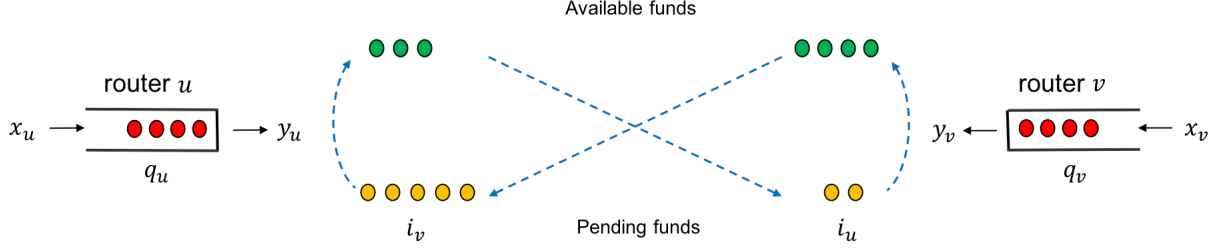


Figure 20: Routers queue transaction units and schedule them across the payment channel based on available capacity and transaction priorities. Funds received on a payment channel remain in a pending state until the final receiver provides the key for the hashlock.

The algorithm has the following intuitive interpretation. $\lambda_{(u,v)}$ and $\mu_{(u,v)}, \mu_{(v,u)}$ are prices that vary due to capacity constraints and imbalance at the payment channels. In Eq. (41), $\lambda_{(u,v)}$ would increase if the total rate on channel (u, v) (in both directions) exceeds its capacity, and would decrease to 0 if there is excess capacity. Similarly, $\mu_{(u,v)}$ increases if the rate in the (u, v) direction exceeds the rate in the (v, u) direction by more than $b_{(u,v)}$, the rate at which funds are deposited for edge (u, v) on chain. As the prices vary, an end-host with a flow on path p would react according to Eq. (37) by increasing its sending rate x_p if the total price of the path p is cheap, and decreasing the rate otherwise. Simultaneously, each edge also adapts its on-chain rebalancing rate based on Eq. (38). If $\mu_{(u,v)} > \gamma$, the price charged for imbalance is higher than the cost of on-chain rebalancing per unit rate. Hence the edge increases its on-chain rebalancing rate $b_{(u,v)}$; otherwise it decreases its utility. Finally, we note that in the absence of any on-chain rebalancing, the algorithm can be simplified by setting $b_{(u,v)} = 0$ for all edges (u, v) .

The net effect is the convergence of the rate and price variables to values such that the overall utility of the network is maximized. We remark that the objective of our optimization problem in Eq. (22) can be modified to also ensure fairness in routing (as described in §5, by associating an appropriate utility function with each sender-receiver pair [37]). A decentralized algorithm for such a case may be derived analogously as our proposed solution.

E Estimating the Demand-Capacity Gap at the Routers

In this section, we explain how Spider estimates the total amount of demand on a channel at any time, for updating the capacity price λ in Eq. (7). From the description of the primal-dual algorithm for the fluid model in Appendix D, we see that updating $\lambda_{(u,v)}$ at a channel $(u, v) \in E$ requires estimating

$$\sum_{\substack{p \in \mathcal{P}: \\ (u,v) \in p}} x_p(t) + \sum_{\substack{p' \in \mathcal{P}: \\ (v,u) \in p'}} x_{p'}(t) - \frac{c_{(u,v)}}{\Delta} \quad (44)$$

at the channel (Eq. (41)). While the total rate at which transactions are arriving at u ($\sum_{p \in \mathcal{P}: (u,v) \in p} x_p(t)$) and at v ($\sum_{p' \in \mathcal{P}: (v,u) \in p'} x_{p'}(t)$) are straightforward to estimate, estimating Δ —the average time taken for transactions to reach their destination from the channel, and for their hashlock keys to

arrive at the channel—is difficult. In Spider, we overcome this problem by estimating the quantity

$$\sum_{\substack{p \in \mathcal{P}: \\ (u,v) \in p}} x_p(t)\Delta + \sum_{\substack{p' \in \mathcal{P}: \\ (v,u) \in p'}} x_{p'}(t)\Delta - c_{(u,v)}, \quad (45)$$

instead of trying to estimate the expression in Eq. (44). Eq. (45) is simply a scaling of Eq. (44), but can be estimated without having to first estimate Δ . To see this, let $\tilde{x}_u(t) = \sum_{p \in \mathcal{P}: (u,v) \in p} x_p(t)$ and $\tilde{x}_v(t) = \sum_{p' \in \mathcal{P}: (v,u) \in p'} x_{p'}(t)$ denote the rate of transaction arrival at u and v respectively. Similarly, let $\tilde{y}_u(t)$ and $\tilde{y}_v(t)$ be the rate at which transactions are serviced from the queue at each of the routers (see Fig. 20 for an illustration). Eq. (45) can now be rewritten as $\tilde{x}_u(t)\Delta + \tilde{x}_v(t)\Delta - c_{(u,v)}$

$$= \left(\frac{\tilde{x}_u(t)}{\tilde{y}_u(t)} \right) \tilde{y}_u(t)\Delta + \left(\frac{\tilde{x}_v(t)}{\tilde{y}_v(t)} \right) \tilde{y}_v(t)\Delta - c_{(u,v)} \quad (46)$$

$$= \left(\frac{\tilde{x}_u(t)}{\tilde{y}_u(t)} \right) i_u(t) + \left(\frac{\tilde{x}_v(t)}{\tilde{y}_v(t)} \right) i_v(t) - c_{(u,v)}, \quad (47)$$

where $i_u(t)$ and $i_v(t)$ are the amount of funds that are currently locked at routers v and u respectively (Fig. 20). Since the funds used when servicing transactions at router u require Δ seconds on average to become available at v , by Little’s law the product of the average service rate $\tilde{y}_u(t)$ and average delay Δ is equal to the average amount of pending transactions $i_u(t)$ at v . Thus, Eq. (47) follows from Eq. (46). However, each of the terms in Eq. (47)—the transaction arrival rates $\tilde{x}_u(t)$, $\tilde{x}_v(t)$, service rates $\tilde{y}_u(t)$, $\tilde{y}_v(t)$, amount of pending transactions $i_u(t)$, $i_v(t)$ —can now be readily estimated at the channel.

Intuitively, since $i_u(t)$ is the amount of pending funds at router v when transactions are being serviced at a rate $\tilde{y}_u(t)$, $\tilde{x}_u(t)i_u(t)/\tilde{y}_u(t)$ is an estimate of the amount of transactions that will be pending if transactions were serviced at a rate $\tilde{x}_u(t)$. As the total amount of pending transactions in the channel cannot exceed the total amount of funds escrowed $c_{(u,v)}$, the difference $\tilde{x}_u(t)i_u(t)/\tilde{y}_u(t) + \tilde{x}_v(t)i_v(t)/\tilde{y}_v(t) - c_{(u,v)}$ is exactly the additional amount of funds required in the channel to support the current rates of transaction arrival. Denoting $\tilde{x}_u(t)i_u(t)/\tilde{y}_u(t)$ as $m_u(t)$ and $\tilde{x}_v(t)i_v(t)/\tilde{y}_v(t)$ as $m_v(t)$, the equation for updating λ at the routers can be written as

$$\lambda_{(u,v)}(t+1) = [\lambda_{(u,v)}(t) + \eta (m_u(t) + m_v(t) - c_{(u,v)} + \beta \min(q_u(t), q_v(t)))]_+, \quad (48)$$

where the $\beta \min(q_u(t), q_v(t))$ term has been included to ensure the queue sizes are small as discussed in §6.3.

F Extended Circulation Results

F.1 Fairness of Schemes

In §7.3, we show that Spider outperforms state-of-the art schemes on the success ratio achieved for a given channel capacity. Here, we break down the success volume by flows (sender-receiver pairs) in order to understand whether all pairs of nodes transacting on the PCN experience improvement

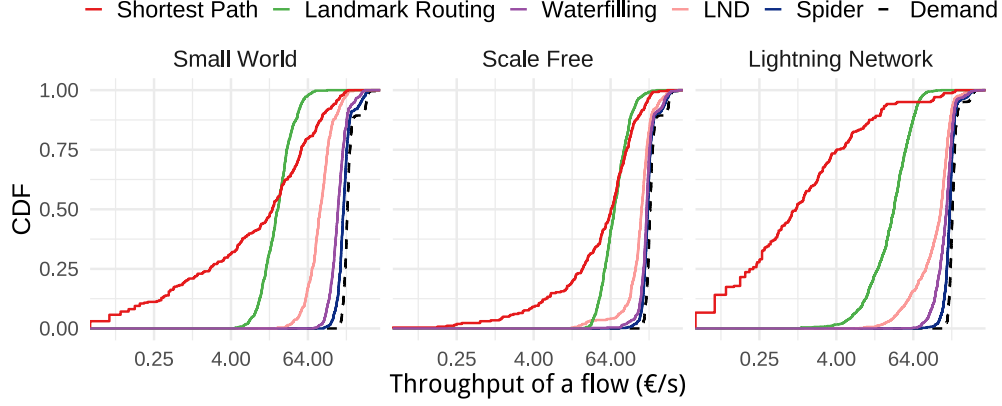


Figure 21: CDF of normalized throughput achieved by different flows under different schemes across topologies. Spider achieves close to 100% throughput given its proximity to the black line denoting demand of flows. Spider is a more vertical line than the baseline LND scheme showing that it is fairer than other schemes: it doesn't hurt the throughput of smaller flows to attain good overall throughput.

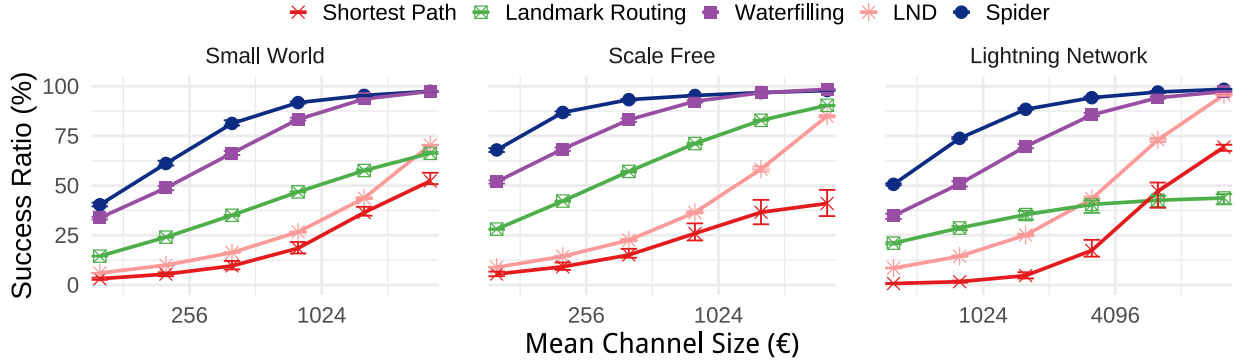


Figure 22: Performance of different algorithms on different topologies with equal channel sizes with different per sender transaction arrival rates. Spider consistently outperforms all other schemes achieving near 100% average success ratio. Error-bars denote the maximum and minimum success ratio across five runs. Note the log scale of the x-axes.

in throughput or if some of them are starved. Fig. 21 shows a CDF of the absolute throughput in €/s achieved by different protocols on a single circulation demand matrix where each sender sends an average of 30 transactions/s on synthetic and real topologies with channel sizes sampled from the LCSD. The mean channel sizes for the synthetic topologies and the real topologies are 4000€ and 16880€ respectively. Similar to §7.3, we run each protocol for 1010s and measure the success volume for transactions arriving between 800-1000s. There are two things to note here: (a) Spider achieves close to 100% throughput in all three scenarios, (b) Spider is fairer to small flows (most vertical line) and doesn't hurt the smallest flows just to benefit on throughput. This is not as true for the baseline LND scheme.

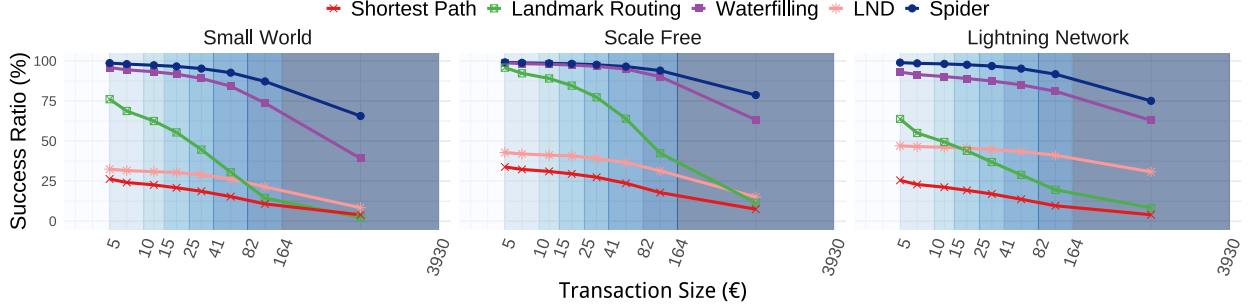


Figure 23: Breakdown of performance of different schemes by size of transactions completed. Each point reports the success ratio for transactions whose size belongs to the interval denoted by the shaded region. Each interval corresponds roughly to 12.5% of the CDF denoted in Fig. 7a. The graphs correspond to the (right) midpoints of the corresponding Lightning sampled channel sizes in Fig. 9.

F.2 Uniform Channel Size Distribution

We run five circulation traffic matrices for 1010s on our three topologies with all channels initialized with equal capacity denoted by the mean channel size. We measure the average success ratio for transactions arriving between 800-1000s for different payment channel sizes.

Fig. 22 shows that across all topologies, Spider outperforms the state-of-the-art schemes. Spider is able to successfully route more than 95% of the transactions with less than 25% of the capacity required by LND. Further Fig. 23 shows that Spider completes nearly 50% more of the largest 12.5% of the transactions attempted in the PCN across all three topologies. Even the waterfilling heuristic outperforms LND by 15-20% depending on the topology.

G DAG Workload On Synthetic Topologies

Fig. 24 shows the effect of adding a DAG component to the transaction demand matrix on the synthetic small world and scale free topologies. A demand of 30 transactions per second per sender with 5%, 20%, 40% DAG components respectively is sent through these PCN topologies for 1000s. We observe the average success ratio and normalized throughput that the different schemes achieve across five such traffic matrices as measured for transactions arriving between 800 and 1000s. No scheme is able to achieve the maximum throughput. However, the achieved throughput is closer to the maximum when there is a smaller component of DAG in the demand matrix. This suggest that the DAG affect PCN balances in a way that also prevents the circulation from going through. We investigate what could have caused this and how pro-active on-chain rebalancing could alleviate this in §7.4.

Fig. 25 shows the success ratio and normalized throughput achieved by different schemes when rebalancing is enabled for the 20% DAG traffic demand from Fig. 24. Spider is able to achieve over 95% success ratio and 90% normalized throughput even when its routers balance only every 10,000 € while LND is never able to sustain more than 75% success ratio even when rebalancing for every 10€ routed. This implies that Spider makes PCNs more economically viable for both routers locking up funds in payment channels and end-users routing via them since they need far fewer on-chain rebalancing events to sustain high throughput and earn routing fees.

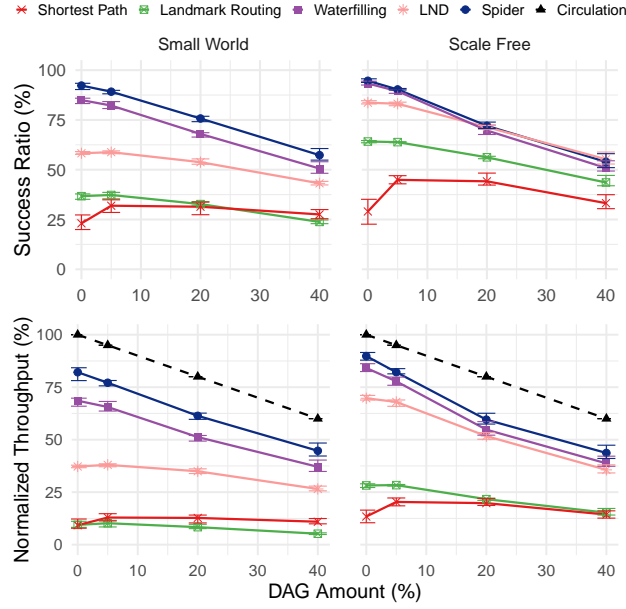


Figure 24: Performance of different algorithms across all topologies as the DAG component in the transaction demand matrix is varied. As the DAG amount is increased, the normalized throughput achieved is further away from the expected optimal circulation throughput. The gap is more pronounced on the real topology.

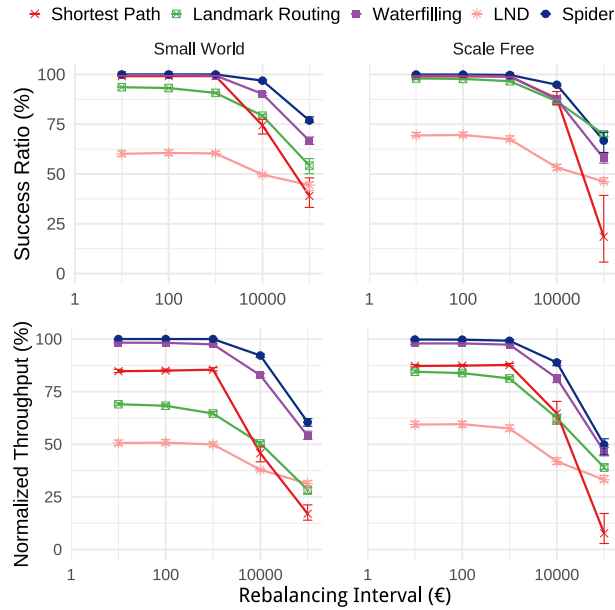


Figure 25: Performance of different algorithms across all topologies as the DAG component in the transaction demand matrix is varied. As the DAG amount is increased, the normalized throughput achieved is further away from the expected optimal circulation throughput. The gap is more pronounced on the real topology.