High Throughput Cryptocurrency Routing in Payment Channel Networks

Vibhaalakshmi Sivaraman, *Massachusetts Institute of Technology;* Shaileshh Bojja Venkatakrishnan, *Ohio State University;* Kathleen Ruan, *Carnegie Mellon University;* Parimarjan Negi and Lei Yang, *Massachusetts Institute of Technology;* Radhika Mittal, *University of Illinois at Urbana-Champaign;* Giulia Fanti, *Carnegie Mellon University;* Mohammad Alizadeh, *Massachusetts Institute of Technology*

https://www.usenix.org/conference/nsdi20/presentation/sivaraman

This paper is included in the Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20)

February 25–27, 2020 • Santa Clara, CA, USA

978-1-939133-13-7

Open access to the Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20) is sponsored by

High Throughput Cryptocurrency Routing in Payment Channel Networks

Vibhaalakshmi Sivaraman[*], Shaileshh Bojja Venkatakrishnan[**], Kathleen Ruan[†], Parimarjan Negi[*], Lei Yang [*], Radhika Mittal [‡], Giulia Fanti[†], Mohammad Alizadeh[*]

[*]Massachusetts Insititute of Technology, [**] Ohio State University,
[†] Carnegie Mellon University, [‡]University 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. Extensive simulations comparing Spider with state-of-the-art approaches shows that Spider requires less than 25% of the funds to successfully route over 95% of transactions on balanced traffic demands, and offloads 4x more transactions onto the PCN on imbalanced demands.

1 Introduction

Despite their growing adoption, cryptocurrencies suffer from poor scalability. For example, the Bitcoin [5] network processes 7 transactions per second, and Ethereum [14] 15 transactions/second, which pales in comparison to the 1,700 transactions per second achieved by the VISA network [56]. Scalability thus remains a major hurdle to the adoption of cryptocurrencies for retail and other 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 [43].

A leading proposal among many solutions to improve cryptocurrency scalability [23, 32, 40] 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 [10, 15], and Ethereum uses Raiden [18].

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 routing 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 either an on-chain transaction (i.e., committing a new transaction to the blockchain) or coordinated cyclic payments between a series of PCN nodes [39]. 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 [51]. Spider's design centers on two ideas that distinguish it from existing approaches. First, Spider senders "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 [59] developed for Internet congestion control. But the routing problem it solves in PCNs differs from standard networks in crucial 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 explicitly.

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 stall 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 [15]. Our evaluations (§7) show that (i) on circulation demands where 100% throughput is achievable, compared to the state-of-the-art, Spider requires 25% of the funds to route over 95% of the transactions and completes 1.3-1.8x more of the largest 25% of transactions based on a credit card transactions dataset [34]; (ii) on DAG demands where 100% throughput is not achievable, Spider offloads 7-8x as many transactions onto the PCN for every transaction on the blockchain, a 4x improvement over current approaches.

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 [46]. 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 [46].

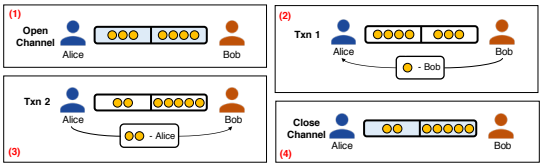


Figure 1: Bidirectional payment channel between Alice and Bob. A blue shaded block indicates a transaction that is committed to the blockchain.

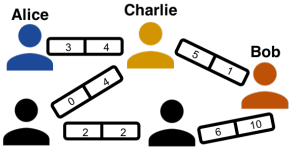


Figure 2: In a payment channel network, Alice can transfer money to Bob by using intermediate nodes' channels as relays. There are two paths from Alice to Bob, but only the path (Alice, Charlie, Bob) can support 3 tokens.

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 [18]. [1] Once Alice is ready to pay, she gives that key to

[1] The protocol called Hashed Timelock Contracts (HTLCs) can be implemented in two ways: the sender generates the key, as in Raiden [18] or the receiver generates the key, as in Lightning [46]. Spider assumes that the sender generates the key.

Bob out-of-band; 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 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 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. [2] 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→B on edge a and all traffic from cluster B→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. 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 transaction. The same applies to cluster B on edge b. Since on-chain transactions are expensive and slow, it is desirable to avoid them. Routing schemes like shortest-path routing do not account for this problem, thereby leading to reduced throughput (§7). In contrast, it is important to choose routes that actively prevent channel imbalance. For example, in Figure 3b, we could send half of the A→B traffic on edge a, and half on edge b, and the same for the B→A traffic. The challenge is making these decisions in a fully decentralized way.

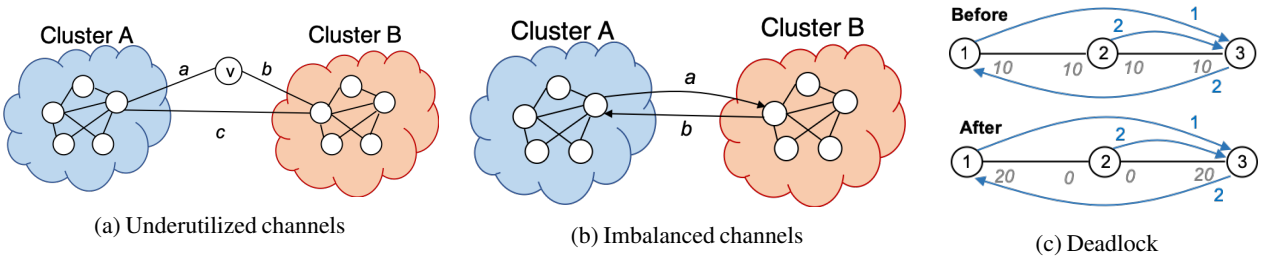
Challenge #2: When to send transactions? Another problem is *when* to send transactions. Most existing PCNs are circuit-switched: transactions are processed instantaneously and atomically upon arrival [18, 46]. This causes a number of problems. If a transaction's value exceeds the available balance on each path from the source to the destination, the transaction fails. Since transaction values in the wild tend to be heavy-tailed [29, 34], either a substantial fraction of real transactions will fail as PCN usage grows, or payment channel operators will need to provision 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 [54]. 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 is a large transaction in one direction on a channel, simply sending it out in smaller units that must all complete together doesn't improve the likelihood of success. Instead, in our design, we allow each transaction-unit to complete independently, and a congestion control algorithm at the sender throttles the rate of these units to match the rate of units in the opposite direction at the bottlenecked payment channel. This effectively allows the tokens at that bottleneck to be replenished and reused multiple times as part of the same transaction, achieving a multiplicative increase in throughput for the same collateral.

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 transact 2 units/sec with node 1.[3] 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 sustain 2 units/sec by only having nodes 1 and 3 to send to each other at a rate of 1 unit/second.

[2] This was done in part for privacy reasons: transactions in the Lightning network use onion-routing, which is easy to implement with source routing [33].

[3] For 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.



(a) Underutilized channels (b) Imbalanced channels (c) Deadlock

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

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 challenging 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: 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: (a) 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, (b) the delay it takes for the destination of a transaction to receive it and send back the secret key unlocking the funds at routers (§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 depends on its delay, and queued up transactions at a depleted link can hold up funds from channels in other parts of the network. 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 [41]. 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 [15] 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. We assume that the underlying communication network is not a bottleneck and PCN users can communicate payment attempts, success and failures with one another easily since these messages do not require much bandwidth.

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 [45]. 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 with $c_{u,v}$ funds in total 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 [31, 37, 38]. A logarithmic utility ensures that the rate allocations are proportionally fair [38]—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

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

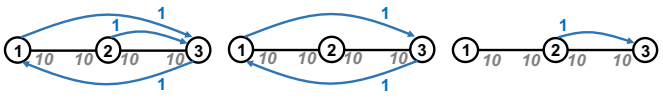
$$\text{s.t.} \quad \sum_{p \in \mathcal{P}_{i,j}} x_p \leq d_{i,j} \quad \forall i,j \in V \tag{2}$$

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

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

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

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.



(a) Payment graph (b) Circulation (c) DAG

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).

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 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.

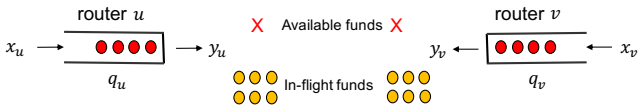
For instance, Fig. 4a shows the traffic demand graph for a PCN with nodes $\{1,2,3\}$ and payment channels between nodes $1-2$ and $2-3$. The weight on each blue edge denotes the demand in transaction-units per second between a pair of users. The underlying black lines denote the topology and channel 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.

App. A formalizes the notion of circulation and shows that the maximum throughput achievable by any balanced routing scheme is at most the total demand contained within the circulation.

6 Design

6.1 Intuition

Spider routers queue up transactions at a payment channel whenever the channel lacks funds to forward them immediately (§5). Thus, queue buildup is a sign that either transaction-units



(a) A capacity limited payment channel.

(b) An imbalance limited payment channel.

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 exceeds the channel capacity, neither router has available funds and queues build up at both u and v. (b) If the arrival rates are imbalanced, *e.g.,* if $x_v > x_u$, then u has excess funds while v has none, causing queue build-up at v.

are arriving faster (in both directions) than the channel can process (Fig. 5a) or that one end of the payment channel lacks sufficient funds(Fig. 5b). It indicates that the capacity constraint (Equation 3) or the balance constraint (Equation 4) is being violated and the sender should adjust its sending rate.

Therefore, if senders use a congestion control protocol that controls queues, they could detect both capacity and imbalance violations and react to them. For example, in Fig. 5a, the protocol would throttle both x_u and x_v. In Fig. 5b, it would decrease x_v to match 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 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 [19], Cubic [35], DCTCP [22], Vegas [27], etc.) and could be adapted for PCNs.

In PCNs, it is desirable to transmit transaction-units along multiple paths to better utilize available capacity. Consequently, Spider's design is inspired by multi-path transport protocols like MPTCP [59]. These protocols couple rate control decisions for multiple paths to achieve both high throughput and fairness among competing flows [58]. 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

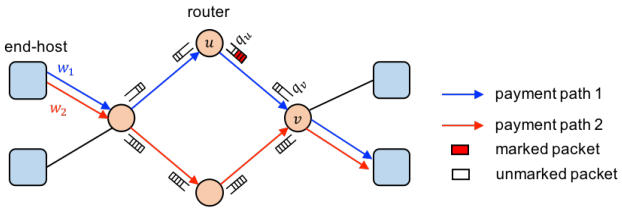


Figure 6: Routers queue up transaction-units and schedule them based on priorities when funds become available. and transaction priorities. If the delay through the queue for a packet exceeds a threshold, they mark the packet. End-hosts maintain and adjust windows for each path to a receiver based on the marks they observe.

packet spends in their queue and mark the packet if the time spent exceeds a pre-determined threshold T. If the transaction-unit is already marked, routers leave the field unchanged and merely forward the transaction-unit. Routers forward acknowledgments from the receiving end-host back to the sender which interprets the marked bit in the ack accordingly. Spider routers schedule transaction-units from their queues according to a scheduling policy, like Smallest-Payment-First or Last-In-First-Out (LIFO). Our evaluations (§7.5) shows that LIFO provides the highest transaction success rate. The idea behind LIFO is to prioritize transaction units from new payments, which are likely to complete within their deadline.

6.3 Spider Transport Layer at End-Hosts

Spider senders send and receive payments on a PCN by interfacing with their transport layer. This layer is configured to support both atomic and non-atomic payments depending on user preferences. Non-atomic payments utilize Spider's packet-switching which breaks up large payments into transaction-units that are delivered to the receiver independently. In this case, senders are notified of how much of the payment was completed allowing them to cancel the rest or retry it on the blockchain. While this approach crucially allows token reuse at bottleneck payment channels for the same transaction (§3), senders also have the option of requesting atomic payments (likely for a higher fee). Our results (§7) show that even with packetization, more than 95% payments complete in full

The transport layer also involves a multi-path protocol which controls the rates at which payments are transferred, based on congestion in the network. For each destination host, a sender chooses a set of k paths to route transaction-units along. The route for a transaction-unit is decided 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 [17, 33]. In §7.5, we evaluate the impact of different path choices on Spider's performance and propose using edge-disjoint widest paths [21] between each sender and receiver in Spider.

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 yet been acked or canceled. 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

$$w_p \leftarrow w_p - \beta, \quad \text{on every marked packet and,} \tag{6}$$

$$w_p \leftarrow w_p + \frac{\alpha}{\sum_{p':p' \in \mathcal{P}_{i,j}} w_{p'}}, \quad \text{on every unmarked packet.} \tag{7}$$

Here, α and β are both positive constants that denote the aggressiveness with which the window size is increased and decreased respectively. Eq. (6)–(7) are similar to MPTCP, but with a multiplicative decrease factor that depends on the fraction of packets marked on a path (similar to DCTCP [22]).

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 cancels the payment, clearing all of its state from the PCN. In particular, it sends a cancellation message to remove any transaction-units queued at routers on each path to the receiver. Notice that transaction-units that arrive at the receiver in the meantime cannot be unlocked because we assume the sender holds the secret key (§2). Senders can then choose to retry the failed portion of the transaction again on the PCN or on the blockchain; such retries would be treated as new transactions. Canceled packets are considered marked and Spider decreases its window in response to them.

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. 20 in App. 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 queuing. 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 (6) and (7). We elaborate more on this fluid model, including specifying how the queue sizes and marking fractions evolve, in App. 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 [26]. The complete and formal mathematical proof showing the above is presented in App. 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 describe our simulation setup (§7.1), validate it via a prototype implementation (§7.2), and present detailed results for circulation demands (§7.3). We then show the effect of adding DAG components to circulations (§7.4), and study 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 can't view channel balances, but they do know each channel's size or total number of tokens (€). 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 route, while correspondingly decrementing or incrementing the channel balances. Funds consumed by a transaction in a channel are *inflight* and unavailable until 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* that is serviced in a last in first out (LIFO) order §7.5. If the queue is full, an incoming transaction is dropped, and a failure message is sent to the sender.

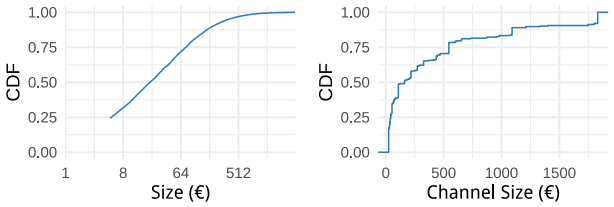
Routing Schemes. We implement and evaluate five different routing schemes in our simulator.

(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 amount inflight on the path. 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 queuing 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.

[4] https://github.com/spider-pcn/spider-omnet

(2) Waterfilling: Waterfilling uses balance information explicitly in contrast to Spider's 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 per path that computes the bottleneck (minimum) channel balance along it. When a path's probe is received, the sender computes the available balance based on its bottleneck and the in-flight transaction-units. A transaction-unit is sent along the path with the highest available balance. If the available balance for all of the k paths is zero (or less), the transaction-unit is queued and retried after the next probe.
(3) Shortest Path: This baseline sends transactions along the shortest path to the destination without transaction splitting.
(4) Landmark Routing: Landmark routing, as used in prior PCN routing schemes [42, 47, 50], chooses k well-connected *landmark* nodes in the topology. For every transaction, the sender computes its shortest path to each landmark and concatenates it with the shortest path from that landmark to the destination to obtain k distinct paths. Then, the sender probes each path to obtain its bottleneck balance, and partitions the transaction 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, the transaction fails.
(5) LND: The PCN scheme currently deployed in the Lightning Network Daemon (LND) [15] attempts first send a transaction 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 until the destination becomes unreachable or the transaction times out. A channel is added back to the local view 5 seconds after its removal.
(6) Celer: App. C.1 compares Spider to Celer's cRoute as proposed in a white-paper [11]. Celer is a back-pressure routing algorithm that routes transactions based on queue and imbalance gradients. Due to computation overheads associated with Celer's large queues, we evaluate it on a smaller topology.
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 generated by adding x random permutation matrices; (ii) circulations with a DAG component, having a total rate y generated by sampling y different sender-receiver pairs where senders and receivers are chosen from two separate exponential distributions. The distribution's skew is set proportional to the desired DAG component in the total traffic matrix.

We translate the rates from the payment graph to discrete transactions with a Poisson arrival process The transaction size distribution (Fig. 7a) is drawn from credit card transaction data [34], and has a mean of 88€ and median 25€ with the largest transaction being 3930€. Each sender sends 30 tx/sec on average shared across 10 destinations. Note that a sender represents a router in our setup, sending transactions to other routers on behalf of many users.
Topology. We set up an LND node [15] to retrieve the Lightning Network topology on July 15, 2019. We snowball sample [36] 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 set 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 RTTs of about a second.



(a) Transaction Size Distribution (b) LN Channel Size Distribution

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

We additionally simulate two synthetic topologies: a Watts-Strogatz small world topology [20] with 50 nodes and 200 edges, and a scale-free Barabasi-Albert graph [4] with 50 nodes and 336 edges. We set the per-hop delay to 30ms in both cases, resulting in RTTs 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 experiments by proportionally scaling up the size of each payment channel. All payment channels are initialized with perfect balance.
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 to for faster response to congestion.
Metrics. We use the following evaluation metrics: (i) *transaction success ratio:* the number of completed transactions over the number of generated transactions. A packetized transaction is complete when all of its transaction-units are successful, (ii) *normalized throughput:* the total amount of payments (in €) completed over the total amount of payments generated, (iii) *transaction latency*: time between arrival and completion for successful transactions, and (iv) *offload factor*: number of transactions offloaded to the PCN for every on-chain transaction. All of these metrics are computed over a measurement

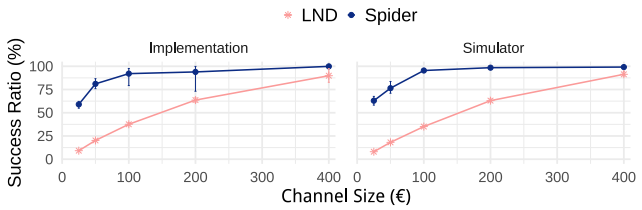


Figure 8: Comparison of performance on simulator and implementation for LND and Spider on a 10 node scale-free topology with 1€ transactions. Spider outperforms LND in both settings. Further, the average success ratio on the simulator and implementation for both schemes are within 5% of each other.

interval when all schemes are in steady-state. Unless specified otherwise, we use a measurement interval of 800-1000s, run experiments for 1010s, and denote the maximum and minimum statistic across five runs using error-bars.

7.2 Prototype Implementation

To support Spider, we modify the Lightning Network Daemon (LND) [15] 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 inflight 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 [15] 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 [6]. We create our own regtest [8] blockchain for the nodes. Channels are created 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 are generated with each sender sending 100 tx/s (each 1€). Receiving nodes communicate invoices via etcd [13] to sending nodes who then complete them using the appropriate scheme. We run LND and Spider on the implementation and measure the transaction RTTs to inform propagation delays on the simulator. We then run the same experiments on the simulator.

Fig. 8 shows the average success ratio that Spider and LND achieve on the implementation and the simulator. There are two 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 for high throughput. For example, persisting HTLC state on disk causes IO bottlenecks and variations of tens of seconds in transaction latencies even on small topologies. Given the fidelity and flexibility of the simulator, we chose to use it for the remaining evaluations.

7.3 Circulation Payment Graph Performance

Recall that on circulation payment graphs, *all* the demand can theoretically be routed if there is sufficient capacity (§5.1 and App. A). However, the capacity at which a routing scheme 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 success ratio for the transactions across different channel sizes. Fig. 9 shows that on all topologies, Spider outperforms the state-of-the-art schemes. Spider successfully routes more than 95% of the transactions with less than 25% of the capacity required by LND. At lower capacities, Spider completes 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. The explicit balance-aware scheme, Waterfilling, also routes more transactions than LND. However, when operating in low capacity regimes, where many paths are congested and have near-zero available balance, senders are unable to use just balance information to differentiate paths. As a result, Waterfilling's performance degrades at low capacity compared to Spider which takes into account queuing delays.

Size of Successful Payments. Spider's benefits are most pronounced at larger transaction sizes, where packetization and congestion control helps more transactions complete. 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, each corresponding to about 12.5% of the transactions in the workload. 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.

Impact on Latency. We anticipate Spider's rate control mechanism to increase latency. Fig. 11 shows the average and 99^{th} percentile latency for successful transactions on the Lightning topology as a function of transaction size. Spider's average and tail latency increase with transaction size because larger transactions are multiplexed over longer periods of time. However, the tail latency increases much more than the average because of the skew in channel sizes in the Lightning topology: most transactions use large channels while a few unfortunate large transactions need more time to reuse tokens from smaller channels. Yet, the largest Spider transactions experience at most 2 seconds of additional delay when

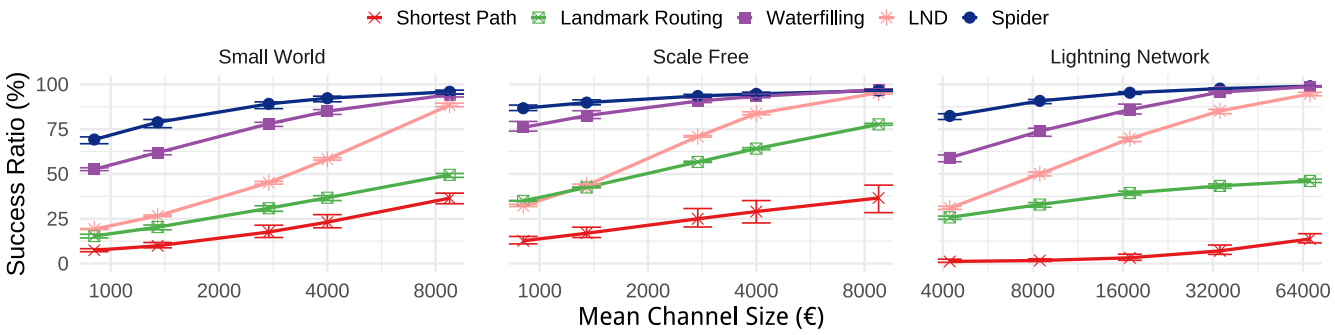


Figure 9: Performance of different algorithms on small-world, scale-free and Lightning Network topologies, for different per sender transaction arrival rates. Spider consistently outperforms all other schemes achieving near 100% average success ratio. Note the log scale of the x-axes.

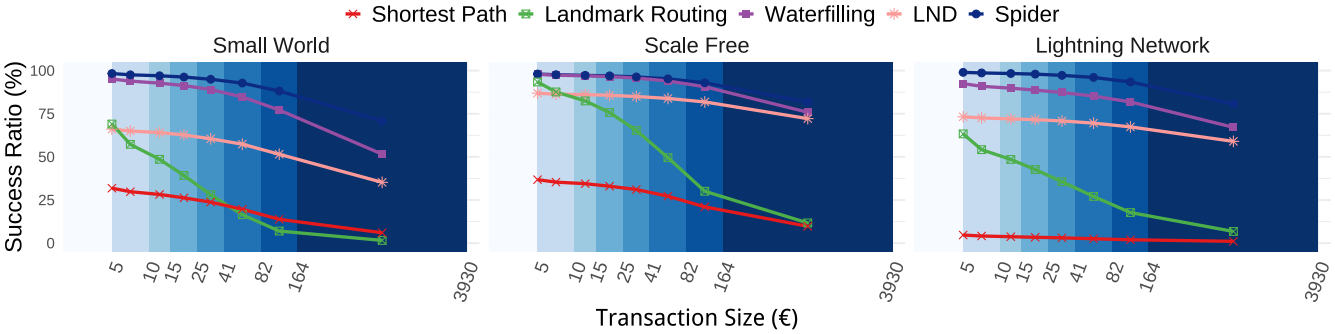


Figure 10: Breakdown of performance of different schemes by size of transactions completed. Each point reports the success ratio for transactions whose size belongs to the interval denoted by the shaded region. Each interval corresponds roughly to a 12.5% weight in the transaction size CDF shown in Fig.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

compared to LND, a small hit relative to the 20% increase in overall success ratio at a mean channel size of 16880€. LND's latency also increases with size since it retries transactions, often upto 10 times until it finds a single path with enough capacity. In contrast, Landmark Routing and Shortest path are size-agnostic in their path-choice for transactions.

Waterfilling pauses transactions when there is no available balance and resumes sending when balance becomes available. Small transactions are unlikely to be paused in their lifetime while mid-size transactions are paused a few times before they complete. In contrast, large transactions are likely to be paused many times, eventually getting canceled if paused too much. This has two implications: (i) the few large transactions that are successful with Waterfilling are not paused much and contribute smaller latencies than mid-size transactions, and (ii) Waterfilling's conservative pause and send mechanism implies there is less contention for the large transactions that are actually sent into the network, leading to smaller latencies than what they experience with Spider.

7.4 Effect of DAGs

Real transaction demands are often not pure circulations: consumer nodes spend more, and merchant nodes receive

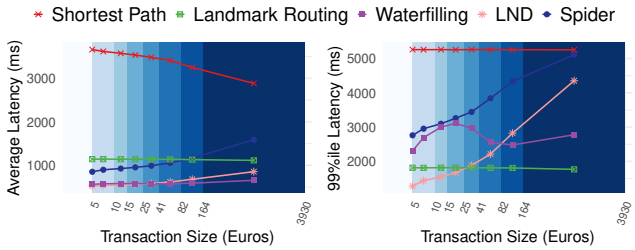


Figure 11: Average and 99%ile transaction latency for different routing schemes on the Lightning topology. Transactions experience 1-2s of additional latency with Spider relative to LND for a 20% improvement in throughput.

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 App. C.4.

Fig. 12 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

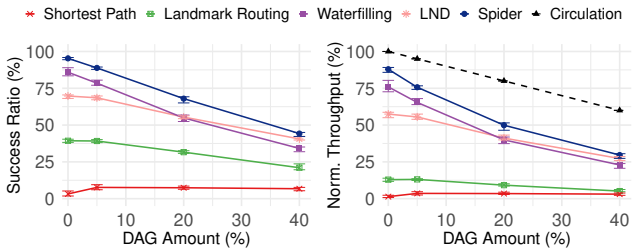


Figure 12: Performance of different algorithms on the Lightning topology as the DAG component in the transaction demand matrix is varied. As the DAG amount is increased, the normalized throughput achieved is further away from the expected optimal circulation throughput.

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. 13 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 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 trade-off for the router.

Fig. 14 shows the success ratio and normalized throughput achieved by different schemes when rebalancing is enabled for the traffic demand with 20% DAG from Fig. 12, or Fig. 13. 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

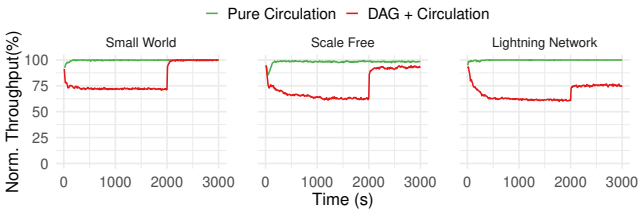


Figure 13: Comparing throughput when a pure circulation demand is run for 3000s to a scenario where a circulation demand is restored for 1000s after 2000s of a demand with 20% DAG. The throughput achieved on the last 1000s of circulation is not always the expected 100% even after the DAG is removed.

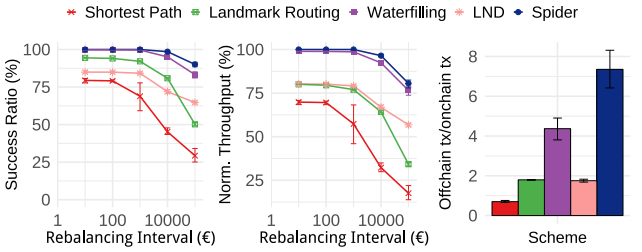


Figure 14: Performance of different algorithms on the Lightning topology when augmented with on-chain rebalancing. Spider needs less frequent rebalancing to sustain high throughput. Spider offloads 3-4x more transactions onto a PCN per blockchain transaction than LND.

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€ [7] 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 [12]. Fig. 14 also captures the same result in the form of the best offloading or number of off-chain PCN transactions per blockchain transaction achieved by each algorithm. Transactions that fail on the PCN as well as rebalancing transactions are counted towards the transactions on the blockchain. Spider is able to route 7-8 times as many transactions off-chain for every blockchain transaction, a 4x improvement from the state-of-the-art LND.

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 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 [60], oblivious paths [48] and

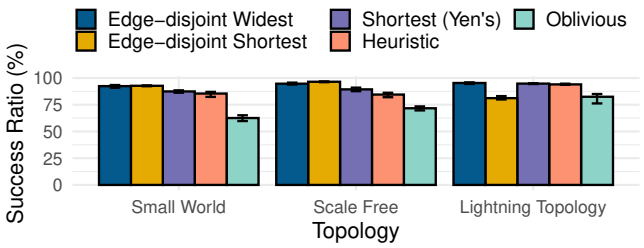


Figure 15: Performance of Spider as the type of paths considered per sender-receiver pair is varied. Edge-disjoint widest outperforms others by 1-10% on the Lightning Topology without being much worse on the synthetic topologies.

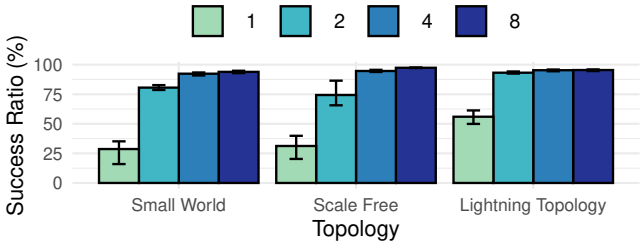


Figure 16: Performance of Spider as the number of edge-disjoint widest paths considered per sender-receiver pair is varied on different topologies. Increasing the number of paths increases success ratio, but the gains are low in going from 4 to 8 paths.

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. 15 shows that edge-disjoint widest paths outperforms other approaches by 1-10% on the Lightning Topology while being only 1-2% worse that 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. 16 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 30-50% 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 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-Payment-First (SPF) in addition to the LIFO baseline. Fig. 17 shows that LIFO achieves a success ratio that is 10-28% higher than its counterparts. This is because

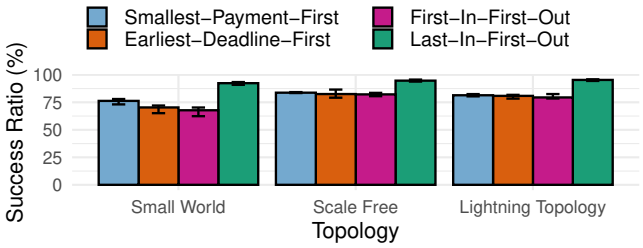


Figure 17: Performance of Spider as the scheduling algorithm at the sender and router queues is varied. Last in first out outperforms all other approaches by over 10% on all topologies.

LIFO prioritizes transactions that are newest or furthest from their deadlines and thus, most likely complete especially when the PCNs is overloaded. 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 compare Spider to Celer, as proposed in a white-paper [11], and show that Spider outperforms Celer's success ratio by 2x on a scale free topology with 10 nodes and 25 edges (App. C.1).
2. 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 (App. C.2).
3. We evaluate the schemes for their fairness across multiple payments and show that Spider does not hurt small payments to gain on throughput (App. C.3).
4. 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 (App. C.4).

8 Related Work

PCN Improvements. Nodes in current Lightning Network implementations, maintain a local view of the network topology and source-route transactions along the shortest path [2, 15]. Classical max-flow-based alternatives are impractical for the Lightning Network that has over 5000 nodes and 30,000 channels [9, 16] due to their computational complexity. Recent proposals have used a modified version of max-flow that differentiates based on the size of transactions [57]. 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 only route transactions to a landmark [55]. This approach is used in Flare [47] and SilentWhispers [42, 44]. *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 [49] and SpeedyMurmurs [50] 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 and prior work [51] show that Spider outperforms both approaches.

PCN improvements outside of the routing layer focus on rebalancing existing payment channels more easily [28, 39]. Revive [39] leverages cycles within channels wanting to rebalance and initiates balancing off-chain payments between them. These techniques are complementary to Spider and can be used to enhance overall performance. However, §7.4 shows that a more general rebalancing scheme that moves funds at each router independently fails to achieve high throughput without a balanced routing scheme.

Utility Maximization and Congestion Control. Network Utility Maximization (NUM) is a popular framework for developing decentralized transport protocols in data networks to optimize a fairness objective [37]. 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 buffer sizes or queuing delays as router signals [22, 30, 53] 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 ride-sharing, for instance [24, 25], and require new innovation in both formulating and solving routing problems.

9 Conclusion

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 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 degrades circulation throughput, necessitating on-chain intervention. In such scenarios, Spider is able to support 4x more transactions than the state-of-the-art on the PCN itself.

This work shows that Spider needs less on-chain rebalancing to relieve deadlocked PCNs. However, it remains to be seen if deadlocks can be prevented altogether. Spider relies on routers signaling queue buildup correctly to the senders, but this work does not analyze incentive compatibility for rogue routers aiming to maximize fees. A more rigorous treatment of the privacy implications of Spider routers relaying queuing delay is left to future work.

Acknowledgments

We thank Andrew Miller, Thaddeus Dryja, Evan Schwartz, Vikram Nathan, and Aditya Akella for their detailed feedback. We also thank the Sponsors of Fintech@CSAIL, the Initiative for CryptoCurrencies and Contracts (IC3), Distributed Technologies Research Foundation, the Cisco Research Center, the National Science Foundation grants CNS-1718270, CNS-1563826, CNS-1910676, CCF-1705007 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] Barabasi Albert Graph. https://networkx.github.io/documentation/networkx-1.9.1/reference/generated/networkx.generators.random_graphs.barabasi_albert_graph.html.

[5] Bitcoin Core. https://bitcoin.org/en/bitcoin-core/.

[6] Bitcoin Core Daemon. https://bitcoin.org/en/full-node#other-linux-daemon.

[7] Bitcoin historical fee chart. https://bitinfocharts.com/comparison/bitcoin-median_transaction_fee.html.

[8] Bitcoin Regtest Mode. https://bitcoin.org/en/developer-examples#regtest-mode.

[9] Blockchain caffe. https://blockchaincaffe.org/map/.

[10] c-lightning: A specification compliant Lightning Network implementation in C. https://github.com/ElementsProject/lightning.

[11] Celer Network: Bring Internet Scale to Every Blockchain. https://www.celer.network/doc/CelerNetwork-Whitepaper.pdf.

[12] Credit Card Merchant Processing Fees. https://paymentdepot.com/blog/average-credit-card-processing-fees/.

[13] etcd: A distributed, reliable key-value store for the most critical data of a distributed system. https://github.com/etcd-io/etcd.

[14] Ethereum. https://www.ethereum.org/.

[15] Lightning Network Daemon. https://github.com/lightningnetwork/lnd.

[16] Lightning Network Search and Analysis Engine. https://1ml.com.

[17] Onion Routed Micropayments for the Lightning Network. https://github.com/lightningnetwork/lightning-onion.

[18] Raiden network. https://raiden.network/.

[19] The NewReno Modification to TCP's Fast Recovery Algorithm. https://tools.ietf.org/html/rfc6582.

[20] Watts Strogatz Graph. https://networkx.github.io/documentation/networkx-1.9/reference/generated/networkx.generators.random_graphs.watts_strogatz_graph.html.

[21] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, 1993.

[22] 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.

[23] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath. Deconstructing the blockchain to approach physical limits. *arXiv preprint arXiv:1810.08092*, 2018.

[24] 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.

[25] S. Banerjee, R. Johari, and C. Riquelme. Dynamic pricing in ridesharing platforms. *ACM SIGecom Exchanges*, 15(1):65–70, 2016.

[26] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[27] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. *TCP Vegas: New techniques for congestion detection and avoidance*, volume 24. ACM, 1994.

[28] C. Burchert, C. Decker, and R. Wattenhofer. Scalable funding of bitcoin micropayment channel networks. *Royal Society open science*, 5(8):180089, 2018.

[29] C. N. Cordi. *Simulating high-throughput cryptocurrency payment channel networks*. PhD thesis, 2017.

[30] N. Dukkipati. *Rate Control Protocol (RCP): Congestion control to make flows complete quickly*. Citeseer, 2008.

[31] 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.

[32] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.

[33] D. Goldschlag, M. Reed, and P. Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.

[34] U. M. L. Group. Credit card fraud detection, 2018. https://www.kaggle.com/mlg-ulb/creditcardfraud.

[35] 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.

[36] P. Hu and W. C. Lau. A survey and taxonomy of graph sampling. *arXiv preprint arXiv:1308.5865*, 2013.

[37] 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.

[38] 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.

[39] 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.

[40] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018.

[41] 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.

[42] 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.

[43] R. McManus. Blockchain speeds & the scalability debate. *Blocksplain*, February 2018.

[44] P. Moreno-Sanchez, A. Kate, M. Maffei, and K. Pecina. Privacy preserving payments in credit networks. In *Network and Distributed Security Symposium*, 2015.

[45] 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.

[46] J. Poon and T. Dryja. The Bitcoin Lightning Network: Scalable Off-chain Instant Payments. *draft version 0.5*, 9:14, 2016.

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

[48] 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.

[49] 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.

[50] 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.

[51] V. Sivaraman, S. B. Venkatakrishnan, M. Alizadeh, G. Fanti, and P. Viswanath. Routing cryptocurrency with the spider network. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, pages 29–35. ACM, 2018.

[52] R. Srikant. *The mathematics of Internet congestion control*. Springer Science & Business Media, 2012.

[53] C.-H. Tai, J. Zhu, and N. Dukkipati. 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.

[54] S. Thomas and E. Schwartz. A protocol for interledger payments. *URL https://interledger. org/interledger. pdf*, 2015.

[55] 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.

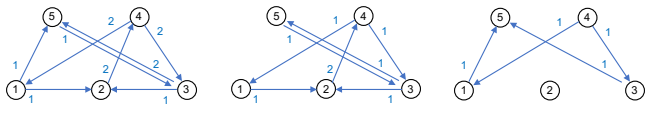
[56] Visa. Visa acceptance for retailers. https://usa.visa.com/run-your-business/small-business-tools/retail.html.

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

[58] D. Wischik, M. Handley, and M. B. Braun. The resource pooling principle. *ACM SIGCOMM Computer Communication Review*, 38(5):47–52, 2008.

[59] 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.

[60] J. Y. Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.



(a) Payment graph (b) Circulation (c) DAG

Figure 18: 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).

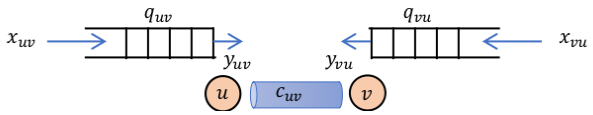


Figure 19: Model of queues at a payment channel between nodes u and v. x_{uv} and y_{uv} denote the rates at which transaction-units for v arrive into and get serviced at the queue at u respectively. c_{uv} is the capacity of the payment channel and q_{uv} denotes the total number of transaction-units waiting in u's queue to be serviced.

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 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 $\nu(C^*)$. □

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

$$\dot{x}_p(t) = \left[\frac{x_p(t)}{\sum_{p' \in \mathcal{P}_{i_p,j_p}} x_{p'}(t)} - \sum_{(u,v)\in p} f_{u,v}(t) x_p(t) \right]^+_{x_p(t)} \quad \forall p \in \mathcal{P} \tag{8}$$

$$\dot{q}_{u,v}(t) = [x_{u,v}(t) - y_{u,v}(t)]^+_{q_{u,v}(t)} \quad \forall (u,v) \in E \tag{9}$$

$$\dot{f}_{u,v}(t) = [q_{u,v}(t) - q_{\text{thresh}}]^+_{f_{u,v}(t)} \quad \forall (u,v) \in E, \tag{10}$$

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

$$\begin{cases} \frac{c_{u,v}}{2\Delta} & \text{if } q_{u,v}(t) > 0 \ \&\ q_{v,u}(t) > 0 \\ \min\{\frac{c_{u,v}}{2\Delta}, x_{v,u}(t)\} & \text{if } q_{u,v}(t) > 0 \ \&\ q_{v,u}(t) = 0 \\ \min\{\frac{c_{u,v}}{2\Delta}, x_{u,v}(t)\} & \text{if } q_{u,v}(t) = 0 \ \&\ q_{v,u}(t) > 0 \\ \min\{\frac{c_{u,v}}{2\Delta}, x_{u,v}(t), x_{v,u}(t)\} & \text{if } q_{u,v}(t) = 0 \ \&\ q_{v,u}(t) = 0 \end{cases} \tag{11}$$

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 (8) models how the rate on a path p increases upon receiving successful acknowledgements or decreases if the packets are marked, per Equations (6) and (7) 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 $\sum_{(u,v)\in p} f_{u,v}$ [52]. 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 (9) and (10) 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 19). 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 (10). 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 (11)), 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 $c_{u,v}/(2\Delta)$ 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 (11).

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.

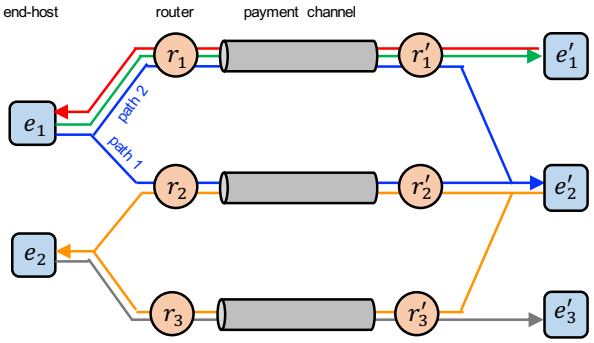


Figure 20: Example of a parallel network topology with bidirectional flows on each payment channel.

B.2 Proof of Optimality

Consider a PCN comprising of two sets of end-hosts $\{e_1,\ldots,e_m\}$ and $\{e_1',\ldots,e_n'\}$ that are connected via k parallel payment channels $(r_1,r_1'),\ldots,(r_k,r_k')$ as shown in Figure 20. The end-hosts from each set have demands to end-hosts on the other set. The end-hosts within a set, however, do 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 (8)) must be non-positive, *i.e.*,

$$\frac{1}{\sum_{p'\in\mathcal{P}_{i_p,j_p}} x_{p'}^*} - \sum_{(u,v)\in p} f_{u,v}^* \begin{cases} =0 & \text{if } x_p^*>0 \\ \le 0 & \text{if } x_p^*=0 \end{cases} \quad \forall p\in\mathcal{P}, \quad (12)$$

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

$$x_{u,v}^* \begin{cases} =y_{u,v}^* & \text{if } q_{u,v}^*>0 \\ \le y_{u,v}^* & \text{if } q_{u,v}^*=0 \end{cases} \quad \forall (u,v)\in E. \quad (13)$$

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 (13) 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 $x_{u,v}^*\le y_{u,v}^*$ from Equation (13) and $y_{u,v}(t)$ is at most $\frac{c_{u,v}}{2\Delta}$ from Equation (11). Similarly the

transaction arrival rates also satisfy the balance constraints in Equation (4). To see this, we first note the 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 (8). Therefore we have $x^*_{u,v} = y^*_{u,v}$ (from Equation (13)) for every channel. Combining this with $y_{u,v}(t) = y_{v,u}(t)$ (Equation (11)), 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 $\mu_{u,v}$

$$\lambda^*_{u,v} \leftarrow (f^*_{u,v} + f^*_{v,u})/2 \quad \forall (u,v) \in E \tag{14}$$

$$\mu^*_{u,v} \leftarrow f^*_{u,v}/2 \quad \forall (u,v) \in E, \tag{15}$$

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 (10)), 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-received 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 (13) and the first sub-case in Equation (11). 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

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

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

$$\sum_{(u,v) \in p} (\lambda^*_{u,v} + \mu^*_{u,v} - \mu^*_{v,u}) = \sum_{(u,v) \in p} \frac{f^*_{u,v} + f^*_{v,u}}{2} + \frac{f^*_{u,v}}{2} - \frac{f^*_{v,u}}{2}$$
$$= \sum_{(u,v) \in p} f^*_{u,v}, \tag{17}$$

where the first equation above follows from our mapping for $\lambda^*_{u,v}$ and $\mu^*_{u,v}$ in Equations (14), (15). Combining this with Equation (12), we see that $x_p \leftarrow x^*_p$ for all $p \in \mathcal{P}$ is a valid solution to the Equation (16). 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.

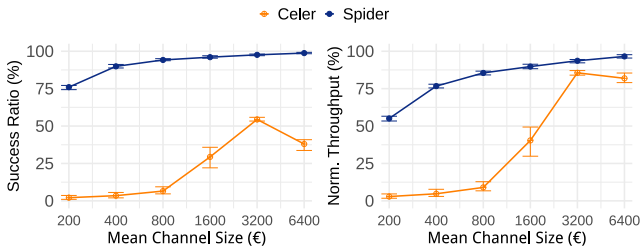


Figure 21: Spider's performance relative to Celer on a 10 node scale free topology. Spider achieves a 2x improvement in success ratio even at Celer's peak performance. Celer's performance dips after a peak since it maintains larger queues at higher capacities, eventually causing timeouts.

C Additional Results

C.1 Comparison with Celer

We run five circulation traffic matrices for 610s on a scale free topology with 10 nodes and 25 edges to compare Spider to Celer [11], a back-pressure based routing scheme. Each node sends 30 txns/s and we vary the mean channel size from 200€ to 6400 €. We measure the average success ratio and success volume for transactions in the 400-600s interval and observe that Spider outperforms Celer at all channel sizes. Celer splits transactions into transaction-units at the source but does not source-route individual transaction-units. Instead, transaction-units for a destination are queued at individual routers and forwarded on the link with the maximum queue and imbalance gradient for that destination. This approach tries to maximize transaction-units in queues to improve network utilization. However, queued-up and in-flight units in PCNs hold up tokens in other parts of the network while they are in-flight waiting for acknowledgements, reducing its capacity. Celer transactions also use long paths, sometimes upto 18 edges in this network with 25 edges. Consequently, tokens in Celer spend few seconds in-flight in contrast to the hundreds of milliseconds with Spider. The time tokens spent in-flight also increases with channel size since Celer tries to maintain larger queues. Celer's performance dips once the in-flight time has increased to the point where transactions start timing out before they can be completed. Due to computational constraints associated with large queues, we do not run Celer on larger topologies.

C.2 Circulations on Synthetic Topologies

We run five circulation traffic matrices for 1010s on our three topologies with all channels having exactly the tokens denoted by the channel size. Fig. 22 shows that across all topologies,

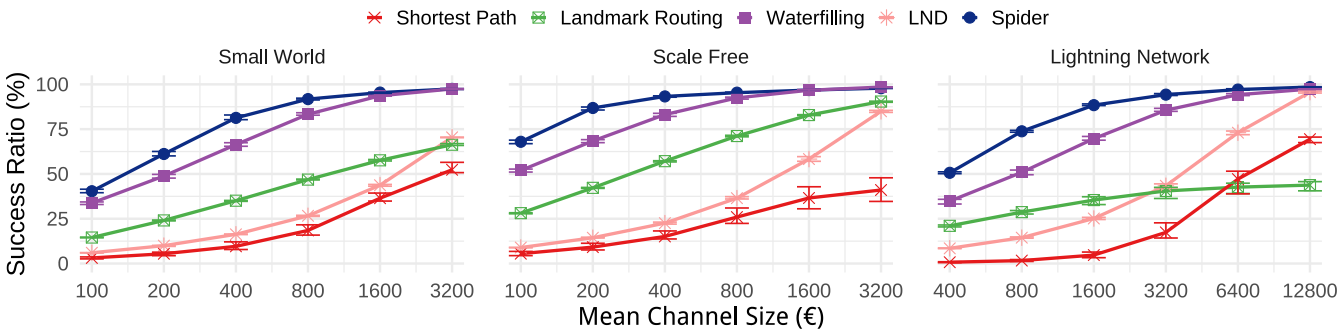


Figure 22: Performance of different algorithms on different topologies with equal channel sizes with different per sender transaction arrival rates. Spider consistently outperforms all other schems achieving near 100% average success ratio. Error-bars denote the maximum and minimum success ratio across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

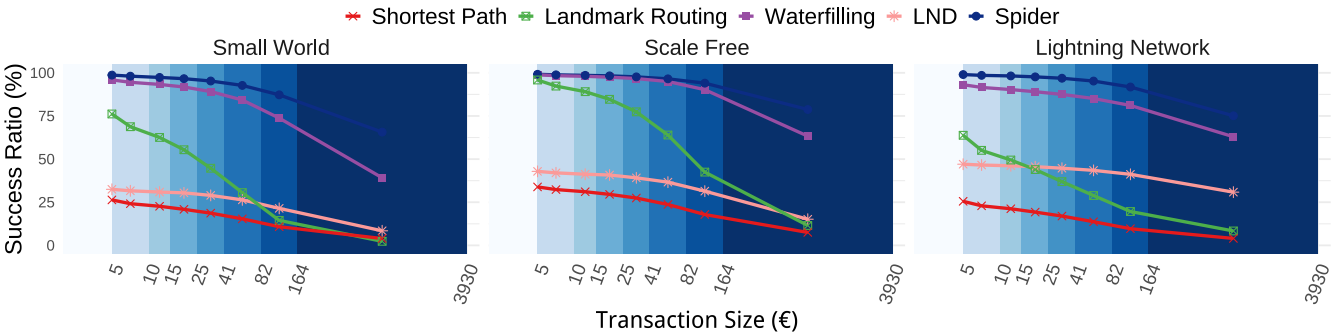


Figure 23: Breakdown of performance of different schemes by size of transactions completed. Each point reports the success ratio for transactions whose size belongs to the interval denoted by the shaded region. Each interval corresponds roughly to 12.5% of the CDF denoted in Fig. 7a. The graphs correspond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

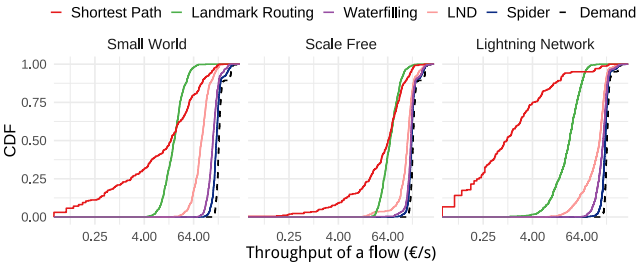


Figure 24: CDF of normalized throughput achieved by different flows under different schemes across topologies. Spider achieves close to 100% throughput given its proximity to the black demand line. Spider is more vertical line than LND because it is fairer: it doesn't hurt the throughput of smaller flows to attain good overall throughput.

Spider outperforms the state-of-the-art schemes on success ratio. 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.

C.3 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) to understand the fairness of the scheme to different pairs of nodes transacting on the PCN. Fig. 24 shows a CDF of the absolute throughput in €/s achieved by different protocols on a single circulation demand matrix when each sender sends an average of 30 tx/s. The mean channel sizes for the synthetic topologies and the real topologies with LCSD channel sizes are 4000€ and 16880€ respectively. We run each protocol for 1010s and measure the success volume for transactions arriving between 800-1000s. We make two observations: (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 LND.

C.4 DAG Workload on Synthetic Topologies

Fig. 25 shows the effect of adding a DAG component to the transaction demand matrix on the synthetic small world and scale free topologies. We observe the success ratio and

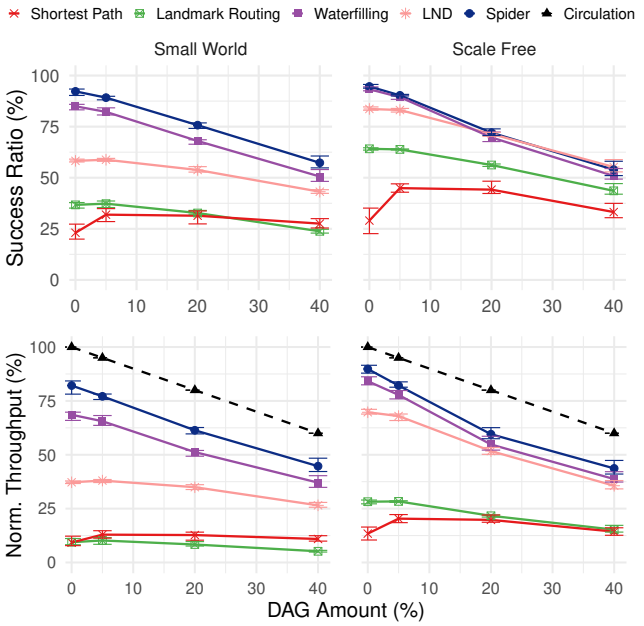


Figure 25: Performance of different algorithms across all topologies as the DAG component in the transaction demand matrix is varied. As the DAG amount is increased, the normalized throughput achieved is further away from the expected optimal circulation throughput. The gap is more pronounced on the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

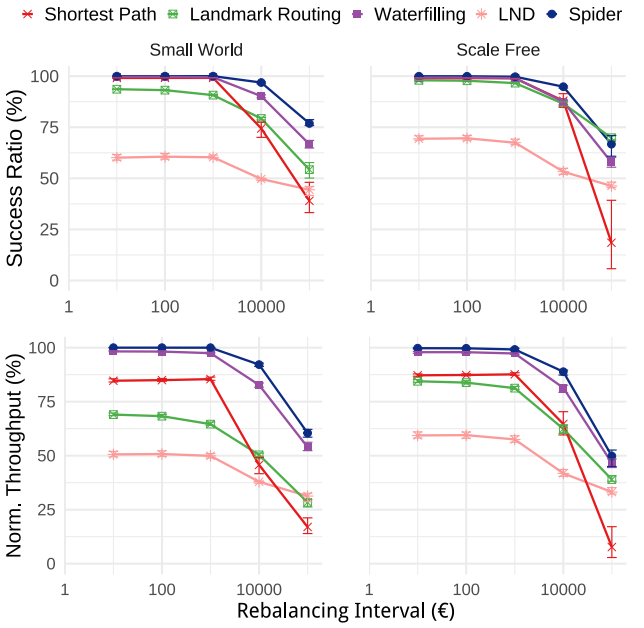


Figure 26: Performance of different algorithms across all topologies as the DAG component in the transaction demand matrix is varied. As the DAG amount is increased, the normalized throughput achieved is further away from the expected optimal circulation throughput. The gap is more pronounced on the real topology.

normalized throughput of different schemes with five different traffic matrices with 30 transactions per second per sender under 5%, 20%, 40% DAG components respectively. 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 suggests again 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. 26 shows the success ratio and normalized throughput achieved by different schemes when rebalancing is enabled for the 20% DAG traffic demand from Fig. 25. 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.