Routing Cryptocurrency with the Spider Network

Vibhaalakshmi Sivaraman*, Shaileshh Bojja Venkatakrishnan*, Mohammad Alizadeh*,
Giulia Fanti†, Pramod Viswanath ‡
*MIT CSAIL, †CMU, ‡UIUC
spider.network@mit.edu

Abstract

With the growing usage of Bitcoin and other cryptocurrencies, many scalability challenges have emerged. A promising scaling solution, exemplified by the Lightning Network, uses a network of bidirectional payment channels that allows fast transactions between two parties. However, routing payments on these networks efficiently is non-trivial, since payments require finding paths with sufficient funds, and channels can become unidirectional over time blocking further transactions through them. Today's payment channel networks exacerbate these problems by attempting to deliver all payments atomically.

We present the Spider network, a new packet-switched architecture for payment channel networks that addresses these challenges. Spider splits payments into transaction units and transmits them over a period of time across different paths. Spider uses congestion control, in-network scheduling, and imbalance-aware routing to optimize delivery of payments. Our results show that Spider improves the number and volume of successful payments on the network by 10-75% and 10-35% respectively compared to practical state-of-the-art approaches.

1 Introduction

Today's cryptocurrencies have poor transaction throughput and slow confirmation times. Bitcoin supports 3-7 transactions per second and takes tens of minutes to confirm a transaction [35]. By comparison, established payment systems like Visa process thousands of transactions per second with a delay of a few seconds [35]. Further, high transaction costs make current blockchains impractical for micropayments [2].

Payment channel networks are a leading proposal for tackling these scalability challenges. A payment channel is a blockchain transaction that escrows a given user Alice's money in order to enable future transactions to a specific recipient Bob, much like a gift card. Once Alice opens a payment channel to Bob, she can transfer funds repeatedly and securely without recording every transaction on the blockchain. By routing payments through intermediate payment channels, participants in a payment channel network can transfer funds even if they do not share a direct payment channel. First proposed in the Lightning Network [27], payment channel networks have been touted as a game-changing technology [17, 23, 32], with multiple implementations under development (e.g., Bitcoin's Lightning Network [27], Ethereum's Raiden Network [8]).

Payment channel networks transfer cryptocurrency, not data, but their design presents several technical and economical challenges familiar to communication networks. First, payment channel networks require efficient mechanisms to find paths for payments and to deliver them with high throughput and low delay. Efficient networking is essential to the economic viability of payment channel networks; it is particularly important to achieve a high transaction throughput without having to escrow a large amount of capital in payment channels. Second, the network must provide the right incentives to both end-users, who desire low transaction fees, and service providers, who wish to maximize their profits from routing payments. Third, the network should ensure the privacy of user transactions.

In this paper we present *Spider*, a new design for payment channel networks. Spider uses two main ideas that distinguish it from existing approaches. First, it uses packet switching. Existing designs attempt to send payments atomically on paths with enough funds to *fully* satisfy the payment — an approach analogous to circuit switching. By contrast, Spider's senders break up payments into *transaction units* and transmit them over a period of time across different network paths. Spider uses congestion control and in-network scheduling of transaction units to achieve high utilization of payment channel funds while supporting a variety of payment delivery services (e.g., atomic and non-atomic payments with different deadlines).

Spider's second key idea is *imbalance-aware* routing. An important challenge for routing is that a payment channel becomes imbalanced when the transaction rate across it is higher in one direction than the other; the party making more payments eventually runs out of funds and cannot send further payments without depositing new funds into the payment channel on the blockchain. We analyze routing with rate-imbalance constraints from first principles and show that the maximum achievable throughput depends on properties of a *payment graph* that captures the flow of currency between network participants. Inspired by prior work on optimization-based routing and rate control [13, 20], we formulate optimization problems for routing with rate-imbalance constraints, and we derive decentralized algorithms for solving these problems as well as simpler heuristic algorithms.

Our preliminary results show that Spider improves the number and volume of successful payments through the payment channel network, compared to prior approaches to routing

HotNets-XVII, November 15–16, 2018, Redmond, WA, USA

ACM ISBN 978-1-4503-6120-0/18/11…$15.00
https://doi.org/10.1145/3286062.3286067

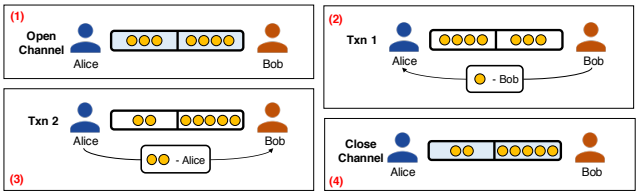


Figure 1: Bidirectional payment channel between Alice and Bob. A blue shaded block indicates a transaction that is committed to the blockchain.

payments. For a given amount of funds escrowed in the network, Spider is able to complete 10-30% more transactions amounting to a 10-15% increase in volume of transactions relative to SilentWhispers [22]. It also completes 55-75% more transactions (25-35% increase in transaction volume) than SpeedyMurmurs [30]. On an ISP-like topology, Spider outperforms a classical max-flow based-approach by 5-15% on both the number and volume of successful transactions.

2 Background

Bidirectional payment channels are the building blocks of a payment channel network. A bidirectional payment channel allows a sender (Alice) to send funds to a receiver (Bob) and vice versa. To open a payment channel, Alice and Bob jointly create a transaction that escrows money for a fixed amount of time [27]. 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 [27].

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 hashlock ensures that all intermediate transactions are only valid after a transaction recipient knows a private key generated by Alice [8]. Once Alice is ready to pay, she gives that key to Bob; he can either broadcast it (if he decides to close the channel) or pass it to Charlie. Charlie is incentivized to relay the key upstream to Alice so that he can also get paid.

3 Related Work

An important problem is how to choose routes for transactions. In the Lightning Network, each node maintains a local view of

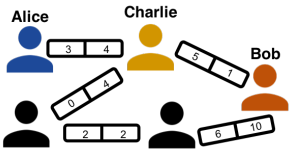


Figure 2: In a payment channel network, Alice can transfer money to Bob by using intermediate nodes' channels as relays. There are two paths from Alice to Bob, but only the path (Alice, Charlie, Bob) can support 3 tokens.

the network topology and source-routes transactions [27]; in current implementations, nodes pick paths using shortest path algorithms [5]. An important benchmark is the *max-flow* routing algorithm [14], which uses a distributed Ford-Fulkerson algorithm to find source-destination paths that support the largest transaction volume for each transaction. If this volume exceeds the transaction value, the transaction succeeds. Max-flow routing is the gold standard in terms of throughput and transaction success rate, but it has high overhead, requiring $O(|V| \cdot |E|^2)$ computation per transaction, where $|V|$ and $|E|$ are the number of nodes and edges in the network, respectively [36]. Currently, the Lightning Network has $\sim$1000 nodes and 10,000 channels, making this very expensive [3, 6].

Two main alternatives have been proposed: landmark routing and embedding-based routing. In *landmark routing*, select routers (landmarks) store routing tables for the rest of the network, and nodes need only route transactions to a landmark [33]. This approach is used in Flare [28] and SilentWhispers [22, 24]. Spider does not use landmarks, but like SilentWhispers, it splits transactions over multiple paths [22]. *Embedding-based* or *distance-based* routing instead 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 [29] and SpeedyMurmurs [30] use embedding-based routing. Computing and updating the embedding dynamically as the topology and link balances change is a primary challenge of these approaches.

A key difference with prior work is that Spider actively accounts for the cost of channel imbalance by preferring routes that rebalance channels. The problem of channel imbalance is receiving increasing attention [21], but prior literature treats rebalancing as a separate, periodic task. Revive [21], for instance, periodically chooses leaders to compute rebalancing transactions for a group of online participants. Spider instead explicitly incorporates it into its routing algorithms.

4 Architecture

In current payment channel networks, the sender first finds one or more paths with enough funds ("capacity") to fully satisfy the payment, and only then transmits it by sending one transaction on each path. This approach is similar to circuit switching and has several drawbacks. First, it makes it difficult to support large payments. Second, it exacerbates imbalance on payment channels. A large transaction can deplete funds on one side of

a payment channel; the party that runs out of funds cannot send more payments until it either receives payments from the other side, or it replenishes funds via a blockchain transaction. The result is *head of line blocking*, where large transactions can block shorter payments that could have been serviced quickly.

Spider is a packet-switched payment channel network that solves these problems. At a high-level, Spider hosts send payments over the network by transmitting a series of *transaction units* over time, much like packets in a data network. Each transaction unit transfers an amount of money bounded by the *maximum transaction unit (MTU)*. Transaction units from different payments are queued at Spider routers, which transmit them as funds become available in payment channels.

4.1 Spider Hosts

Spider hosts run a transport layer that provides standard interfaces for applications to send and receive payments on the network. We envision a message-oriented transport rather than a stream-oriented transport like TCP. To send a payment, the application specifies the destination address, the amount to send, a deadline, and the maximum acceptable routing fee.

The transport provides interfaces for both atomic and non-atomic payments. It guarantees that atomic payments are either fully delivered or that no payment takes place. For non-atomic payments, the transport is allowed to partially deliver the payment; it must then inform the sender precisely how much it delivered by the deadline and ensure that no further transactions are made as part of that payment. The sender can attempt the remainder of the payment at a later time, or decide to complete the payment on the blockchain. As we will see, relaxing atomicity improves network efficiency, and we therefore expect the routing cost for non-atomic payments to be cheaper.

Recall that transactions through payment channel networks are locked by a cryptographic hashlock, whose private key is known only to the sender (§2). To implement non-atomic payments, the sender simply waits for confirmation from the receiver that she has received a transaction unit (identified by a payment ID and sequence number), and only then sends her the key. The sender therefore knows exactly how much of a payment the receiver can claim. It can withhold the key for in-flight transactions that arrive after the deadline, or cancel them by informing routers.

Spider is also compatible with atomic payments using recent mechanisms like Atomic Multi-Path Payments (AMP) [1]. AMP splits a payment over multiple paths and derives the keys for all payment transaction units from a single "base key"; by using secret sharing, it ensures that the receiver cannot unlock any money before receiving all transaction units.

Spider hosts use a *congestion control* algorithm to determine the rate to send transaction units for different payments. Designing congestion control algorithms for payment channel networks is beyond the scope of this paper, but we briefly remark on some interesting aspects. Standard goals for congestion control in data networks such as high utilization, fairness,

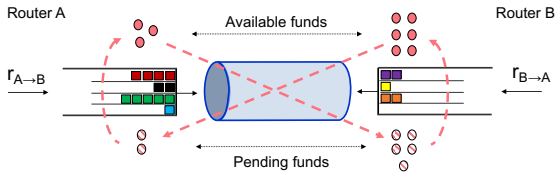


Figure 3: Routers queue transaction units and schedule them across the payment channel based on available capacity and transaction priorities. Funds received on a payment channel remain in a pending state until the final receiver provides the key for the hashlock.

and low delay also apply to payment channel networks. Additionally, transfers in payment channel networks have deadlines, and therefore approaches that adapt congestion control to meet deadlines are particularly relevant [18, 34]. Hosts can implement congestion control through implicit signals like delay or explicit signals from the routers (e.g., queue sizes, available capacity, imbalance, etc.).

A unique aspect of payment channel networks is that sending at higher rates does not always reduce capacity for other payments; it may improve performance for other payments by rebalancing payment channels. For example, a sender that sees imbalanced payment channels in the downstream direction can aggressively increase its rate to rebalance those channels. Exploring imbalance-aware congestion control algorithms is an interesting direction for future work.

4.2 Spider Routers

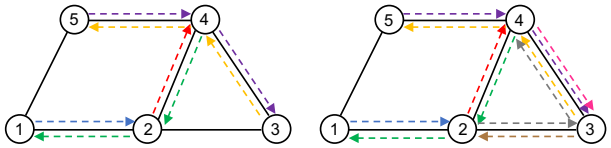
Spider routers are responsible for forwarding transaction units to the intended receiver. Existing designs like the Lightning Network use Onion routing [15] to ensure privacy of user payments. Spider routers can use similar mechanisms for each transaction unit to provide privacy [7].

A Spider router queues transaction units when it lacks the funds to send them immediately (Fig. 3). As it receives funds from the other side of the payment channel, it uses them to send new transaction units from its queue. Funds from newly-arrived transaction units are not available to use immediately. The router must wait until it receives the key for the hashlock from the final destination. This delay limits the capacity of a payment channel. If a transaction takes on average Δ seconds to confirm, then a payment channel with total funds c can support, on average, transactions of net value no more than c/Δ currency units per second.

Queuing of transaction units at Spider routers could result in increased delays for some payments. However, the routers can offer different classes of service by scheduling transaction units based on payment requirements, such as prioritizing payments based on size, deadline, or routing fees [12].

5 Routing

Motivating example. The key to effective routing in payment channel networks is to keep payment channels *balanced*. To illustrate the importance of balanced routing, consider the



(a) Shortest-path routing (b) Optimal routing

Figure 4: Example illustrating balanced routing by two different schemes. Each dotted edge represents 1 unit of flow along the direction of the arrow. The colors indicate separate flows.

5-node payment-channel network shown in Fig. 4. Suppose sender, receiver pairs seek to transact at the rates shown in Fig. 5a. For example, node 1 wishes to send at rate 1 to nodes 2 and 5, and node 2 wishes to send at rate 2 to node 4. Fig. 4 shows two different routing strategies under these demands. Notice that in both cases, the net rate in the two directions of every edge is equal. This *balance* requirement is necessary for any routing strategy to ensure that payment channels do not run out of funds in one direction.

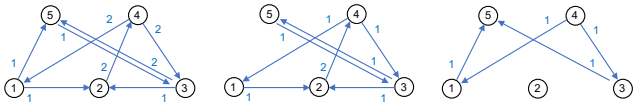
Fig. 4a shows the result for shortest-path balanced routing, wherein senders route only along the shortest path to their receiver nodes. For example, node 4 routes a flow of rate 1 along the path $4 \rightarrow 2 \rightarrow 1$ (shown in green). The maximum total rate (*throughput*) that can be sent by this routing scheme is 5 units; any rate assignment offering a higher throughput does not satisfy the balance constraints. However, an alternate routing scheme of Fig. 4b in which senders do not necessarily send along the shortest paths achieves a higher throughput of 8 units. Here, node 2 sends a flow of rate 1 along the path $2 \rightarrow 3 \rightarrow 4$, while the shortest path is $2 \rightarrow 4$. This enables nodes 3 and 4 to also send 1 unit of flow to nodes 2 and 3 respectively.

Our goal is to design a decentralized routing algorithm to maximize throughput while maintaining channel balance. We first describe a fluid model of the network (§5.1), wherein transactions between source, destination pairs are modeled as continuous flows. The model provides insights on the fundamental limits on throughput that can be achieved with balance constraints (§5.2) and motivates our algorithms (§5.3).

5.1 Fluid Model

Consider the payment channel network modeled as a graph $G(V,E)$, with routers V and payment channels E. For any source, destination routers $i,j \in V$, let $d_{i,j} \geq 0$ denote the average rate at which transaction units have to be transferred from i to j. Let c_e denote the total amount of funds in channel e, for $e \in E$, and Δ the average latency experienced by transaction units due to network delays. Lastly let $\mathcal{P}_{i,j}$ denote the set of paths from i to j in G, for $i,j \in V$. We include only 'trails', i.e., paths without repeated edges, in $\mathcal{P}_{i,j}$.

In the fluid model, the demands $d_{i,j}$ are satisfied by sending *flows* from source to destination routers. A flow is defined by a (path, value) tuple, where path denotes the route taken and value denotes the rate carried by the flow. Operationally this can be interpreted as routing transaction units along the flow path such that the average rate of currency transferred along the path equals the value of the flow. Thus, in the fluid model, maximizing throughput is equivalent to finding flows of maximum total value and can be formulated as a Linear Program (LP):



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

Figure 5: Payment graph corresponding to the demands in the example of Fig. 4. It decomposes into a maximum circulation and DAG components as shown in (b) and (c).

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

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

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

$$\sum_{p \in \mathcal{P}:(u,v) \in p} x_p - \sum_{p \in \mathcal{P}:(v,u) \in p} x_p = 0 \quad \forall (u,v) \in E$$

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

Here, x_p denotes the flow along path p and $\mathcal{P} = \cup_{i,j \in V} \mathcal{P}_{i,j}$ is the set of all paths. The second set of constraints reflect the capacity limits of payment channels (§4.2) and the third set of constraints enforce the balance requirement. We emphasize that the balance constraints are fundamentally necessary (much like capacity and demand constraints) for the flows computed to be actually realizable.

5.2 Limits on Throughput

Next, we show that the maximum throughput achievable is fundamentally restricted by the structure of demand $[d]_{i,j}$ across users, because of the link balancing requirements.

Payment graphs and circulation. Define a *payment graph* $H(V,E_H)$ as a weighted directed graph with nodes V and edges E_H. An edge $(i,j) \in E_H$ if $d_{i,j} > 0$ with $d_{i,j}$ also being the weight of that edge. Payment graphs are useful for analyzing throughput as any flow imbalance across cuts of H cannot be balanced by any routing scheme in G. Fig. 5a shows an example of a payment graph corresponding to the example discussed at the beginning of §5.

We define the *circulation* graph of a payment graph H as another directed weighted graph $C(V, E_C)$ with $E_C \subseteq E_H$, $w_C(i,j) \leq w_H(i,j)$ for all $(i,j) \in E_C$, with the total weight of incoming and outgoing edges equal at any node. $w_C(i,j)$, $w_H(i,j)$ denote edge weight of edge (i,j) in graphs C, H respectively. The circulation graph captures flows that form cycles in the payment graph. Letting $\sum_{(i,j) \in E_C} w_C(i,j)$ be the total *value* $v(C)$ of the circulation, there exists a circulation graph C^* (not necessarily unique) of maximum value for a given payment graph H. A maximum circulation graph C^* can be constructed by successively removing cycles of constant flow from H, and adding them to C^*. The graph remaining after removing all

cycles from H is a weighted directed acyclic graph (DAG). Fig. 5b and 5c show the decomposition of the payment graph of Fig. 5a into circulation and DAG components.

PROPOSITION 1. *For a payment graph H with a maximum circulation graph C^*, there exists a routing scheme that achieves a throughput of $\nu(C^*)$ on a network with payment channels of unlimited capacity. Conversely, no routing scheme can achieve a throughput greater than $\nu(C^*)$ on any network.*

We refer to an extended version of this paper [10] for the proof. Thus, even if the network is not capacity-limited, the maximum achievable throughput can be less than 100% if the payment graph is not a circulation. The routing presented in Fig. 4b corresponds precisely to routing the maximum circulation component of the payment graph (Fig. 5b) and is hence optimal. Yet it is only able to route $8/12 = 75\%$ of the demands in the payment graph. To route demands beyond the circulation, some routers have to incrementally deposit funds and replenish their channel balances on the blockchain. The question of which routers should perform these deposits (and at what rate) can be answered by generalizing the LP in Eq. (1). We refer the reader to the extended version [10] for details.

5.3 Algorithms

Directly solving the LP in Eq. (1) would require a centralized entity to estimate the average payment demands and make routing decisions for the network as a whole. Fortunately, the structure of the LP and its dual naturally motivate a decentralized "Primal-Dual" [26] algorithm. Unlike the LP, the decentralized algorithm does not require explicit payment demands; it routes instantaneous payments demands based on real-time congestion and channel imbalance information.

Each payment channel has a *price* in each direction. Routers locally update these prices based on both congestion and imbalance in their payment channels. End-host senders compute prices of different paths whenever there is a transaction to be completed, and send along the best route. The use of dual variables for prices is common in the utility-maximization-based rate control and routing literature (e.g., see [20]). A key difference from prior work is that in addition to price variables for link capacity constraints, we also have price variables for link balance constraints. This ensures that links with skewed balances have a high price and effectively rebalances the network. Due to space constraints, we include more details in an extended version of this paper [10].

The Primal-Dual algorithm will find the optimum routes for the network but may take many iterations to converge [16, 25]. Therefore we also propose a simple multi-path load balancing algorithm inspired by similar algorithms in networking [19, 37]. In this approach, sources independently try to minimize imbalance on their paths by always sending on paths with the largest available capacity, much like "waterfilling" algorithms for max-min fairness. A source measures the available capacity on a set of paths to the destination. It then transmits on the path with highest capacity until its capacity is the same as the second-highest-capacity path; then it transmits on both of these paths until they reach the capacity of the third highest-capacity-path, and so on.

We expect Waterfilling to converge faster since routes directly react to channel balances, unlike the Primal-Dual algorithm where routes react to channel prices, which are themselves slowly reacting to actual available capacities. However, the eventual equilibrium routes computed by Waterfilling could have a total throughput that is sub-optimal, because each source load balances its own paths greedily [31].

6 Preliminary Evaluation

6.1 Setup

Simulator. We modified an existing simulator for payment channel networks [11] to model transaction arrivals and completion events. Incoming transactions are routed according to the routing algorithm if funds are available on the desired paths. Successful payments incur a delay of 0.5 seconds before the funds are available at the receiver. In the meantime, these funds are held *inflight* and are unavailable for use by any party along the path. As soon as a transaction completes, these funds are released. The simulator supports non-atomic payments through a global queue of pending payments. The queue is periodically polled to see if transactions can progress further. They are then scheduled according to a scheduling algorithm. We leave implementing in-network queues and rate control to future work.
Dataset. We evaluated the algorithms on two different topologies: an ISP-topology [4] and a subgraph from the original topology of Ripple [9], an existing currency exchange network that transacts in XRPs. The ISP topology has 32 nodes and 152 edges. For the ISP topology, we generated 200,000 transactions, with sizes sampled from the Ripple data after removing the largest 10% of transactions. The average transaction size for this dataset was 170 XRP with a maximum size of 1780 XRP. The sender for each transaction was sampled from an exponential distribution of nodes while the receiver was sampled uniformly at random. All graph edges were given the same capacity, a number in the range 10000 XRP to 100000 XRP per link across different experiments.

We also used data from the Ripple network from January 2013 [11]. The original dataset had 90,000 nodes and 330,000 edges. We pruned it to remove the degree-1 nodes (which don't make routing decisions) as well as edges with no funds between them. The largest resulting component had 3774 nodes and 12512 edges. The 75,000 transactions from the original dataset that are between nodes in this subgraph had an average size of 345 XRP a maximum size of 2892 XRP. Consequently, we set the capacity of links in the reduced Ripple graph to 30000.
Schemes. We evaluated SpeedyMurmurs [30], SilentWhispers [22], and max-flow routing. All of these schemes use atomic payments. We also implemented shortest-path routing with non-atomic payments as another baseline for our packet-switched network. We compared these schemes to Spider (LP) and Spider (Waterfilling). Spider (LP) is a centralized scheme; it solves the LP in Eq. (1) once using the total traffic

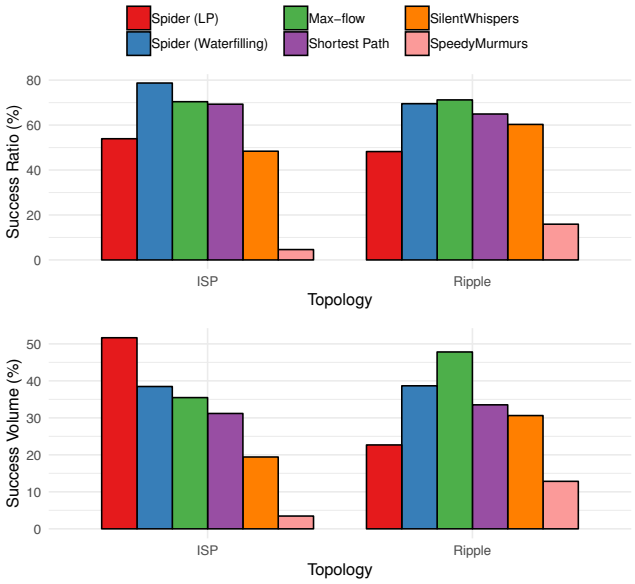


Figure 6: Comparison of payments completed across schemes on the ISP and Ripple Topologies when the capacity per link is 30,000

demand as input, and it uses the resulting flows for the entirety of the experiment. We use Spider (LP) as a proxy for the decentralized primal-dual algorithm, whose evaluation we leave to future work. Spider (Waterfilling) is the decentralized algorithm described in §5.3. We restrict both algorithms to use 4 disjoint shortest paths for every source-destination pair. All non-atomic payments are scheduled in order of increasing incomplete payment amount, i.e. according to the *shortest remaining processing time (SRPT)* policy [12].

Metrics. We evaluate these routing schemes for their *success ratio* and *success volume*. The former captures how many payments amongst those tried actually completed. The latter focuses on the volume of payments that went through as a fraction of the total volume across all attempted payments.

6.2 Results

We summarize our results in Fig. 6. The results were collected after 200*s* for the ISP topology and 85*s* for the Ripple topology. All the channels in both topologies were initialized with a capacity of 30000 XRP, equally split between the two parties. We can see that splitting the payments into transaction units and scheduling them according to SRPT already provides a 10% increase in success ratio over SilentWhispers and SpeedyMurmurs even for the shortest path routing scheme. Although Max-flow performs quite well, it has a high overhead per transaction as discussed in §3. In comparison, Spider (Waterfilling) is able to leverage knowledge of imbalance to perform within 5% of Max-flow despite being restricted to only 4 paths.

Spider (LP), on the other hand, attains a success volume of 52% and 22% for the ISP and Ripple topologies respectively. Both of these correspond precisely to the circulation component of the payment graph. This is because Spider (LP) uses

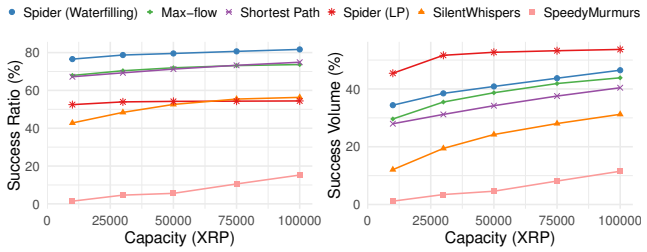


Figure 7: Effect of increasing capacity per link on the success metrics when routing payments on the ISP topology. All links in the network have the same credit.

an estimate of the demand matrix to make decisions for the entire duration of the simulation. While this approach works for stationary transaction arrival patterns (like the ISP topology), it does not work well for the Ripple network, where traffic demands vary over time. Further, the LP assigns zero flows to all paths for certain commodities, so no payments between them will ever get attempted. We plan to explore objectives like proportional fairness [20] in the future to overcome this problem.

How does capacity impact success? We varied the capacity on each link in the ISP topology from 10000 XRP to 100000 XRP and measured the success across the schemes. Fig. 7 summarizes the results. As expected, as the capacity increases, more transactions start succeeding. The total volume of successful transactions also experiences an increase. Additionally, to achieve a certain success volume or success ratio, the amount of capital that needs to be locked in with Spider (Waterfilling) is much lower than what would need to be invested in any other scheme. Spider (LP) is less sensitive to changes in capacity, because it does a better job of avoiding imbalance.

7 Discussion and Future Work

Payment channel networks promise to be an important ingredient for scaling future blockchain systems, and their design presents many exciting and unique networking challenges.

As a first step, this paper proposes a new packet-switched architecture for payment channel networks that harnesses in-network scheduling and imbalance-aware routing. We envision further potential gains from router and end-host collaboration, such as rejecting large transactions that are unlikely to complete within the deadline to increase throughput.

Orthogonal to this, our routing algorithms set routing fees to maximize throughput for rational users that prefer cheaper routes. However, our design does not address incentives for network service providers that wish to maximize profits from routing fees or adversaries that tamper with the reported fees. Analyzing Spider's robustness against such adversaries is an important direction for future work.

Lastly, we have focused on optimizing payment delivery for a given network and payment channel capacities. Our results show fundamental limits to performance that depend on network topology and payment patterns. Designing topologies that admit efficient routing in a decentralized fashion is an interesting area for investigation.

Acknowledgments

We thank Andrew Miller and Thaddeus Dryja for their feedback, and the Distributed Technologies Research Foundation, the Cisco Research Center, the National Science Foundation grants CNS-1718270, CNS-1563826 and CNS-1617702, and the Army Research Office under grant W911NF1810332 for their support.

References

[1] AMP: Atomic Multi-Path Payments over Lightning. https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html.

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

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

[4] ISP Topology Zoo. http://www.topology-zoo.org/.

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

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

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

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

[9] Ripplenet. https://ripple.com/.

[10] Routing Cryptocurrency with the Spider Network. https://arxiv.org/abs/1809.05088.

[11] SpeedyMurmurs Software. https://crysp.uwaterloo.ca/software/speedymurmurs/.

[12] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 435–446, New York, NY, USA, 2013. ACM.

[13] 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.

[14] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956.

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

[16] M. Gurbuzbalaban, A. Ozdaglar, and P. A. Parrilo. On the convergence rate of incremental aggregated gradient algorithms. *SIAM Journal on Optimization*, 27(2):1035–1048, 2017.

[17] A. Hertig. Lightning: The Bitcoin Scaling Tech You Really Should Know. December 2017.

[18] C.-Y. Hong, M. Caesar, and P. Godfrey. Finishing flows quickly with preemptive scheduling. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 127–138. ACM, 2012.

[19] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the tightrope: Responsive yet stable traffic engineering. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 253–264. ACM, 2005.

[20] 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.

[21] 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.

[22] 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.

[23] J. Medley. Bitcoin Lightning Network: Scaling Cryptocurrencies for Mainstream Use. July 2018.

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

[25] A. Nedic and A. Ozdaglar. Subgradient methods in network resource allocation: Rate analysis. In *Information Sciences and Systems, 2008. CISS 2008. 42nd Annual Conference on*, pages 1189–1194. IEEE, 2008.

[26] 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.

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

[28] P. Prihodko, S. Zhigulin, M. Sahno, A. Ostrovskiy, and O. Osuntokun. Flare: An approach to routing in lightning network. *White Paper (bitfury. com/content/5-white-papers-research/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016. pdf)*, 2016.

[29] 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.

[30] 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.

[31] T. Roughgarden and E. Tardos. How bad is selfish routing? *J. ACM*, 49(2):236–259, Mar. 2002.

[32] K. Torpey. Greg Maxwell: Lightning Network Better Than Sidechains for Scaling Bitcoin, 2016. https://bitcoinmagazine.com/articles/greg-maxwell-lightning-network-better-than-sidechains-for-scaling-bitcoin-1461077424/.

[33] 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.

[34] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 115–126, New York, NY, USA, 2012. ACM.

[35] J. Vermeulen. Bitcoin and Ethereum vs Visa and PayPal - Transactions per second. April 2017.

[36] H. S. Wilf. *Algorithms and complexity*. AK Peters/CRC Press, 2002.

[37] 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.