Routing Cryptocurrency with the Spider Network

Vibhaalakshmi Sivaraman[1], Shaileshh Bojja Venkatakrishnan[1], Mohammad Alizadeh[1], Giulia Fanti[2], and Pramod Viswanath[3]

[1]Massachusetts Institute of Technology
[2]Carnegie Mellon University
[3]University of Illinois at Urbana-Champaign

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. In this paper, we present the Spider network, a new packet-switched architecture for payment channel networks. Spider splits payments into transaction units and transmits them over time across different paths. Spider uses congestion control, payment scheduling, and imbalance-aware routing to optimize delivery of payments. Our results show that Spider improves the volume and number of successful payments on the network by 10-45% and 5-40% respectively compared to 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 [29]. By comparison, established payment systems like Visa process thousands of transactions per second with a delay of a few seconds [29]. Further, high transaction costs make current blockchains impractical for micropayments. The median Bitcoin transaction fee regularly exceeds $1 and reached $34 in December 2017 [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

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

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 [22], payment channel networks have been touted as a game-changing technology in the cryptocurrency community [13, 26, 19]. Multiple implementations are under development (e.g., Bitcoin's Lightning Network [22], Ethereum's Raiden Network [5]). In July 2018, a pilot program started to allow over 100 merchants to accept payments over the Lightning Network [9].

Payment channel networks transfer cryptocurrency, not data, but their design presents technical and economical challenges familiar to communication networks. First, payment channel networks require efficient mechanisms to find paths with enough capacity for payments and to deliver them with high throughput and low delay. Efficient networking is essential to the economic viability of payment channel networks. Since funds deposited into payment channels cannot be used for other economic activities, it is particularly important to achieve a high transaction throughput with a small amount of capital locked in the network. Second, the network design must provide the right economic incentives to both end-users and service providers that route payments. Third, the network should ensure the privacy of user transactions.

In this paper, we take steps towards addressing some of these challenges and present the Spider network. Spider is a packet-switched payment channel network: it breaks up payments into *transaction units* and transmits them across different network paths over a period of time. Spider uses congestion control and in-network scheduling mechanisms to achieve high utilization and best-effort payment delivery within a deadline. In comparison to existing designs that attempt to send transactions atomically, we show that Spider's use of packet switching principles improves transaction success rate and throughput.

A key facet of Spider that differentiates it from standard networks 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 on-chain. We take a first-principles approach to routing with rate-imbalance constraints, and show that the maximum achievable throughput depends on properties of a *payment graph* that captures how currency flows between network participants. We formulate optimization problems for routing with rate-imbalance constraints and derive decentralized algorithms for solving these problems as well as simpler heuristic algorithms. Our work is inspired by prior work on optimization-based routing and rate control in communication networks [15, 10] and multipath transport protocols like MPTCP [31].

Our preliminary results show that Spider improves the number and volume of successful payments through the payment channel network. For a given amount of funds locked into the network, Spider is able to complete 10-75% more transactions amounting to a 10-45% increase in volume of transactions relative to SpeedyMurmurs [25] and SilentWhispers [18], two prior approaches to the path-discovery problem for payment channel networks. On an ISP-like topology, Spider also outperforms a classical max-flow based-approach by 5-15% on both the number and volume of successful transactions.

In summary, payment channel networks promise to be a key ingredient for scaling future blockchain systems. Their design presents exciting intellectual and engineering challenges, and we hope that this paper inspires more work in the networking community in this area.

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 [22]. Suppose Alice puts 3 units in the channel, and Bob puts four (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 [22].

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 [5]. Once Alice is ready to pay, she gives that key to Bob; he can either broadcast it (if he decides to close the channel) or pass it to Charlie. Charlie is incentivized to relay the key upstream to Alice so that he can also get paid. Note that the underlying cryptography backing payment channels assumes that transactions on the payment channels are larger than the blockchain transaction fee to ensure that broadcasting the true balance on a channel is profitable.

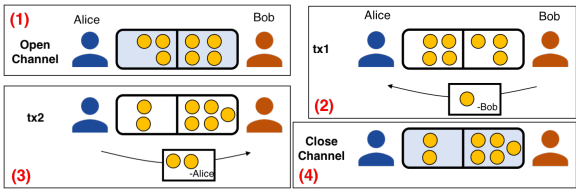


Figure 1: Bidirectional payment channel between Alice and Bob. A blue shaded block indicates a transaction that is committed to the blockchain.

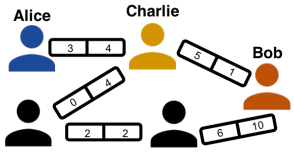


Figure 2: In a payment channel network, Alice can transfer money to Bob by using intermediate nodes' channels as relays. There are two paths from Alice to Bob, but only the path (Alice, Charlie, Bob) can support 3 tokens.

3 Related Work

An important problem is how to choose routes for transactions. In the Lightning Network, each node keeps a routing table for the rest of the network and source-routes transactions [22]. A key benchmark is the *max-flow* routing algorithm [11]. For each transaction, max-flow uses a distributed implementation of the Ford-Fulkerson method to find source-destination paths that support the largest transaction volume. If this volume exceeds the transaction value, the transaction succeeds. Max-flow routing has been 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 [30].

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; hence individual nodes only need to route transactions to a landmark [27]. This approach is used in systems like Flare [23] and SilentWhispers [18, 20]. Spider does not use landmarks, but like SilentWhispers, it splits transactions over multiple paths [18]. *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. Systems like VOUTE [24] and SpeedyMurmurs [25] use embedding-based routing. One challenge is computing and updating the embedding as the graph and link balances change over time.

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 [17], but prior literature treats rebalancing as a separate, periodic task; 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 transaction on the blockchain. 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*, analogous to 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.

Non-atomic payments. Recall that transactions are locked by a cryptographic hash lock, whose private key is known only to the sender (§2). This key can alternatively be generated by the recepient [22] leading to a slightly different protocol. In Spider, the sender generates a new key for every transaction unit since this gives her better control over transaction completions. For instance, 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 unlock. It can withhold the key for "in-flight" transactions that arrive after the deadline, or proactively cancel them by informing the routers.

Atomic payments. Spider is also compatible with atomic payments using recently-proposed mechanisms like Atomic Multi-Path Payments (AMP) [1] that split a payment over multiple paths while guaranteeing atomicity. The idea is to derive the keys for all the transaction units of a payment from a single "base key," and use additive secret sharing to ensure that the receiver cannot unlock any of the transaction units until she has received all of them.

Congestion control. 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, 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 [28, 14]. To make congestion control decisions, hosts can use 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, but may in fact improve performance for other payments by promoting balance across payment channels. For example, if a sender discovers that payment channels on certain paths have a high imbalance in the downstream direction, it may aggressively increase its rate to balance those channels. Exploring imbalance-aware congestion control algorithms is an interesting direction for future work.

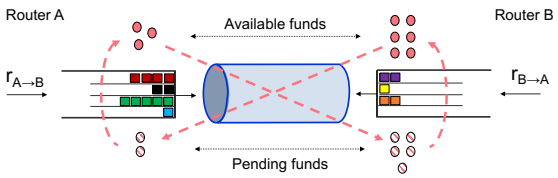


Figure 3: Routers queue transaction units and schedule them across the payment channel based on available capacity and transaction priorities. Funds received on a payment channel remain in a pending state until the final receiver provides the key for the hash lock.

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 [12] to ensure privacy of user payments. Spider routers can use similar mechanisms for each transaction unit to provide privacy [4].

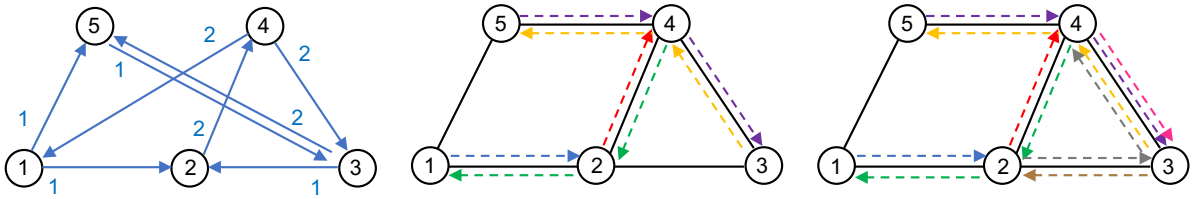
A Spider router queues transaction units when it lacks the funds to send them immediately (Fig. 3). As it receives funds from the other side of the payment channel, it uses them to send new transaction units from its queue. Funds from new-arrived transaction units are not available to use immediately. The router must wait until it receives the key for the hash lock 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.

The queuing of transaction units at Spider routers could result in increased delays for some payments. However, the routers can offer different classes of service or schedule transaction units based on payment requirements. For example, they can prioritize payments based on size, deadline, or routing fees [8].

5 Routing

The key to effective routing in payment channel networks is 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 payments eventually runs out of funds and cannot send further payments until it either receives funds from the other side, or it deposits new funds into the payment channel via an on-chain rebalancing transaction. Since on-chain transactions are expensive and slow, it is desirable to avoid them as much as possible.

In this section, we take a first-principles approach to routing with rate-imbalance constraints. We first answer the question: What is the maximum achievable throughput in a payment channel network with and without on-chain rebalancing transactions? We use a fluid model of the network, wherein transactions between source, destination pairs are modeled as continuous flows, to show that the maximum achievable throughput depends on properties of a *payment graph* that captures how currency flows between network participants (§5.2.2). We formulate optimization problems for routing with rate-imbalance constraints. These optimization problems generalize classical network utility maximization [21] formulations for routing and rate control in communication networks [15, 10]. We discuss how dual decomposition of these optimization problems leads naturally



(a) Payment graph **(b)** Shortest-path routing **(c)** Optimal balanced routing

Figure 4: Example illustrating balanced routing. (a) The payment graph shows the desired transaction rates between pairs of nodes. (b) and (c) The maximum transaction rate achievable with shortest-path routing and the optimal balanced routing schemes on the topology shown. Each dotted edge represents 1 unit of flow along the direction of the arrow. The colors indicate separate flows.

to decentralized algorithms for imbalance-aware routing in payment channel networks (§5.3).

5.1 A Motivating Example

To illustrate the importance of balanced routing, consider the 5-node payment-channel network shown in Fig. 4. Suppose sender, receiver pairs seek to transact at the rates shown in Fig. 4a. 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 for a specific network topology. In each case, we impose the constraint that the net rate in either direction of any edge must be equal to ensure that payment channels do not run out of funds.

Fig. 4b 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 rate-balance constraints. However, an alternate routing scheme of Fig. 4c 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.

5.2 Limits on Throughput

5.2.1 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 transactions 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 $\mathcal{P}_{i,j}$.

The network attempts to satisfy the demands $d_{i,j}$ 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 transactions

along the flow's path such that the average rate of currency transferred along the path equals the value of the flow.

Valid flows must obey two main constraints. First, payment channels are limited in their capacity to process a large rate of transactions, because funds allocated to a transaction are unavailable to use for other transactions for some amount of time. Assume it takes Δ seconds on average for a transaction to reach its destination, get confirmed, and its hash lock key to reach the next payment channel on the upstream path. Then a payment channel with c units of capacity can support, on an average, transactions of net value no more than c/Δ currency units per second. Second, to maintain balance the total flow along the two directions of a payment channel must be the same. If the flows are not balanced, the channel will eventually become unusable along one of the directions unless it is periodically rebalanced via on-chain transactions (we revisit this possibility below).

With these constraints, maximizing throughput in the fluid model is equivalent to finding flows of maximum total value and can be formulated as a Linear Program (LP):

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

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

$$\sum_{p \in \mathcal{P}:(u,v) \in p} x_p + \sum_{p \in \mathcal{P}:(v,u) \in p} x_p \leq \frac{c_{(u,v)}}{\Delta} \quad \forall (u,v) \in E \tag{3}$$

$$\sum_{p \in \mathcal{P}:(u,v) \in p} x_p - \sum_{p \in \mathcal{P}:(v,u) \in p} x_p \leq 0 \quad \forall (u,v) \in E \tag{4}$$

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

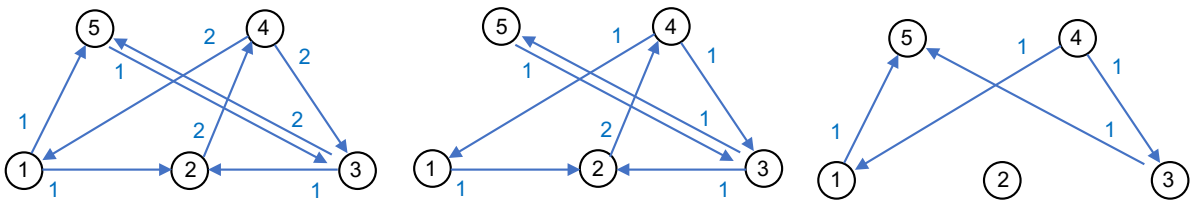
Here, x_p denotes the flow along path p and $\mathcal{P} = \cup_{i,j \in V} \mathcal{P}_{i,j}$ is the set of all paths. The constraints in eq. (3) reflect the capacity limits of payment channels, and the constraints in eq. (4) enforce the balance requirement.

5.2.2 Throughput under Perfect Balance

Next, we show that the maximum throughput achievable under perfect balance is fundamentally restricted by the structure of demand $[d]_{i,j}$ across nodes. Specifically the maximum throughput possible is equal to the largest amount of "circulation" contained in the demands, with higher throughputs possible only if payment channels are rebalanced via on chain transactions.

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 do not depend on the network topology; they depend only on the pattern of payments between the network nodes. Payment graphs are useful for analyzing throughput because any flow imbalance across cuts of H cannot be balanced by any routing scheme in the payment channel network G. Fig. 5a shows the payment graph for the example discussed in §5.1.

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 $(i, j) \in E_C$, and the total weight of incoming and outgoing edges being equal at any node. $w_C(i, j)$, $w_H(i, j)$ denote edge weight of edge



(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).

(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* $\nu(C)$ of the circulation, there exists a circulation graph C^* of maximum value for a given payment graph H. Maximum circulation graphs are not necessarily unique. 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 shows 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 $\nu(C^*)$ with perfect balance (i.e., $B = 0$) on a network with payment channels of unlimited capacity. Conversely, no routing scheme can achieve a throughput greater than $\nu(C^*)$ with perfect balance on any network.*

Proof. To see that a throughput of $\nu(C^*)$ is achievable, consider routing the circulation along any spanning tree T of the payment network G. 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 can be routed. We claim that such a routing scheme is perfectly balanced on all the links. This is because for any partition $S, V\backslash S$ of C^*, the net flow going from S to $V\backslash S$ is equal to the net flow going from $V\backslash S$ 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.

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 $H_{\texttt{SCH}} \subseteq H$ be a payment graph where the edges represent the portion of demand that is actually routed in SCH. Since $\nu(H_{\texttt{SCH}}) > \nu(C^*)$, $H_{\texttt{SCH}}$ is not a circulation and there exists a partition $S, V\backslash S$ such that the net flow from S to $V\backslash S$ is strictly greater than the net flow from $V\backslash S$ to S in $H_{\texttt{SCH}}$. However, the net flows routed by SCH across the same partition $S, V\backslash S$ 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 $\nu(C^*)$. □

Proposition 1 shows that the maximum achievable throughput in a payment channel network with perfect balance can be less than 100% if the demands for payments between nodes is not a circulation. For example, the routing presented in Fig. 4c 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. The DAG component is not routable without violating the balance constraints.

Using payment channels with higher capacity can mitigate problems caused by rate imbalance, but it cannot solve it fundamentally. A payment channel with more funds can sustain rate imbalance for a longer period of time, but it will eventually run out of funds if the rate imbalance persists. To remain usable in both directions, such a payment channel will have to be repeatedly rebalanced via on-chain transactions. In current implementations, rebalancing requires closing a payment channel and reopening a new one. However, two nodes can in principle use multiple smaller payment channels instead of a single large payment channel, so that they can rebalance them one at a time.

We next consider the possibility of such on-chain rebalancing and analyze its impact on throughput in the fluid model.

5.2.3 Throughput with On-Chain Rebalancing

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

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

$$\text{maximize} \qquad \sum_{i,j \in V} \sum_{p \in \mathcal{P}_{i,j}} x_p - \gamma \sum_{(u,v) \in E} b_{(u,v)} \tag{6}$$

$$\text{s.t.} \qquad \sum_{p \in \mathcal{P}_{i,j}} x_p \leq d_{i,j} \quad \forall i, j \in V \tag{7}$$

$$\sum_{p \in \mathcal{P}:(u,v) \in p} x_p + \sum_{p \in \mathcal{P}:(v,u) \in p} x_p \leq \frac{c_{(u,v)}}{\Delta} \quad \forall (u,v) \in E \tag{8}$$

$$\sum_{p \in \mathcal{P}:(u,v) \in p} x_p - \sum_{p \in \mathcal{P}:(v,u) \in p} x_p \leq b_{(u,v)} \quad \forall (u,v) \in E \tag{9}$$

$$x_p \geq 0 \quad \forall p \in \mathcal{P} \tag{10}$$

$$b_{(u,v)} \geq 0 \quad \forall (u,v) \in E. \tag{11}$$

The objective in the problem above consists of two terms. The first term is the total throughput, which we would like to maximize. The second term is the "cost" of on-chain rebalancing at a total rate of $\sum_{(u,v) \in E} b_{(u,v)}$, which we would like to minimize. The parameter γ dictates how we weight these two conflicting goals. We can interpret γ as the increase in throughput required to offset one unit of on-chain rebalancing rate. In practice, γ would depend on a variety of factors, such as the ratio of blockchain transaction fees to routing fees and how the transaction fees are computed. For

example, if blockchain transaction fees do not depend on transaction size, routers can reduce the cost of rebalancing by using larger transactions, thereby effectively reducing γ in the above model.

We now analyze how on-chain rebalancing impacts throughput. Let $B \triangleq \sum_{(u,v)\in E} b_{(u,v)}$ be the total rate of on-chain rebalancing, and let's assume for simplicity that the payment channel capacities are unlimited. For large γ, B will be close to 0 in the optimal solution. From Proposition 1, we know that the maximum achievable throughput in this case is $\nu(C^*)$. On the other hand, for $\gamma \approx 0$, B can become arbitrarily large. It is not difficult to see that the network throughput can satisfy all of the demand $\sum_{i,j} d_{i,j}$ in this case, provided that the links have sufficient capacity (i.e, if the constraints (8) are not tight). In general, as γ decreases, the total throughput and total rebalancing rate both increase in the optimal solution, until the throughput reaches the maximum possible throughput given the capacity constraints.

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

$$\text{maximize} \quad \sum_{i,j\in V} \sum_{p\in\mathcal{P}_{i,j}} x_p \tag{12}$$

$$\text{s.t.} \quad \sum_{p\in\mathcal{P}_{i,j}} x_p \leq d_{i,j} \quad \forall i,j \in V \tag{13}$$

$$\sum_{p\in\mathcal{P}:(u,v)\in p} x_p + \sum_{p\in\mathcal{P}:(v,u)\in p} x_p \leq \frac{c_{(u,v)}}{\Delta} \quad \forall (u,v) \in E \tag{14}$$

$$\sum_{p\in\mathcal{P}:(u,v)\in p} x_p - \sum_{p\in\mathcal{P}:(v,u)\in p} x_p \leq b_{(u,v)} \quad \forall (u,v) \in E \tag{15}$$

$$\sum_{(u,v)\in E} b_{(u,v)} \leq B \tag{16}$$

$$x_p \geq 0 \quad \forall p \in \mathcal{P} \tag{17}$$

$$b_{(u,v)} \geq 0 \quad \forall (u,v) \in E. \tag{18}$$

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

5.3 Algorithms

In this section, we derive decentralized algorithms for routing and rate control in payment channel networks based on the previous optimization problems. These algorithms follow naturally from a dual decomposition of the optimization problems and have the following basic structure. Each payment channel has a *price* in each direction. Routers locally update these prices based on both congestion and imbalance across payment channels. To select paths, end-hosts monitor the total

price of different paths and choose the cheapest options. Using dual variables for prices is common in the utility-maximization-based rate control and routing literature (e.g., see [15]). A key difference from prior work, however, 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 helps steer the routing towards rebalancing the payment channels.

We focus on the optimization problem in eqs. (6)–(11) for the general case with on-chain rebalancing. The algorithm without on-chain rebalancing is a special case.

Consider the partial lagrangian:

$$\begin{aligned} L(\mathbf{x}, \mathbf{b}, \lambda, \mu) = \sum_{i,j \in V} \sum_{p \in \mathcal{P}_{i,j}} x_p - \gamma \sum_{(u,v) \in E} b_{(u,v)} \\ - \sum_{(u,v) \in E} \lambda_{(u,v)} \left[\sum_{p \in \mathcal{P}:(u,v) \in p} x_p + \sum_{p \in \mathcal{P}:(v,u) \in p} x_p - \frac{c_{(u,v)}}{\Delta} \right] \\ - \sum_{(u,v) \in E} \mu_{(u,v)} \left[\sum_{p \in \mathcal{P}:(u,v) \in p} x_p - \sum_{p \in \mathcal{P}:(v,u) \in p} x_p - b_{(u,v)} \right], \end{aligned} \tag{19}$$

where $\lambda_{(u,v)}$ and $\mu_{(u,v)}$ are lagrange variables corresponding to the capacity and rate-imbalance constraints in eq. (8) and eq. (9) respectively. The partial lagrangian can be rewritten as:

$$\begin{aligned} L(\mathbf{x}, \mathbf{b}, \lambda, \mu) = \sum_{i,j \in V} \left[\sum_{p \in \mathcal{P}_{i,j}} x_p - \sum_{p \in \mathcal{P}_{i,j}} x_p \sum_{(u,v) \in p} \left(\lambda_{(u,v)} + \lambda_{(v,u)} + \mu_{(u,v)} - \mu_{(v,u)} \right) \right] \\ + \sum_{(u,v) \in E} \left(\mu_{(u,v)} - \gamma \right) b_{(u,v)} + \sum_{(u,v) \in E} \lambda_{(u,v)} \frac{c_{(u,v)}}{\Delta}. \end{aligned} \tag{20}$$

Define $z_{(u,v)} \triangleq \lambda_{(u,v)} + \lambda_{(v,u)} + \mu_{(u,v)} - \mu_{(v,u)}$ to be the price for edge $(u,v) \in E$, and $z_p \triangleq \sum_{(u,v) \in p} z_{(u,v)}$ to be price for path $p \in \mathcal{P}$. The important fact about the above partial lagrangian is that it decomposes into separate terms for the rate variables of each source/destination pair $\{x_p : p \in \mathcal{P}_{i,j}\}$, and the rebalancing rate of each edge $b_{(u,v)}$. This suggests the following primal-dual algorithm for solving the optimization problem in eqs. (6)–(11):

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

$$\begin{aligned} x_p(t) &= x_p(t-1) + \alpha \left(1 - z_p(t)\right), \\ x_p(t) &= \mathrm{Proj}_{\mathcal{X}_{i,j}}(x_p(t)), \end{aligned} \tag{21}$$

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

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

$$b_{(u,v)}(t) = \left[b_{(u,v)}(t-1) + \beta \left(\mu_{(u,v)}(t) - \gamma \right) \right]_+, \tag{22}$$

  where $[\cdot]_+ \triangleq max(\cdot, 0)$.

- **Dual step.** In the dual step, the routers update the edge prices based on capacity and imbalance constraints at the payment channels. Specifically, each edge performs the following updates independently:

$$\lambda_{(u,v)}(t+1) = \left[\lambda_{(u,v)}(t) + \eta\left(\sum_{p\in\mathcal{P}:(u,v)\in p} x_p(t) + \sum_{p\in\mathcal{P}:(v,u)\in p} x_p(t) - \frac{c_{(u,v)}}{\Delta}\right)\right]_+, \quad (23)$$

$$\mu_{(u,v)}(t+1) = \left[\mu_{(u,v)}(t) + \kappa\left(\sum_{p\in\mathcal{P}:(u,v)\in p} x_p(t) - \sum_{p\in\mathcal{P}:(v,u)\in p} x_p(t) - b_{(u,v)}\right)\right]_+. \quad (24)$$

The parameters $\alpha, \beta, \eta, \kappa$ are positive "step size" constants that determine the speed of the dynamics. Using standard arguments, it can be shown that for sufficiently small step sizes, the above algorithm converges to the optimal solution of the optimization problem in eqs. (6)–(11).

The algorithm has the following intuitive interpretation: $\lambda_{(u,v)}$ and $\mu_{(u,v)}$ are prices that vary due to capacity constraints and imbalance at the payment channels. In eq. (23), $\lambda_{(u,v)}$ increases if the net rate across the payment channel (u, v) (in both directions) exceeds its capacity, and it decreases down to zero if there is excess capacity. Similarly, in eq. (24), $\mu_{(u,v)}$ increases if the rate in the (u, v) direction exceeds the rate in the (v, u) direction by more than $b_{(u,v)}$, the rate at which funds are deposited for edge (u, v) on chain. As these prices vary, each source/destination pair reacts by changing the sending rate on its paths according to eq. (21). The effect is to reduce the rate on expensive paths and increase it on cheap paths. Simultaneously, each edge also adapts its on-chain rebalancing rate based on eq. (22). If $\mu_{(u,v)} > \gamma$, the price charged for imbalance is higher than the cost of on-chain rebalancing per unit rate. Hence the edge increases its on-chain rebalancing rate $b_{(u,v)}$; otherwise it decreases its throughput. Finally, we note that in the absence of any on-chain rebalancing, the algorithm can be simplified by setting $b_{(u,v)} = 0$ for all edges (u, v).

We remark that the objective of our optimization problem in eq. (1) can be modified to also ensure fairness in routing, by associating an appropriate utility function with each sender-receiver pair [16]. A decentralized algorithm for such a case may be derived analogously as our proposed solution.

5.3.1 Practical considerations

In practice, routers have to dynamically estimate the rate over their payment channels from the transactions that they encounter. The source nodes, whenever they have to send transactions, query for the path prices, and adapt the rate on each path based on these prices. This rate adaptation can occur in different ways depending on the implementation. For example, in an implementation where source nodes use only one path for routing each transaction, they can select these paths such that the frequency of usage of different paths over time is roughly proportional to the optimal flow rate along the paths. In the next section, we describe a design that allows more fine-grained rate control by splitting transactions into small units than can be transmitted on different paths.

A typical challenge with such iterative algorithms for adjusting path prices and sending rates is slow convergence. An algorithm that is slow to converge may not be able to adapt the routing to changes in the transaction arrival pattern. If the transaction arrival patterns change frequently, this may result in a perpetually suboptimal routing.

Therefore in practice it may be beneficial to also consider simpler approaches to balancing transactions across different paths that can converge quickly. One such approach is for sources to 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 first 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.

For both types of algorithms, practical implementations would restrict the set of paths considered between each source and destination, so that the overhead of probing the path conditions is not too high. There are a variety of possible strategies of selecting these paths, e.g., the K shortest paths or the K highest-capacity paths between every source/destination pair. We leave an investigation of the best way to select the paths to future work.

6 Preliminary Evaluation

6.1 Setup

Simulator. We modified an existing simulator for payment channel networks [7] to model transaction arrivals and completion events. Arriving transactions are routed according to the routing algorithm as long as funds are available on the paths chosen by the algorithm. Payments that are actually routed 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 and when a transaction completes, these funds are released. The simulator supports non-atomic payments through a global queue that tracks all incomplete payments. These transactions are periodically polled to see if they can make any further progress. 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 [3] and a subgraph from the original topology of Ripple [6], an existing currency exchange network that transacts in XRPs. The ISP topology is relatively simple allowing us to reason about the dynamics of the system. We used a graph with 32 nodes and 152 edges and generated $200,000$ transactions to send on it. The transactions were synthetically generated with the sizes sampled from Ripple data after pruning out the largest 10%. The average transaction size for this dataset is 170 XRP with the largest one being 1780 XRP. The sender for each transaction was sampled from the set of nodes using an exponential distribution while the receiver was sampled uniformly at random. We set all edges in the graph to have the same capacity, which we varied from 10000 XRP to 100000 XRP per link in different experiments.

We also used data from the Ripple network from January 2013 [7]. The original dataset had 90,000 nodes and 330,000 edges. We pruned the dataset 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 have an average size of 345 XRP with the largest transaction size being 2892 XRP. Consequently, we set the capacity of all the links in the reduced Ripple graph to 30000.

Schemes. We evaluate SpeedyMurmurs [25], SilentWhispers [18], and max-flow routing. All of these schemes use atomic payments. We 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), the routing algorithms described in §5. Spider (LP) solves the LP in Eq. (1) once based on the long-term payment demands and uses the solution to set a weight for selecting each path. 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 [8].
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 at the end of $200s$ for the ISP topology and $85s$ for the Ripple topology. All the edges 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 SpeedyMurmurs and SilentWhispers 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 an estimate of the demand matrix to make decisions for the entire duration of the simulation. While this approach works for a stationary transaction arrival pattern, as is the case with the ISP topology, it does not work too well for the Ripple network in which the traffic demands vary over time. Further, the LP assigns zero flows to all paths for certain commodities which means no payments between them will ever get attempted. We plan to explore other objectives like proportional fairness [15] 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

This paper proposes a new packet-switched architecture for payment channel networks. A first step in sending transaction units through this network is the discovery of payment paths. To that

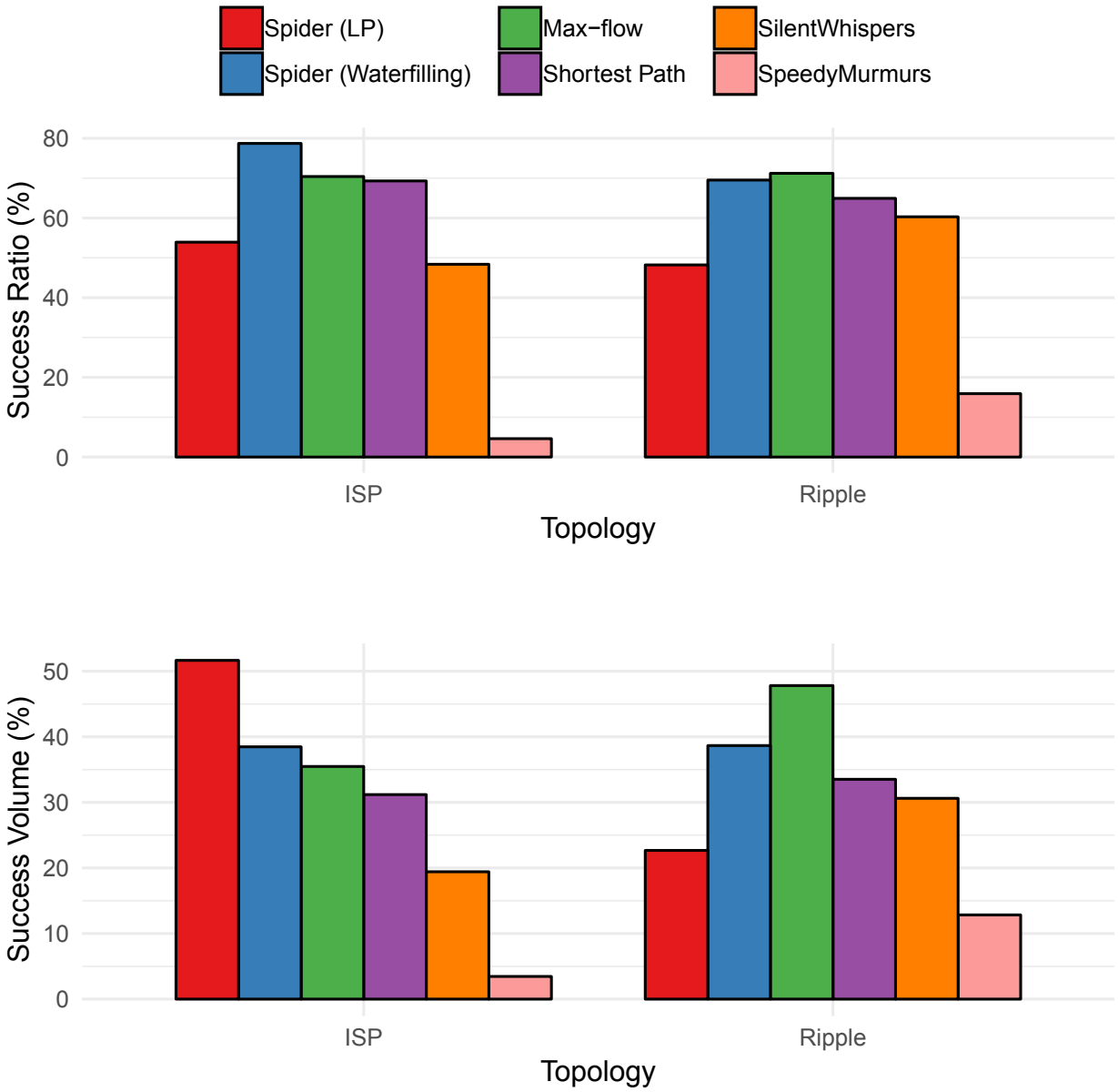


Figure 6: Comparison of payments completed across schemes on the ISP and Ripple Topologies when the capacity per link is 30,000

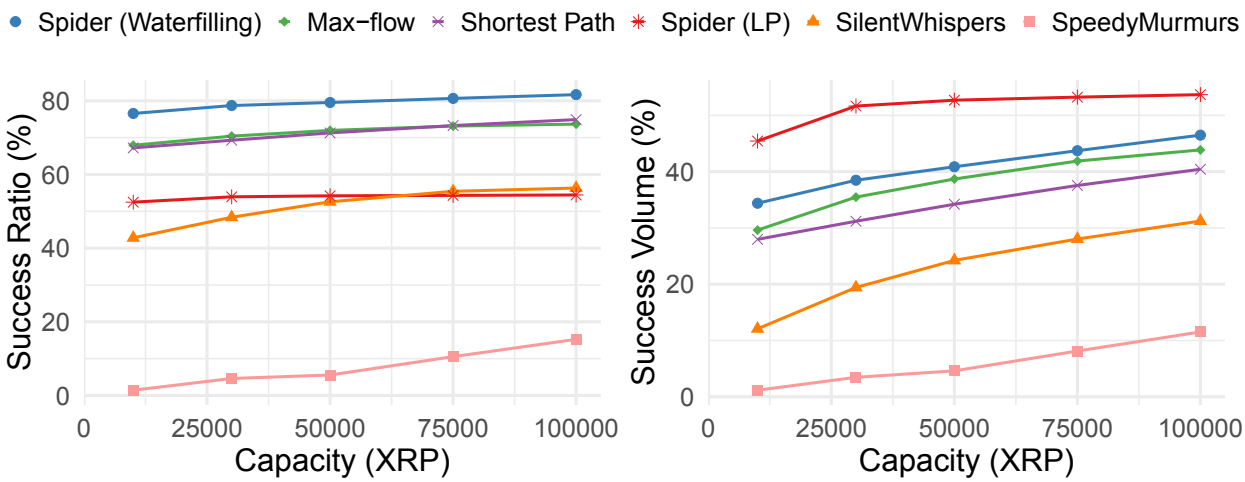


Figure 7: Effect of increasing capacity per link on the success metrics when routing payments on the ISP topology. All links in the network have the same credit.

effect, we explored the benefits of imbalance-aware routing approaches in this paper. However, we envision more potential gains from router and end-host collaboration. Spider routers have information about transaction units belonging to payments across many sender-receiver pairs. This can be leveraged for admission control. In other words, routers can decide payment priorities or reject some extremely large transactions that are unlikely to complete within the deadline. On the other hand, Spider end hosts could employ rate control strategies and their own admission control policies influenced by feedback on payment completion rates or router hints.

Our routing algorithms suggest a way to set routing fees to maximize network throughput with rational users that prefer cheaper routes. However, our design does not address incentives and implications for network service providers that wish to maximize their profits from routing fees. Further, we have assumed routers do not inject additional funds into the payment channels after their initial escrow; however routers have an incentive to periodically replenish funds if it leads to an overall increase in their return on investment from routing fees. An analysis of economic incentives for network stakeholders is an exciting direction for future work.

Lastly, we have focused on optimizing payment delivery for a given network topology and payment channel capacities. However, our results show fundamental limits to performance based on the flow of currency across the network. Understanding how best to design network topology and influence its formation in a decentralized payment channel network is an important area for investigation.

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] Isp topology zoo. http://www.topology-zoo.org/.

[4] Onion routed micropayments for the lightning network. https://github.com/lightningnetwork/lightning-onion.

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

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

[7] Speedymurmurs software. https://crysp.uwaterloo.ca/software/speedymurmurs/.

[8] 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.

[9] L. Cuen. 100 merchants can now trial Bitcoin's lightning network risk free. July 2018.

[10] 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.

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

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

[13] A. Hertig. Lightning: The Bitcoin scaling tech you really should know. December 2017.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] J. Medley. Bitcoin lightning network: Scaling cryptocurrencies for mainstream use. July 2018.

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

[21] 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.

[22] J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments. *draft version 0.5*, 9:14, 2016.

[23] 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_n etwork_7_7_2016. pdf)*, 2016.

[24] 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.

[25] 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.

[26] 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-1461`

[27] 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.

[28] 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.

[29] J. Vermeulen. Bitcoin and ethereum vs visa and paypal ? transactions per second. April 2017.

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

[31] 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.