

High-Efficiency Cryptocurrency Routing in Payment Channel Networks

by

Vibhaalakshmi Sivaraman

B.S.E., Princeton University (2017)

Submitted to the Department of Electrical Engineering and Computer
Science

in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2019

© Massachusetts Institute of Technology 2019. All rights reserved.

Author
Department of Electrical Engineering and Computer Science
August 30, 2019

Certified by
Mohammad Alizadeh
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

High-Efficiency Cryptocurrency Routing in Payment Channel Networks

by

Vibhaalakshmi Sivaraman

Submitted to the Department of Electrical Engineering and Computer Science
on August 30, 2019, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

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 (PCNs) exacerbate these problems by attempting to deliver all payments atomically. This thesis presents the Spider protocol, a protocol inspired by congestion control for data networks that addresses these challenges. We formalize the PCN routing problem as an optimization problem and motivate Spider using that. Spider splits payments into transaction units and uses an explicit multipath transport protocol to control the rates at which the transaction units are sent through the PCN. Spider routers signal congestion to end-hosts based on observed queuing delay and end-hosts accordingly adjust sending rates on their paths. This thesis shows through extensive simulations that Spider requires less than 25% of the funds needed by state-of-the-art approaches to successfully route over 95% of the transactions across a wide range of synthetic and real topologies. Our improvements are significant across all sizes of transactions: Spider completes 40% more of the largest 25% of transactions attempted on the real Lightning Network topology compared to the state-of-the-art.

Thesis Supervisor: Mohammad Alizadeh

Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

I first thank my advisor, Mohammad for being incredibly patient and approachable both with me and this project. His attention to detail and unwavering focus have taught me to be structured and meticulous about the research I undertake. His ability to employ mathematical tools to explain the most bizarre experimental results is something I strive to develop.

I couldn't have found a better collaborator in Shaileshh. He has been a constant sounding board for any ideas I come up with and has spent many painful hours reasoning through this protocol with me. I am grateful for his ability to explain seemingly esoteric concepts in simple words. I might not be comfortable using mathematical formulations yet, but I certainly am less fearful of it, thanks to him.

This work would not have been possible without my many other collaborators: Kathy, Lei, Pari, Radhika, Giulia and Pramod. I am thankful to every one of them for their contributions to this project.

MIT wouldn't be the same without my lab-mates who've made coming into work enjoyable. Thanks to Rachel, Nalini, Yamin, Pritpal and Siddhartha for turning MIT and Boston into a second home. Thanks also to all friends from undergrad who have been a welcome break from the research world.

My parents and my grandma are a constant pillar of support from afar. Their reassurances in stressful times have given me the strength to keep going. My brother has been a great resource both at a personal and a professional level, teaching me about doing good research and keeping a level-head at the same time. Lastly, thanks to Arjun for asking the hard questions all the time even when I despise them most.

Contents

1	Introduction	13
2	Background	17
3	Challenges in Today's Payment Channel Networks	19
4	Theoretical Framework	25
4.1	Architecture Overview	25
4.2	The Routing Problem	27
4.2.1	Implications for Throughput	28
4.3	A Primal-Dual Decomposition Based Approach	29
4.3.1	Router Design	30
4.3.2	Transport Layer Design at End-Hosts	32
4.3.3	Challenges	33
5	A Practical Protocol	35
5.1	Intuition Towards a Practical Solution	35
5.2	The Spider Protocol	40
5.2.1	Message Formats	40
5.2.2	Spider Router Design	41
5.2.3	Spider Transport Layer Design at End-Hosts	41
6	Evaluation	43
6.1	Simulator Implementation	43

6.2	Routing Schemes	45
6.3	Experimental Setup	47
6.4	Performance on Circulation Payment Graphs	51
6.5	Effect of Adding DAGs	54
6.6	Spider's design choices	57
7	Related Work	59
8	Discussion	63
9	Conclusion	65
	Appendix	75
9.1	Throughput Bounds	75
9.2	Primal-Dual Algorithm Derivation	76
9.3	Estimating the Demand-Capacity Gap at the Routers	79

List of Figures

2-1	Bidirectional payment channel between Alice and Bob. A blue shaded block indicates a transaction that is committed to the blockchain.	18
2-2	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.	18
3-1	Example illustrating the problems with state-of-the-art PCN routing schemes. . . .	20
4-1	Basic Architecture for Spider. Routers compute channel imbalance and congestion prices based on queued up and arriving transactions. End-hosts maintain rates for each path to a receiver. They periodically send out probes to collect path prices which are used to update the rates.	26
4-2	Payment graph (denoted by blue lines) for a 3 node network (left). It decomposes into a maximum circulation and DAG components as shown in (b) and (c).	29
5-1	Model of queues at a payment channel between nodes u and v . x_{uv} and y_{uv} denote the rates at which transaction-units for v arrive into and get serviced at the queue at u respectively. c_{uv} is the capacity of the payment channel and q_{uv} denotes the total number of transaction-units waiting in u 's queue to be serviced.	36

5-2	Dynamics of running a simple protocol that responds to queue buildup at routers on a toy PCN. When the demand at v is much lesser than both the demand at u and the capacity of the uv payment channel, both nodes' sending rates drop to match the queue service rates dictated by the smaller of the two demands causing a controlled queue at u . When the demand at v is much lesser than the demand at u but is higher than half the capacity of the uv payment channel, the sending and servicing processes become uniform at the same rate (83€/s) keeping both router queues at a constant amount.	38
5-3	Spider Architecture: Routers queue up transaction-units and schedule them across payment channels based on available capacity and transaction priorities. If the delay through the queue for a packet exceeds a threshold, they mark the packet. End-hosts maintain windows for each path to a receiver which are adjusted based on whether acked transaction-units are marked.	40
6-1	Topology and transaction dataset used for real-world evaluations.	48
6-2	Performance of different algorithms on different topologies with different per sender transaction arrival rates. Spider consistently outperforms all other schemes achieving near 100% average success ratio. Error-bars denote the maximum and minimum fraction of successful transactions across five runs with different circulation graphs. Note the log scale of the x-axes.	52
6-3	Breakdown by size across of performance of different schemes across all topology and capacity distributions. Each point reports the fraction of successful transactions whose size belongs to the interval denoted by the shaded region. Each interval corresponds roughly to 12.5% of the CDF denoted in Fig. 6-1b. The graphs correspond to the (right) midpoints of the corresponding channel sizes in Fig. 6-2 .	53
6-4	CDF of normalized throughput achieved by different flows under different schemes across topologies. Spider achieves close to 100% throughput given its proximity to the black line denoting demand of flows. Spider is a more vertical line than the baseline LND scheme showing that it is fairer than other schemes: it doesn't hurt the throughput of smaller flows to attain good overall throughput.	54

6-5	Performance of different algorithms across all topologies as the DAG component in the transaction demand matrix is varied. As the DAG amount is increased, the normalized throughput achieved is further away from the expected optimal circulation throughput. The gap is more pronounced on the real topology.	55
6-6	Comparing throughput when a pure circulation demand is run for 3000s to a scenario where a circulation demand is restored for 1000s after 2000s of a demand with 20% DAG. The throughput achieved on the last 1000s of circulation is not the expected 100% even after the DAG is removed	56
6-7	Performance of Spider as the number of edge-disjoint shortest paths considered per sender-receiver pair is varied on two different topologies.	57
6-8	Performance of Spider as the type of paths considered per sender-receiver pair is varied.	58
9-1	Example payment graph (denoted by blue lines) for a five node network (left). It decomposes into a maximum circulation and DAG components as shown in (b) and (c).	75
9-2	Update figure. 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.	79

--

Chapter 1

Introduction

Despite growing adoption of cryptocurrencies in the last decade [17], they are not commonly used for everyday transactions (e.g., retail). There are many reasons for this, but one major reason is poor scalability. For example, the Bitcoin network processes 7 transactions per second, and Ethereum 15 transactions/second, with latencies ranging from tens of minutes (Ethereum) to hours (Bitcoin) [42]. For comparison, the Visa network processes around 1,700 transactions per second on average, and up to 24,000 transactions per second [58]. The poor scalability of major cryptocurrencies stems mainly from inefficiencies in the underlying consensus protocol; every transaction must go through the full consensus protocol to be confirmed, which can be time-consuming and expensive.

A leading proposal for improving the scalability of cryptocurrencies is **payment channel networks** (PCNs). At a high level, PCNs are overlay networks that allow a transaction to be quickly verified by a recipient without submitting the transaction to the blockchain for confirmation. PCNs rely on so-called *payment channels*. A payment channel is a cryptocurrency transaction that escrows money on the blockchain for a predetermined duration, with a prespecified destination. For example, Alice can set up a payment channel with Bob in which she escrows 10 tokens for a month. Now Alice can send Bob transactions from the escrow account, and Bob can validate them without submitting them to the blockchain. This is because (a) the payment channel is recorded on the blockchain, so he knows the escrow account is valid, and (b) the channel specifies Bob as the only possible recipient, so he can be sure the funds were not double-spent by checking the validity of Alice's signature

and her past transactions with him. If Bob or Alice want to close the payment channel at any point, they can broadcast the most recent signed transaction message to the blockchain to finalize the transfer of funds.

A PCN is a network of bidirectional payment channels. The key idea is that if Alice and Charlie each share a channel with Bob but not with each other, they can route transactions through Bob for a nominal fee. This enables fast, secure transactions without requiring nodes to maintain payment channels with every possible transaction partner. PCNs have received a great deal of attention in recent years, with the Lightning network currently being built for Bitcoin [8, 6], and the Raiden network for Ethereum [13]. PCNs are viewed by many as a primary avenue for scaling existing blockchains whose consensus protocols are already fixed [55].

Despite their promise, many open questions remain about the long-term viability of PCNs. To run a PCN, operators must lock up their funds in payment channels, potentially incurring substantial opportunity cost. In principle, PCNs offset this cost by giving routers a fractional routing fee of each transaction they route. Hence the cost-benefit tradeoff depends critically on the efficiency of the network: the more transactions being routed per unit of locked-up collateral, the more routing fees get collected. *High-throughput transaction routing* is therefore of central importance.

However, we find that existing routing protocols achieve poor throughput for two main reasons. First, existing systems choose naive shortest-path routing for transactions. This leads to congestion and *imbalance* on payment channels. Notice that even if Alice and Bob start out with a balanced channel (say each puts in 10 tokens), then if Alice sends 10 tokens to Bob, the entire balance of their channel will reside with Bob. Now Alice cannot execute any more transactions without refilling the channel via an on-chain transaction (i.e., committing a new transaction to the blockchain). In a PCN, routing is only successful if all channels on the route have sufficient funds. Hence imbalanced channels lead to reduced throughput throughout the network, not just at the depleted link.

A second problem with existing systems is that they route each incoming transaction atomically and instantaneously, in full. This approach is particularly harmful for larger transactions, which can fail completely if there is no path to the destination with enough

supporting balance. Even if the transaction does not fail, this approach can result in imbalanced channels that later reduce throughput for everyone else.

The Lightning network is currently in its early days, and most of its transactions are still small [57]. For this reason, some of the issues we pointed out have not become critical yet. However, as PCNs grow in adoption, transaction distributions are likely to become larger, with heavier tails. Because of this, the listed problems could create serious scalability bottlenecks.

The purpose of this thesis is both to study problems with existing routing solutions and propose a viable alternative. We begin by systematically demonstrating a number of problems with existing PCN routing mechanisms. We do this by first analyzing the theoretically-optimal throughput possible for different demands and topologies, and then simulate existing approaches in a new packet-based simulator for PCNs. These results suggest that existing routing and transport mechanisms achieve throughput far from the theoretical limits.

We next introduce an explicit multipath transport protocol for PCNs called Spider. Spider is a packet-switched architecture for *balance-aware*, high-throughput routing in PCNs. We develop Spider by posing the routing problem as an optimization problem in which channels are constrained to remain balanced. We draw parallels to Internet congestion control and use that to derive fully-distributed update rules for the rate control and routing decisions of each node. In Spider, routers independently mark packets based on the queueing delay observed by transactions going through them, in turn, signaling to end-hosts to control windows for transactions on their paths. End-hosts increase windows when packets are not marked and decrease them otherwise.

We show through extensive simulations that Spider requires less than 25% of the funds needed by state-of-the-art approaches to successfully route over 95% of the transactions across a variety of synthetic and real graph topologies and channel capacity distributions. For a given budget of funds in a PCN modelled off the real Lightning Network, Spider is able to successfully route upto 50% more transactions than the state-of-the-art. Simultaneously, Spider is able to complete 40% more of the largest 25% of transactions. Despite packetization, transactions in Spider complete within at most a few seconds.

Chapter 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 [47]. Suppose Alice puts 3 units in the channel, and Bob puts 4 (Fig. 2-1). 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. 2-1). 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 [47].

A payment channel network is a collection of bidirectional payment channels (Fig. 2-2). 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. 2-2, 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 [13]. 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.

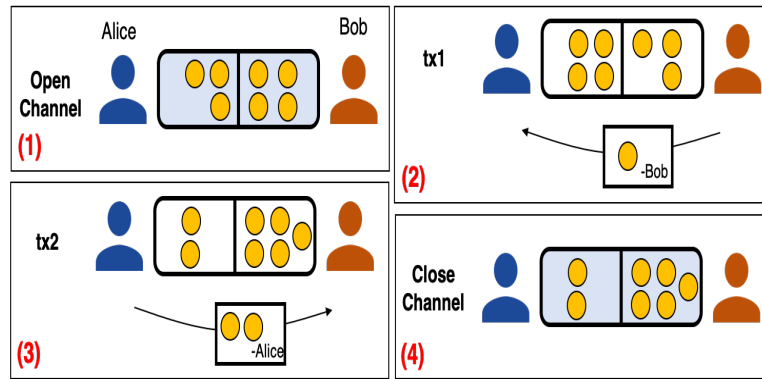


Figure 2-1: Bidirectional payment channel between Alice and Bob. A blue shaded block indicates a transaction that is committed to the blockchain.

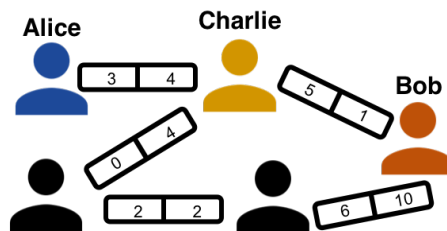


Figure 2-2: In a payment channel network, Alice can transfer money to Bob by using intermediate nodes' channels as relays. There are two paths from Alice to Bob, but only the path (Alice, Charlie, Bob) can support 3 tokens.

Chapter 3

Challenges in Today's Payment Channel Networks

A major cost of running PCNs is the collateral needed to set up payment channels. As long as a channel is open, that collateral is locked up, incurring an opportunity cost for the owner. For PCNs to be financially viable, this opportunity cost should be offset by routing fees, which are charged on each transaction that passes through a router. To collect more routing fees, routers try to process as many transactions as possible for a given amount of collateral. We refer to the total collateral in the system as the network's *capacity*; a key performance metric is therefore the *transaction throughput per unit capacity*, where throughput can be measured either in number of transactions per second or transaction value per second. Although transaction latency also matters, successfully-completed transactions in a PCN typically have latencies several orders of magnitude lower than the alternative (on-chain transactions), so by maximizing throughput, we are also controlling latency. We therefore focus on completing transactions on timescales of seconds, not minimizing latency.

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

Where to route transactions. A central question in PCNs is what route(s) to use for sending a transaction from sender to destination. PCNs like the Lightning and Raiden

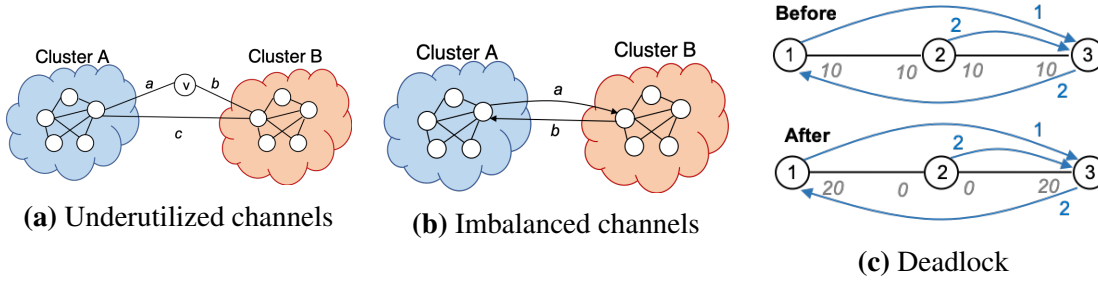


Figure 3-1: Example illustrating the problems with state-of-the-art PCN routing schemes.

networks are source-routed. This was done in part for privacy reasons: transactions in the Lightning network use onion-routing, which is easy to implement with source routing. Most clients by default pick the shortest path from the source to the destination.

However, shortest-path routing degrades throughput in two key ways. The first is to cause underutilization of the network. To see this, consider the PCN shown in Fig. 3-1a. Suppose we have two fully-connected clusters of nodes that seek to transact with each other at the same rate, and the clusters are connected by two paths, one consisting of channels $a - b$, and the other channel c . If the nodes in cluster A try to reach cluster B via the shortest path, they would all take channel c , as would the traffic in the opposite direction. This leads to congestion on channel c , and underutilization of edges a and b .

A second problem is more unique to PCNs. Consider a similar topology in Figure 3-1b, and suppose we fully utilize the network by sending all traffic from cluster $A \rightarrow B$ on edge a and all traffic from cluster $B \rightarrow A$ on edge b . Since the clusters are transacting at the same rates, the rate on both edges is the same. However, in a payment channel, as funds flow in one direction over a channel, the channel becomes *imbalanced*: all of the balance ends up on one side of the channel. Because of this, 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. Naive routing schemes like shortest-path routing do not account for this problem, thereby leading to reduced throughput (§6). Hence, to exhibit good throughput per unit capacity, it is important to choose routes that actively prevent channel imbalance. For example, in Figure 3-1b, we could send half of the $A \rightarrow B$ traffic

on edge a , and half on edge b , and the same for the $B \rightarrow A$ traffic. The challenge is making these decisions in a fully-automated way.

When to send transactions. Another central problem is *when* to send transactions. Most existing PCNs are circuit-switched, with transactions being processed instantaneously and atomically upon arrival into the system [47, 13]. Even when routes are chosen optimally, this can lead to a number of problems. One is that if a transaction’s value exceeds the available balance on each path from the source to the destination, the transaction fails. Since transaction values in the wild tend to be heavy-tailed [31, 23], either a substantial fraction of real transactions will fail as PCN usage grows, or payment channel operators will need to provision higher collateral to satisfy demand.

Even when transactions do not fail outright, the default policy of sending transactions instantaneously and atomically exacerbates the imbalance problem by sending full transactions over the same channel at the same time, transferring the full transaction value to one side of the channel. A natural idea to alleviate these problems is to packetize transactions: transactions can be split into smaller transaction units that can be multiplexed over space (by traversing different paths) and in time (by being sent at different rates). Versions of this idea have been proposed before; atomic multi-path payments (AMP) enable transactions to traverse different paths in the Lightning network [4], and the Interledger protocol uses a similar packetization to conduct cross-ledger payments [54]. However, a key observation is that it is not enough to subdivide transactions into smaller units: to achieve good throughput, it is also important to multiplex in *time* as well, by executing rate control.

In particular, consider the PCN shown in Fig. 3-1b and a scenario where the nodes in the two clusters want to transact with each other at the same average rate, but there might be transient bursts of large transactions. Assume that the routes are picked such that over a long period the rates on both edges a and b match in the two directions. If the edges a and b just happen to have all the funds accumulated on cluster B’s end at a point in time, cluster A cannot send any transactions. Particularly, a large transaction (say 10 units) cannot be immediately sent out by merely splitting it into smaller units. However, if we were able to schedule it appropriately and multiplex in time, nodes in cluster A could wait

for transactions to arrive from cluster B and send out the 10 units over a period of time such that the average rate matches in the two directions.

Deadlocks. The third challenge in PCNs is the idea that the introduction of certain flows can actively harm the throughput achieved by other flows in the network. To see this, consider the topology and demand rates in Figure 3-1c. Suppose users 1 and 2 want to transmit 1-unit transactions to node 3 at rates of 1 and 2 units/second, respectively, and node 3 wants to send to node 1 at rate 2 units/sec. Notice that the specified transaction rates are imbalanced: there is a net flow of funds out of node 2 and into nodes 1 and 3. Suppose the payment channels are initially balanced, with 10 units on each side and we only start out with flows between nodes 1 and 3. For this demand and topology, the system can achieve a total throughput of 2 units/sec by having nodes 1 and 3 to send to each other at a rate of 1 unit/second. Notice also that since the payment graph is a tree, there is only one path between each pair of nodes, so route selection is not a variable in this problem.

However, once transactions from node 2 are introduced, this example achieves zero throughput at steady-state. The reason is that node 2 sends transactions to node 3 faster than its funds are being replenished, which reduces its balances to 0. Even if transactions out of node 2 are slowed down, they will still deplete the funds at 2 over a longer period of time. Since node 2 needs positive balance to route transactions between nodes 1 and 3, the transactions between 1 and 3 cannot be processed, even though each of the endpoints locally has sufficient balance. The network finds itself in a *deadlock* that can only be resolved by node 2 replenishing its balance with an on-chain transaction (or *rebalancing* its channels). Notice that this situation could not have been solved by just rate-control; it requires active knowledge of which flows are draining funds and stopping them entirely. We show the same effect in a larger experiment in §6.5.

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

mechanism. Hence, maximizing throughput in a PCN is difficult in part because we cannot subdivide the problem into separable components, making it difficult to evaluate proximity to a “good” solution.

Another challenge is the fact that PCNs behave very differently from traditional communication networks. First of all, unlike network bandwidth which might recover over time from an outage, the resources on each link are physical quantities that cannot be replenished simply by waiting: the tokens in a payment channel are restored only when a neighbor sends more transactions. Such problems have been explored in the context of ridesharing, for instance [20, 21], and require new innovation in both formulating and solving routing problems. Second, the way we think about network capacity in the PCN problem is quite different from other networking applications. In traditional applications, a link’s capacity is viewed as a fixed, physical quantity. However, in PCNs, the capacity is dependent on the input rate of transactions into channels and their queue sizes also. Further, a depleted link can actually degrade throughput in other parts of the network. This leads to cascading effects that make congestion control particularly critical. We explain this further in Section 4.

Chapter 4

Theoretical Framework

In this chapter, we first discuss the packet-switched architecture (§4.1) that Spider uses for its routing protocol. We then formulate the routing problem as an optimization problem §4.2 and describe a first-principles approach to solving it in §4.3.

4.1 Architecture Overview

As laid out in §3, state-of-the-art approaches suffer from poor throughput in PCNs because they do not carefully control *where* and *when* transactions are sent out into the network. Spider holistically addresses the above challenges. First, to handle variable transaction sizes, Spider uses a packet-switched architecture that splits transactions into a series of *transaction units*, with each transaction unit transferring a small amount of money bounded by a *maximum transaction unit (MTU)*. This is inspired by packet-switching for the Internet which has been shown to be more effective than circuit-switching [39]. These transaction-units are then sent over different routes at different times. The spatial and temporal diversity of packet switching, when combined with rate control, enables PCNs to deal with much more diverse transaction loads *with less collateral in the network*.

Spider’s packet-switched architecture involves end-hosts that initiate payments and transmit them over the network as a series of transaction-units. Transaction splitting doesn’t compromise on the security of payments as transaction-units can be transmitted independently with separate keys per transaction-unit. As receivers receive transaction-units and

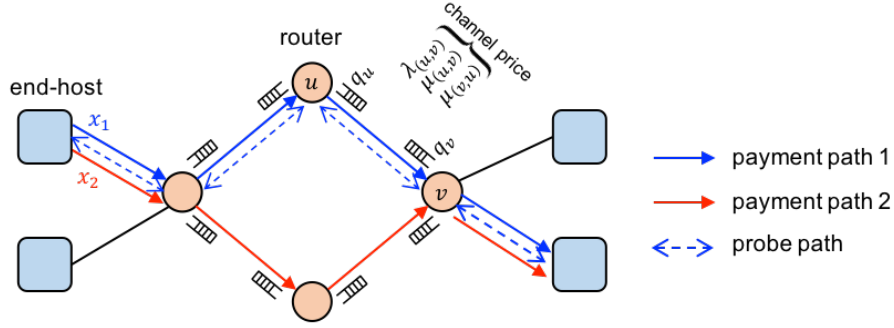


Figure 4-1: Basic Architecture for Spider. Routers compute channel imbalance and congestion prices based on queued up and arriving transactions. End-hosts maintain rates for each path to a receiver. They periodically send out probes to collect path prices which are used to update the rates.

acknowledge them, senders can selectively reveal secrets for acknowledged transaction-units alone (§2). Senders who want atomic payments can also use proposals like Atomic Multi-Path Payments (AMP) [4] to guarantee atomic completion of payments that are split across multiple paths.

Transaction-units from different end-hosts are forwarded from the source to the destination by Spider routers inside a PCN. Spider routers queue up transactions at a given payment channel whenever the channel lacks the funds to forward the transaction(s) immediately. As a router receives funds from the other side of its payment channel, it uses the funds to forward more transaction-units from the queue. In current PCN implementations, a transaction fails immediately if it encounters a channel with insufficient balance on its route. Thus, even temporary lack of channel balance can cause many transactions to fail, which Spider avoids.

Spider, as illustrated in Fig. 4-1, uses an explicit multipath transport layer protocol. End-hosts monitor candidate paths and maintain sending rates for them that are adjusted based on router feedback. It achieves high throughput while maintaining balanced rates at the channels, and controlling congestion. The protocol is inspired by MPTCP [60] and similar algorithms for data networks that also optimize for high utilization and fairness [36, 26] that are known to work in practice. §4.2 formalizes the optimization problem for PCN routing and §4.3 derives a protocol from it.

$G(V, E)$	Graph of the PCN with a set of V routers and E payment channels
\mathcal{P}_{ij}	Set of paths that sender i uses to receiver j
\mathcal{P}	$\bigcup_{i,j \in V} \mathcal{P}_{ij}$
x_p	Average rate of transaction-units on path p between sender i and receiver j
x_{uv}	$\sum_{p \in \mathcal{P}: (u,v) \in p} x_p$
d_{ij}	Demand from sender i to receiver j
c_{uv}	Total amount of tokens escrowed into payment channel (denotes channel size)(u, v)
Δ	Average time (s) over which tokens sent across a payment channel are unusable

Table 4.1: Notation for routing problem

4.2 The Routing Problem

To understand how balance-aware routing can be performed, we consider a fluid model of the system in which payments are modeled as continuous "fluid flows" between users. The fluid model allows us to express the routing problem as an optimization problem and derive useful decentralized algorithms from it, much like the Network Utility Maximization (NUM) framework from traditional networks [45].

Let U denote a concave increasing utility function such that $U_i(x)$ captures the overall utility received by sender i for a rate of x . A canonical choice for $U_i(x)$ is $\log x$, in which case the optimal solution is said to achieve proportional fairness [36] across flows or sender-receiver pairs. Spider optimizes for this utility function in order to ensure that no individual sender's payments are completely throttled. Other choices are also possible though, e.g., if $U_i(x) = x$ then objective captures total throughput. An alternate protocol described in §4.3 optimizes for total throughput.

Let x_p denote the rate on a path p between sender i and receiver j when attempting to satisfy a demand of d_{ij} . x_{uv} captures the total rate on an edge that has a total of c_{uv} tokens. Given the notation in Table. 4.1, the routing problem can be described as the following optimization problem where the goal is to maximize overall social utility.

$$\text{maximize} \quad \sum_{i,j \in V} U \left(\sum_{p \in \mathcal{P}_{ij}} x_p \right) \quad (4.1)$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{P}_{ij}} x_p \leq d_{ij} \quad \forall i, j \in V \quad (4.2)$$

$$x_{uv} + x_{vu} \leq \frac{c_{uv}}{\Delta} \quad \forall (u, v) \in E \quad (4.3)$$

$$x_{uv} = x_{vu} \quad \forall (u, v) \in E \quad (4.4)$$

$$x_p \geq 0 \quad \forall p \in \mathcal{P}. \quad (4.5)$$

Demand Constraints. Equation. 4.2 ensures that no sender-receiver pair sees more total flow across all its paths than the demand dictates.

Capacity Constraints. Routers have limited capital on their payments channels which restricts the maximum rate at which funds can be routed through the channels (Fig. 3-1a). In particular, when transaction-units are sent at a rate x_{uv} across a payment channel between u and v , $x_{uv}\Delta$ credits are locked (i.e., unavailable for use), due to the latency in receiving the secret key from the destination. This implies that the average rate of transactions (across both directions) on a payment channel cannot exceed $\frac{c_{uv}}{\Delta}$ (Equation. 4.3). This is similar to the notion of bandwidth in data networks.

Balance Constraints. The balance constraint stipulates that the total rate at which transaction-units are sent in one direction along a payment channel matches the other. This is a key difference from traditional data networks. In PCNs, if the long-term rates x_{uv} and x_{vu} are mismatched on edge (u, v) (say $x_{uv} > x_{vu}$), over time, all the credits c_{uv} will accumulate at v deeming the directed edge (u, v) unusable (Fig. 3-1b). Thus, the dynamics of payment channels necessitate Equation. 4.4 on every edge in the network. This affects PCN routing and has implications for the maximum achievable transaction throughput.

4.2.1 Implications for Throughput

A consequence of the balance constraints is that certain traffic demands are more efficient to route than certain others. Demands that have a *circulation* structure (total outgoing demand

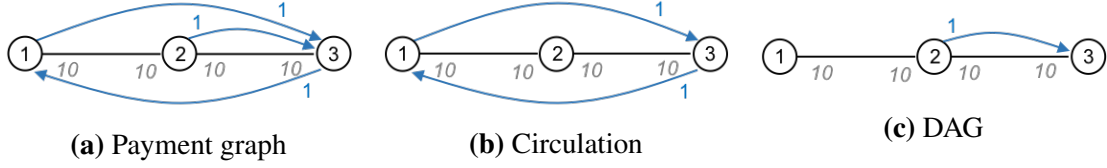


Figure 4-2: Payment graph (denoted by blue lines) for a 3 node network (left). It decomposes into a maximum circulation and DAG components as shown in (b) and (c).

matches total incoming demand at a router) can be routed efficiently. The cyclic structure of such demands enables routing along paths such that the rates are naturally balanced in channels. However, for demands without a circulation structure, *i.e.*, Directed Acyclic Graphs (DAGs), balanced routing is impossible to achieve, regardless of the amount of capacity in the channels.

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

We formalize this notion of circulation graphs and show that the maximum achievable throughput by any balanced routing scheme is at most the total demand contained in a maximum circulation graph in Appendix 9.1.

4.3 A Primal-Dual Decomposition Based Approach

We now describe an algorithm based on standard primal-dual decomposition techniques used in utility-maximization-based rate control and routing literature (e.g. [36]). To arrive at this algorithm, we consider the optimization problem described in §4.2 but with $U(x) = x$. This formulation seeks to maximize the total throughput of the network. The structure

of the Lagrangian of the LP allows us to naturally decompose the overall optimization problem into separate subproblems for each sender-receiver pair.

A solution to this LP can be computed via a decentralized algorithm in which each sender maintains rates, instead of windows, at which payments are sent on each of its paths. Each payment channel has a *price* in each direction. Routers locally update these prices depending on both congestion and imbalance across the payment channels, while end-hosts adjust their rates by monitoring the total price on the different paths. The primal variables of the LP represent the rate of payments on each path, and the dual variables represent the channel prices.

While this approach has been used before [36], a key difference from prior work is the presence of price variables for link balance constraints in addition to the price variables for capacity constraints. This ensures that the price of a channel having a skewed balance is different in each direction, and helps steer the flow rates to counter the skew. A detailed derivation of the decentralized algorithm in the fluid model using principles from duality theory is discussed in Appendix 9.2.

4.3.1 Router Design

Routers in each payment channel maintain price variables, which are updated periodically based on the current arrival rate of transaction units in the channel, available channel balance, and the number of transaction units queued up at the routers. The price variables at the routers determine the path prices, which in turn affect the rates at which end-hosts transmit transaction units (we discuss more in §5.2.3).

In a payment channel $(u, v) \in E$, routers u and v hold estimates for three types of price variables: λ_{uv} , μ_{uv} and μ_{vu} . These are dual variables corresponding to the capacity and imbalance constraints in Equation (4.4) and (4.3) respectively. The *capacity price* λ_{uv} signals congestion in the channel if the total rate at which transactions arrive exceeds its capacity; the *congestion prices* μ_{uv} and μ_{vu} are used to signal imbalance in the channel in the two directions. These variables are updated periodically to ensure that the capacity and

imbalance conditions are not violated in the channel. Prices are updated every τ seconds according to the rules described next.

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

$$\mu_{uv}(t+1) = \left[\mu_{uv}(t) + \kappa \left(n_u(t) - n_v(t) + \frac{q_u(t)}{T} \tau - \frac{q_v(t)}{T} \tau \right) \right]_+, \quad (4.6)$$

where κ is a positive step-size parameter for controlling the rate at which the price varies and T is a parameter for controlling the sensitivity of the prices to queue size.¹ Intuitively, if more funds arrive in the u - v direction compared to the v - u direction (i.e., $n_u(t) > n_v(t)$), the price μ_{uv} increases while the price μ_{vu} decreases. The higher price in the u - v direction signals end-hosts that are routing along (u, v) to throttle their rates, and signal end-hosts routing along (v, u) to increase their rates. Similarly, if there is queue buildup on one side due to an imbalance in the rates, Equation (4.6) causes the price to increase in the direction where there is greater queue buildup. This in turn would reduce the arrival rate in that direction, and cause the queue to drain.

Capacity Price. The price variable for capacity λ_{uv} is also updated every τ seconds as follows:

$$\lambda_{uv}(t+1) = [\lambda_{uv}(t) + \eta (m_u(t) + m_v(t) - c_{uv} + \beta \min(q_u(t), q_v(t)))]_+. \quad (4.7)$$

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

¹The price update for μ_{vu} is analogous to Equation (4.6), but with u and v interchanged.

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

4.3.2 Transport Layer Design at End-Hosts

As described in §4.1, Spider-hosts run a multi-path transport protocol with pre-determined paths which controls the rates at which payments are transferred, based on observations of the channel prices or router feedback. End-hosts here use the above probe messages to evaluate the channel prices on each path. The total price of a path p is given by

$$z_p = \sum_{(u,v):(u,v) \in p} (2\lambda_{uv} + \mu_{uv} - \mu_{vu}), \quad (4.8)$$

which captures the aggregate amount of imbalance and excess demand, as signaled by the corresponding price variables, in the path. We refer to Appendix 9.2 for a mathematical intuition behind Equation (4.8). Probes are sent periodically every τ seconds (i.e., the same frequency at which channel prices are updated §5.2.2) on each path. A probe sent out on path p sums the price $2\lambda_{uv} + \mu_{uv} - \mu_{vu}$ of each channel (u, v) it visits, until it reaches the destination host on p . The destination host then transmits the probe back to the sender along the reverse path. The rate to send x_p on each path p is updated using the path price z_p from the most recently received probe as

$$x_p(t+1) = x_p(t) + \alpha(1 - z_p(t)), \quad (4.9)$$

where α is a positive step-size parameter. Thus the rate to send on a path decreases if the path price is high—indicating a large amount of imbalance or capacity deficit in the path—and increases otherwise. This is similar to how the window on a path decreases if a large fraction of packets from the router are marked and increases otherwise.

4.3.3 Challenges

There are a number of challenges in making this algorithm work in practice. Firstly, in order to compute the imbalance prices, the two routers in a payment channel need to exchange information about their arrival patterns and their respective queue states to calculate n_u and m_u in Eq.4.6–4.7. This implies that routers cannot deploy this in isolation. Secondly, we found that the scheme was extremely sensitive to the many parameters involved in the algorithm, making it hard to tune for a variety of topologies and capacity distributions. Lastly, a pure pacing based approach could cause bursts in transaction-units sent that lead to large queue buildups much before the prices react appropriately. To account for this, the algorithm needs to be augmented with windows [35] and active queue control to work in practice. Due to these difficulties, we propose a more practical and simpler protocol described in §5.2.

Chapter 5

A Practical Protocol

In §4.3, we propose an algorithm based on first principles and standard primal-dual decomposition techniques. The algorithm requires coordination between routers on a payment channel to exchange their respective arrival rates and queue sizes, making it hard to deploy in practice. We notice, however, that the routing problem described in §4.2 bears similarities to congestion control for the Internet: users are attempting to maximize their utilities subject to capacity constraints. Solutions for traditional data networks typically involve routers monitoring their own queues and signaling congestion to the end-hosts independently [18, 46, 33]. In this chapter, we ask if similar algorithms could work for PCN routing. We first describe the differences between traditional data networks and PCNs in §5.1 via a motivating example and then use that to propose a concrete protocol in §5.2.

5.1 Intuition Towards a Practical Solution

As mentioned above, the routing problem described in §4.2 is similar to congestion control for the Internet where users are attempting to maximize their utilities subject to capacity constraints. Furthermore, like communication networks, transactions arriving at a router wait in queues at the routers until there is enough capacity to service the transactions. However, unlike communication networks where the capacity or the service rate of routers is exogenous, in payment channels, the “instantaneous capacity” is a direct function of the transaction arrival rate at the routers and the state of the queues themselves. We call

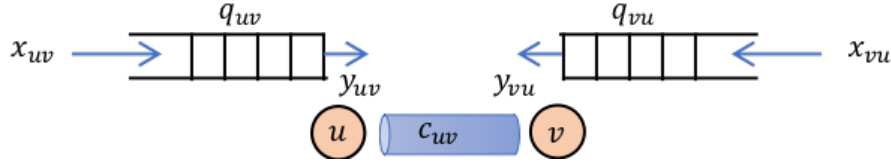


Figure 5-1: Model of queues at a payment channel between nodes u and v . x_{uv} and y_{uv} denote the rates at which transaction-units for v arrive into and get serviced at the queue at u respectively. c_{uv} is the capacity of the payment channel and q_{uv} denotes the total number of transaction-units waiting in u 's queue to be serviced.

this *state-dependent capacity* in contrast to capacity in communication networks which is state-independent.

State-dependent Capacity. To understand how capacity depends on state, consider the model of queues in payment channels described in Fig. 5-1. Transactions arrive at the two ends u and v of the payment channels at rates x_{uv} and x_{vu} . If all tokens c_{uv} in the payment channel were in use and they took Δ seconds to be confirmed and available for use again, the capacity of the payment channel would be $\frac{c_{uv}}{\Delta}$. This represents the aggregate highest service rate that the two queues in the payment channel together can sustain. However, note that tokens at u are only replenished by servicing transaction-units at v . Only then can more transaction-units be serviced at u . In other words, in steady state, if y_{uv} and y_{vu} denoted the rates at which transactions are serviced at queues at u and v respectively, $y_{uv} = y_{vu}$. The capacity constraint (Equation. 4.3) dictates that $y_{uv} + y_{vu} = c_{uv}$.

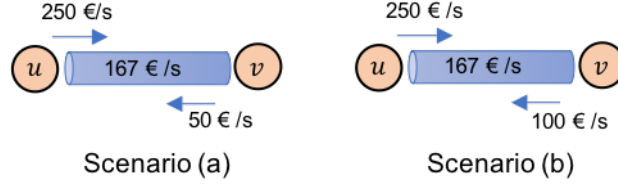
If we consider the scenario where both the queues at u and v are filling up at a rate higher than what the payment channel can support, both queues will be non-zero, but they cannot be serviced at an aggregate rate higher than the capacity of the payment channel. If one end (say u) sends at a rate lower than the other end and one that is lower than half the capacity, there will be no queue buildup at u . However, tokens at v get replenished only at a rate dictated by u 's sending rate. Thus, v cannot service at a rate higher than u 's sending rate. This leads to the following expression that describes the service rates of the two queues in the payment channel.

$$y_{uv} = y_{vu} = \begin{cases} \frac{c_{uv}}{2\Delta} & \text{if } q_{uv} > 0 \ \& \ q_{vu} > 0 \\ \min\{\frac{c_{uv}}{2\Delta}, x_{vu}\} & \text{if } q_{uv} > 0 \ \& \ q_{vu} = 0 \\ \min\{\frac{c_{uv}}{2\Delta}, x_{uv}\} & \text{if } q_{uv} = 0 \ \& \ q_{vu} > 0 \\ \min\{\frac{c_{uv}}{2\Delta}, x_{uv}, x_{vu}\} & \text{if } q_{uv} = 0 \ \& \ q_{vu} = 0 \end{cases} \quad (5.1)$$

Rate Control for State-dependent Capacity. As described by Equation. 5.1, the service rates of queues in payment channels are dependent on the state of the payment channel, namely, its queue size and the transaction arrival rate into the queues. Unlike data networks, they are not time-varying exogenous quantities. Despite this, would traditional congestion control algorithms [18, 46, 33] that control queue sizes still achieve high throughput in a PCN?

To understand what effect controlling queue sizes would have in a PCN, we consider a simple protocol that increases the sending window on a path if the queueing delay through the queue is under 300ms and decreases it otherwise. We observe how such a protocol fares by looking at the long term sending rates and the queue dynamics in a toy example with two nodes attempting to send transactions to each other. We consider two scenarios involving nodes u and v that share a payment channel that can support a maximum aggregate rate of 167€/s: a) one node sends at a rate below half the capacity and the other above (Fig. 5-2b) b) both nodes send at rates above half the capacity of the payment channel (Fig. 5-2c).

In scenario (a), node u wants to send 250€/s to v and v wants to send 50€/s. This is clearly imbalanced and given the service rates described in Equation. 5.1, both nodes cannot service more than 50€/s to each other. Fig. 5-2b shows that under the protocol, the long-term sending rates match the service rates of 50 €/s at either queue making it well-behaved. When router u attempts to send at a rate higher than 50 €/s, its tokens do not get replenished fast enough from v (which only has a demand of 50 €/s and therefore cannot send more tokens). Consequently, a queue builds up at u signaling a reduction in sending rate. Once the queue drains and the queueing delay goes below 300ms, the sending window is permitted to increase in rates and this process repeats. Throughout this, the queue at u is kept at a controlled small size and the average rate at which transaction-units are sent in either direction is 50 €/s (the lower of the two arrival rates). Note that a queue never builds



(a) Two separate scenarios on a toy topology where two connected nodes u and v are generating transaction-units to be sent to each other at the denoted rates. The payment channel can support utmost a total of 167 €/s.

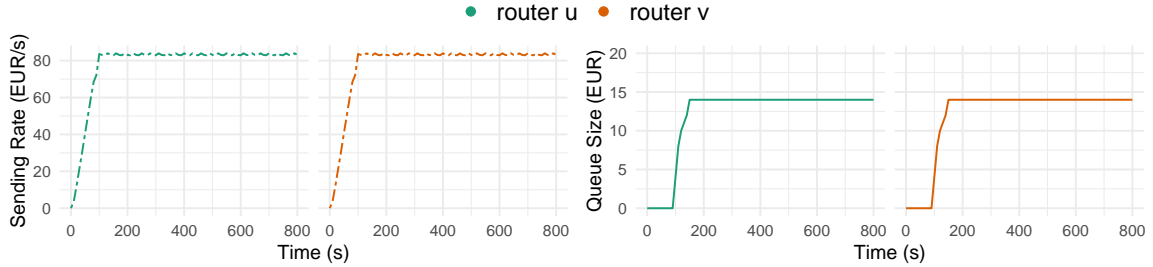
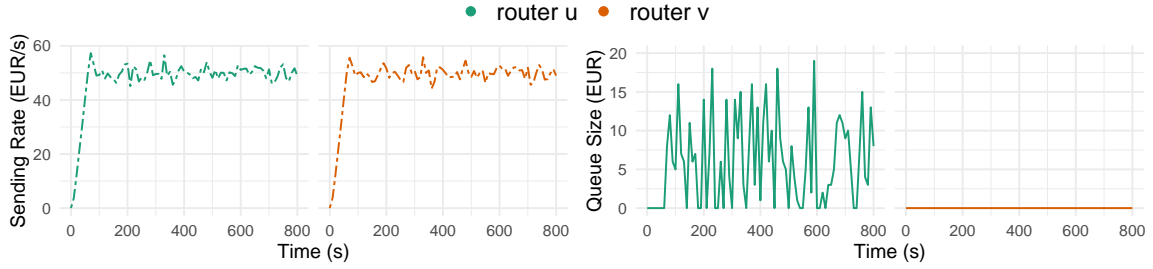


Figure 5-2: Dynamics of running a simple protocol that responds to queue buildup at routers on a toy PCN. When the demand at v is much lesser than both the demand at u and the capacity of the uv payment channel, both nodes' sending rates drop to match the queue service rates dictated by the smaller of the two demands causing a controlled queue at u . When the demand at v is much lesser than the demand at u but is higher than half the capacity of the uv payment channel, the sending and servicing processes become uniform at the same rate (83€/s) keeping both router queues at a constant amount.

at v because u is always able to return any tokens v sends it because of its outstanding higher demand.

In scenario (b), node u wants to send 250€/s to v and v wants to send 100€/s. This is imbalanced, but more importantly, both nodes have a demand that is higher than half the capacity of the payment channel. This means that neither end's demand can be fully satisfied. When the protocol is run, this manifests itself in the form of queue buildup at both ends as shown in Fig. 5-2c. In response to the increase in queueing delay, both nodes reduce their sending rates until the queueing delay is under 300ms and then start increasing them again. This periodic process of reducing and increasing rates in response to queue buildup causes the sending and service rates to stabilize perfectly around some average long-term value. We know from Equation. 5.1 that the queues at u and v converge to the same long-term service rate. Given that in aggregate, the payment channel cannot sustain more than 167 €/s, both nodes roughly converge to a service rate of 83 €/s. Fig. 5-2c shows that once again, the protocol is able to control the sending rates such that it exactly matches this service rate ensuring that the queue size doesn't blow up despite the much higher demand.

In essence, a protocol based on response to queue buildup is able to accurately capture the dynamics induced by the balance and capacity constraints in PCNs. At first glance, it might be surprising that a class of algorithms aimed at optimizing throughput for links with exogenous capacities is also able to achieve optimal throughput with state-dependent capacities. But, the key insight here is that violation of the balance constraint, the main driver behind the state-dependent nature of payment channel capacities, also manifests in queue buildup much like violation of the capacity constraint does. Adjusting the sending rate in response to this queue buildup allows the imbalance to lessen over time enabling the system to stabilize. Violation of capacity constraints results in queue buildup at both nodes in the payment channel, as opposed to one, but a quick reduction in sending rate helps keep the queue size under control and, stabilizes the sending rate to the queue service rates in this case too.

We build on this insight and develop the Spider protocol, a multipath transport protocol that uses feedback on queueing delay from the routers to control window sizes at the end-

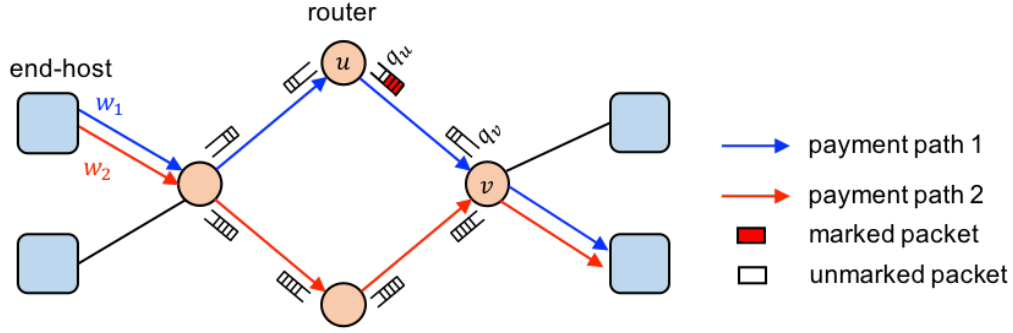


Figure 5-3: Spider Architecture: Routers queue up transaction-units and schedule them across payment channels based on available capacity and transaction priorities. If the delay through the queue for a packet exceeds a threshold, they mark the packet. End-hosts maintain windows for each path to a receiver which are adjusted based on whether acked transaction-units are marked.

hosts. We describe the protocol in detail in §5.2. Fig. 5-3 shows a schematic diagram of the various components in the Spider PCN. In this chapter, §5.2.1, we describe the message formats required for Spider protocol, followed by a discussion of the marking scheme at routers (§5.2.2). and the transport layer design at end-hosts (§5.2.3).

5.2 The Spider Protocol

5.2.1 Message Formats

In order to relay feedback back from the routers to the end-hosts, we introduce a new field in the header of the onion-routed transaction-units that denotes whether the transaction-unit is marked or not. A marked transaction-unit indicates to the sender that the transaction-unit encountered some congestion or imbalance along its route to the receiver. An end-host simply adds an empty unmarked field to the transaction-unit it initiates. Routers along the path to the receiver may change this field to marked. Since this field is modified only in the header and doesn't tamper with the onion routed payload of the packet itself, it does not affect the privacy of the payment being made. When a receiver receives a transaction-unit, the receiver echoes back this marked field in the header of the onion-routed acknowledgment that it sends to the sender. This feedback on the path can then be used by the sender

to make decisions on future payments on the same path.

5.2.2 Spider Router Design

Spider routers queue up transactions at a given payment channel whenever the channel lacks the funds to forward the transaction(s) immediately (§4). Thus, queue buildup is a sign that either transaction-units are arriving faster (in both directions) than the channel can process or that there are insufficient funds at one of the two ends of a payment channel (§5.1). In other words, it suggests that either the capacity constraint (Equation. 4.3) or the balance constraint is being violated (Equation. 4.4) and that the sender should accordingly adjust its sending rate.

Spider routers monitor the time that each packet spends in its queue and mark the packet if the time spent is larger than a pre-determined threshold T . If the transaction-unit is already marked, routers leave the field unchanged and merely forward the transaction-unit. When the end-host sends back an acknowledgment with the “marked” field appropriately set, Spider routers forward it back to the sender which interprets it accordingly.

5.2.3 Spider Transport Layer Design at End-Hosts

Spider-hosts run a multi-path transport protocol which controls the rates at which payments are transferred, based on observations of congestion in the network. For each destination host, a sender chooses a set of k paths along which to route transaction-units. The route a transaction unit takes is decided up front at the sender before transmitting the unit. It is written into the transaction using onion encryption, to hide the full route from intermediate routers and provide privacy [30, 12]. In §6.6, we provide a detailed evaluation of the impact of different path choices on Spider’s performance. Based on these results, we propose using $k = 4$ edge-disjoint widest paths between each sender and receiver in Spider.

In order to control the rate at which payments are sent on a path, end-hosts maintain a window size w_p for every candidate path to every destination. This window size denotes the maximum number of pending payments that can be outstanding on path p at any point in time. End-hosts track the transactions that have been sent out on each path but haven’t

been acked or cancelled yet. A new transaction is transmitted on a path p only if the total amount of pending transactions does not exceed w_p .

End-hosts adjust w_p based on router feedback on congestion and imbalance: the more marked packets they get, the less they should be sending on the path and vice-versa. In particular, on every marked packet (or cancelled packet) between source i and receiver j on path p ,

$$w_p \leftarrow w_p - \beta, \quad (5.2)$$

and on every unmarked packet,

$$w_p \leftarrow w_p + \frac{\alpha}{\sum_{p': p' \in \mathcal{P}_{i,j}} w_{p'}}. \quad (5.3)$$

Here, α and β are both positive constants that denote the aggressiveness with which the window size is increased and decreased respectively. Eq. (5.2)–(5.3) are similar to AIMD schemes for data networks [15, 7]; over an RTT, if a fraction f of transaction-units on a path p are marked, the window size is reduced by $\beta f w_p$ on that path p and increased by $\alpha(1 - f)$ across all paths between source i and receiver j .

Chapter 6

Evaluation

We develop an event-based simulator for PCNs using the OMNET++ framework [1], and use it to extensively evaluate Spider across a wide range of scenarios. We begin this section with a description of our simulator (§6.1), the schemes against which we compare Spider (§6.2), and our experiment setup (§6.3). We then present our results, showing that Spider requires less than 25% of the capacity needed by state-of-the-art approaches to successfully route over 95% of the transactions across a variety of graph topologies for a circulation demand (§6.4). We understand where the benefits come from and what the impact on fairness across users is. We then show the impact of adding DAG components to the circulation payment graphs §6.5. Finally, we study the impact Spider’s design choices (§6.6) in the controlled circulation setting.

6.1 Simulator Implementation

We extend the OMNET++ discrete-event simulator (v5.4.1) [1] to model the flow of transactions along a PCN. Existing PCN simulators [14, 52] adopt a centralized approach to serially schedule transactions on a logical payment channel graph, without accounting for propagation delays or the time that funds in a payment channel are locked while waiting for acknowledgments from the receiver. They, therefore, fail to capture important network-wide properties that influence transaction throughput (e.g., the reduction in channel capacity while a transaction-unit waits for an acknowledgment, congestion due to multiple

outstanding transactions, and so on). Ours is the first PCN simulator that accurately models the network-wide effects of transaction processing, by explicitly passing messages between nodes in a PCN, that are connected to one another via channels with specified link delays and bandwidths (similar to conventional packet simulators [10, 11, 2]).¹

We model the end-to-end transaction processing pipeline for PCNs using three main message types: (i) *transactions* generated at the sender, and source-routed to the receiver via intermediate router nodes. (ii) *acknowledgments* generated by the receiver upon receiving a transaction, and source-routed along the reverse path of the transaction, and (iii) *balance update messages* sent by a node to its neighbor upon receiving an acknowledgment from it.

As discussed in §2, when a transaction is sent along a payment channel from node *A* to node *B*, the funds corresponding to the transaction amount get locked up (i.e. the channel balance at node *A* is decremented without increasing the balance at node *B*). The funds are released (*B*'s balance is incremented) only when *B* forwards that transaction's acknowledgment (propagated from the receiver) to *A* and receives a balance update from it.

As a sender, each endhost (i) generates transactions destined for other endhosts (based on a specified distribution), and (ii) determines when to send a transaction, and along which path, as per the routing schemes described in §6.2. It optionally splits a generated transaction into MTU-sized segments before routing them, if required by the routing scheme (e.g. by Spider). Transactions that are waiting to be sent, are placed in a *per-destination queue* at the sender. The sender also keeps track of which transactions are in-flight and along which path. This state is updated as and when transactions are attempted or acknowledgments are received. When a transaction is generated, it is associated with a *timeout* value. When a timeout is triggered for a waiting transaction or an in-flight transaction, it is treated as a failed transaction, and its state is cleared from all nodes in the network by passing a *clear message* along appropriate routes. Receiving end-hosts simply generate an acknowledgment upon receiving a transaction, that is source-routed along its reverse path. All endhosts maintain a view of the entire PCN topology, to compute suitable source-routes.

¹We intend to make our simulator publicly available.

A router forwards incoming transactions and acknowledgments along the payment channels specified in their source-routes, and sends balance updates as mentioned above. A transaction is forwarded on a payment channel only if the channel has sufficient balance. Otherwise, the transaction is put in a *per-channel queue* at the router, which gets serviced in a FIFO order when balance becomes available. If the queue is full, the incoming transaction is dropped, and a *failure message* is sent to the sender, indicating that the transaction has failed.²

We simulate different topologies of inter-connected routers, as described in §6.3, with an endhost attached to each router via a payment channel whose capacity is set to a very large value. This models realistic scenarios where capacity constraints are observed only on payment channels between two routers and end-hosts fund edge payment channels as necessary to allow transactions through them.

Notice that our simulator only models the flow of transactions (and other messages) across the PCN, and abstracts away its security-related aspects (such as transaction hash and secrets, signatures and onion routing). It, therefore, does not capture any overheads due to cryptographic processing.

6.2 Routing Schemes

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

Spider: We implement Spider as described in §5.2. Every sender maintains a set of k edge-disjoint widest paths to each destination and a window size per path. The window bounds the total number of transaction-units in-flight on that path. End-hosts split all generated transactions into MTU-sized transaction-units before routing them independently on different paths. A given transaction-unit is placed in a FIFO queue of waiting transaction-units to its destination until it can be sent on one of the k paths either due to other transactions completing or a growth in the window size. Spider routers support queues (as described

²As mentioned in §4, we introduce the concept of router queues for Spider. Routers in current PCN implementations [8, 51] do not support queues. They simply drop the transaction and send a failure message when the channel has insufficient balance. This behavior can be modeled in our simulator by setting the router queue size to zero.

in §5.2.2) and monitor the delay through their queues for every transaction-unit that gets queued at them. If this delay exceeds a threshold T , they mark the transaction-unit. If a transaction-unit is marked, the receiving end-host also marks the acknowledgment back to the sender. The sender, then appropriately adjusts the window size for that path as described in §5.2.3.

Waterfilling: In addition to Spider, which is designed using a rigorous LP formulation, we also develop a simpler balance-aware routing scheme based on a *waterfilling* heuristic. As with Spider, each sender splits the transactions into transaction-units and picks k edge-disjoint widest paths to each destination. It maintains one outstanding probe along each of these paths at all times, that computes the bottleneck (minimum) channel balance along the path. The sender then subtracts the amount of in-flight transactions along a path from its bottleneck balance, to obtain the available balance for the path. A transaction-unit for a given destination is then sent along the path with the highest remaining balance among the k choices for that destination. If the remaining balance along all of the k paths is zero (or less), the transaction segment is queued up and retried for transmission the next time a probe comes back. Waterfilling routers also maintain fixed-size router queues to store transaction-units that cannot be serviced immediately, and service them when funds become available.

Shortest Path: We implement this as a naive baseline. The transaction is blindly sent along the shortest path to the destination without any transaction-splitting. Router queues are enabled to store transactions in FIFO order that arrive when channel balance is insufficient. However, as we will see in §6.4, such a balance-agnostic scheme fills up the router queues quickly, experiencing a large number of dropped (failed) transactions.

Landmark Routing: We implement Landmark Routing since it is the building block of many prior schemes for routing ([48], [41], [51]). This scheme chooses k well connected nodes in the topology as landmarks and routes every transaction through them. Every sender computes their shortest path to each landmark and concatenates this with the shortest path from that landmark to the destination to obtain k paths. Then, for every transaction, the sender sends out a probe on each of the k paths to obtain its bottleneck (minimum) channel balance. The sender then uses this information to randomly partition the transaction across

the k paths such that each path can support its share of the total transaction. If such a partition does not exist or if any one of these partitioned transactions fail to complete, the transaction is deemed a failure. Since this scheme does not discuss router queues, we disable them.

LND: To compare Spider against a PCN routing scheme used in practice, we implement the scheme currently deployed in the Lightning Network Daemon (LND) [8]. The scheme ensures that every transaction generated at the sender is sent along the current shortest path based on a local view of the PCN topology. If the transaction fails due to insufficient balance at a particular channel along the path, the sender removes that channel edge from its local view, recomputes the shortest path and retries the transaction on the new path. This process is repeated every time a failure is received due to insufficient balance, until the destination becomes unreachable due to edge removals or the transaction times out. Both cases are treated as failures. Each removed edge is added back to the local view five seconds after its removal. We disable transaction splitting and router queues to implement this scheme as used in practice.

6.3 Experimental Setup

We now describe the transaction workload, PCN topology, simulator parameters and evaluation metrics in more detail.

Workload: To generate the transaction workload, we first generate a payment graph or transaction demand matrix that specifies the rate at which a given sender transacts with every other receiver. Some of these rates might be 0 implying that certain sender-receiver pairs don't transact. We then translate these rates to discrete transactions with a fixed size that arrive according to a poisson process at the desired rate for a sender-receiver pair. The transaction sizes are sampled from credit card transaction data [31]. The distribution of transaction sizes is shown in Fig. 6-1b. The mean transaction size is 88€ with the largest transaction being 3930€. Waterfilling and Spider split all transactions into independent transactions each of size 1€ before routing them appropriately.

We first analyze the performance of different routing schemes with a circulation pay-

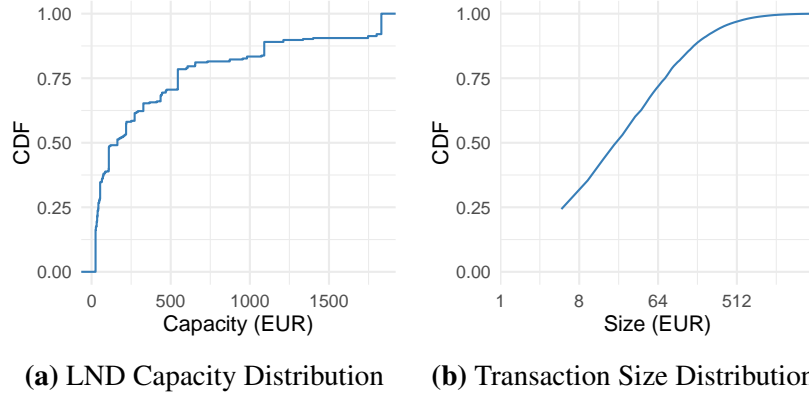


Figure 6-1: Topology and transaction dataset used for real-world evaluations.

ment graph in §6.4 where theoretically, a good routing algorithm can route all of the demand. To generate such a payment graph with a fixed total sending rate x per sender, we create x different permutation matrices and add them together.

To create an entirely DAG workload of a total rate y , we sample y different sender-receiver pairs where the senders and receivers are chosen from two separate skewed exponential distributions. The mean of the distribution is set according to the desired skew. We combine the DAG workload and circulation workload appropriately to achieve a workload that is a combination of both as used in §6.5.

Topology: We run our evaluations on both synthetic and real topologies with two different capacity distributions. The topology and capacity distributions only affect the payment channels between router nodes in our evaluations. We connect a single end host to every one of these router nodes. The end-hosts alone initiate and receive transactions. The payment channel from the end host to the router is made extremely large in order to ensure that it never runs out of capacity and an end-host is always able to send and receive payments. All payment channels are initialized perfectly balanced at the start of the experiment. Throughout the evaluation, we make a distinction between channel sizes and capacities: the former denotes the number of tokens in € in a payment channel while the latter in €/s denotes the maximum rate at which transaction-units can be successfully transmitted across a payment channel.

In order to evaluate Spider on today’s PCNs, we use a subgraph of the Lightning Network as seen by our own LND [8] node on July 15th 2019. We snowball sample[34] from

this graph to obtain a PCN consisting of 106 nodes and 265 payment channels. Payment channel sizes obtained from the Lightning Network are in Satoshis; we converted them into € to use with our transaction data set. Further, since the median transaction size is 25€, we set the bottom 15% of the sampled subgraph’s payment channel sizes to 25€, making the minimum payment channel size 25€. The resulting distribution of sizes (shown in Fig. 6-1a) has a mean and median size of 421€ and 163€ respectively. We refer to this distribution as the Lightning Channel Size Distribution (LCSD) through the rest of this evaluation.

In addition to the Lightning topology, we also experiment with synthetic topologies. We use a Watts-Strogatz small world topology with 50 nodes and 200 edges, and a scale-free Barabasi-Albert graph with 50 nodes and 336 edges.

We vary both the mean and the distribution of channel sizes across all topologies. We first consider the simple case where all payment channels have equal tokens in them. We also evaluate Spider on a second distribution of payment channel sizes that is based on the LCSD: we sample from it for the synthetic topologies and use it directly for the real topology. When varying the mean channel size with the LCSD, we scale up all of the payment channel sizes proportionate to the new desired mean, thereby preserving the skew in the distribution itself.

The above described combination of channel size distributions and topology types results in six different scenarios as shown in §6.4. Since the combination of Lightning topology and LCSD represents the most realistic scenarios, we set the hop delay for the peer-to-peer links in that scenario alone based on real ping times from our LND node to all nodes on the Lightning Network. This means that the average minimum RTT of a transaction is about a second. For all other scenarios, the peer-to-peer link between nodes has a delay of 30 ms, resulting in transaction RTTs of 200-300ms.

Parameters: Since transactions range in size from 5€ to 3930€, we set the MTU as 1€. This allows Waterfilling and Spider to split even the smallest transactions and further allows the minimum window size to be as low as 1€ for Spider. This is important because we want the windows to grow as small as possible if the algorithm dictates that a particular path should not be used at all.

Every transaction has a timeout of 5 seconds associated with it once it is sent out into the network. Thus, 5 seconds after its first attempt, it is considered a failure and all state in the network associated with it is cleared.

Schemes that have queues (Waterfilling, Spider and shortest path) have a queue size of 12000€ per node per payment channel. If the *total value* of all transaction-units queued up at a node for a particular payment channel exceeds 12000€, further transactions arriving for that payment channel at the node are deemed failures immediately. Schemes that don't have queues have a queue size of 0; if a router along the path has no instantaneous balance, it is immediately considered a failure.

All algorithms that use multiple paths (Waterfilling, Landmark Routing and Spider) use 4 paths by default. For Waterfilling and Spider, these paths are edge-disjoint widest paths. We vary both the number of paths and the nature of paths in §6.6 to justify this choice.

Spider has three parameters associated with its updates. α or the window increase factor is set to 10. This means that over one RTT, if no packets are marked, the window will increase by 10 across all paths for a given flow. β or the multiplicative decrease factor is set to 0.1. The marking threshold for the queue delay is set to 300ms.

Metrics: We evaluate different routing schemes based on the following three metrics: (i) transaction success ratio (ii) normalized throughput. *Transaction success ratio* is the number of transactions completed over the number of transactions arrived. A transaction is complete only if all transaction-units that it was split into complete. *Absolute throughput* is the total number of transaction-units that were completed over the measurement interval. *Normalized throughput* is the number of transaction-units that were completed over the total number of transaction-units that arrived in an interval. This accounts for cases where some but not all transaction-units that a transaction was split into completed. All of these metrics were computed across the same measurement interval across all algorithms. The measurement interval was set such that all algorithms had achieved steady state by then. For the experiments detailed in §6.4 and §6.6 which use circulation demands, this interval was set to 800-1000s for experiments that were run for 1000s.

6.4 Performance on Circulation Payment Graphs

As discussed in §4.2.1, no balanced routing scheme can achieve throughput greater than the maximum circulation in the payment graph. Therefore, we begin our evaluation with a traffic matrix representing a circulation payment graph. We know that in these graphs that theoretically, all the demand can be successfully routed if there is sufficient capacity. Thus, the capacity at which a given scheme attains this 100% throughput is a mark of how efficient the scheme is at maintaining balance: the more balanced a scheme is, the lesser capacity it needs for high throughput. Conversely, the more imbalanced a scheme is, it relies more on the capacity of the payment channels to support additional transactions.

We first show how efficient Spider is at routing a circulation traffic demand. Next, we break down the successful transactions by size and lastly discuss how fair Spider is across flows. We run all experiments for 1000s under a traffic demand where every node is sending an average of 30 transactions per second to about 10 different destinations on average. The transactions sizes are sampled from the distribution in Fig. 6-1b and we measure the transaction success ratio for transactions that arrive between 800-1000s. This ensures that we capture the steady state throughput.

Efficiency of Routing Schemes. We run five different circulations with each node sending 30 transactions on average on all three topologies and across both capacity distributions. We vary the average payment channel size on a logarithmic scale and observe the average fraction of transactions successfully routed by each protocol. Fig. 6-2 shows that across all topologies and both channel size distributions Spider outperforms the state-of-the-art schemes. Spider is able to successfully route more than 95% of the transactions with less than 25% of the capacity that LND needs. Further at lower capacities, Spider is able to route upto 30% more transactions successfully when compared to LND. This is because Spider maintains balance in the network by responding quickly to any queue buildup at payment channels, thus allowing it to make better use of the capacity of the network. A consequence of this is that Spider requires an order of magnitude lesser capacity than LND to be able to route more 80% of the transactions on average. While the Lightning Network is still in early stages of deployment, channels are likely to be small in size and a 30%

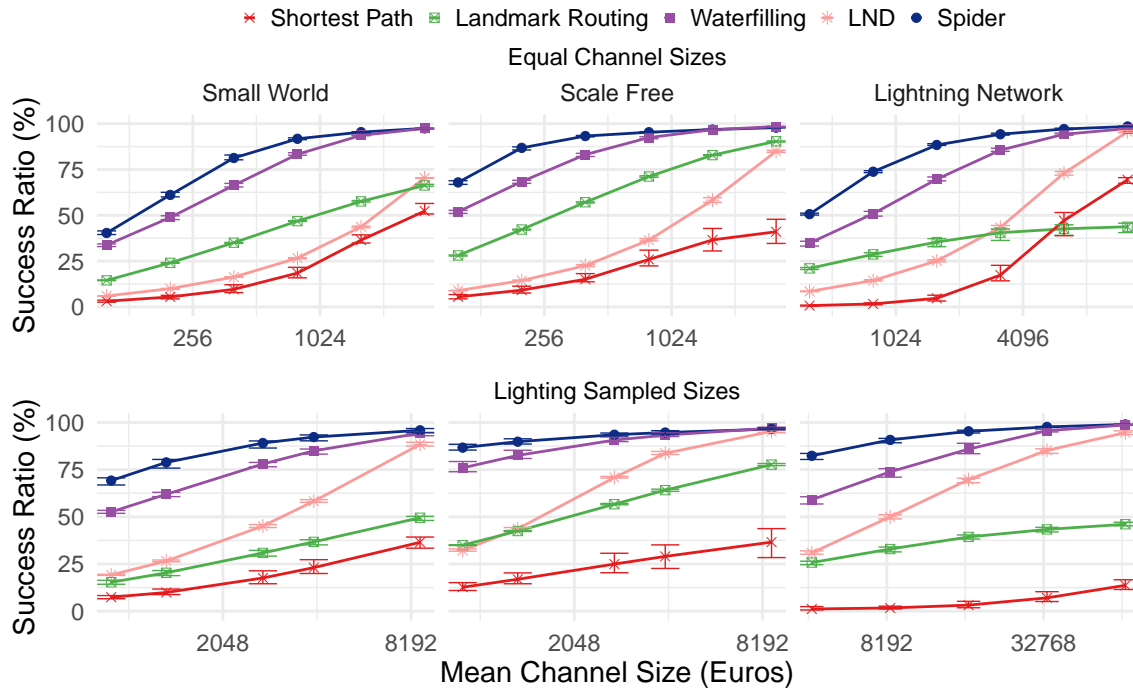


Figure 6-2: Performance of different algorithms on different topologies with different per sender transaction arrival rates. Spider consistently outperforms all other schemes achieving near 100% average success ratio. Error-bars denote the maximum and minimum fraction of successful transactions across five runs with different circulation graphs. Note the log scale of the x-axes.

improvement in success ratios marks a very significant improvement for PCNs. Note that even the simple load-balancing heuristic approach Waterfilling is able to route many more transactions than LND does today.

Size of Successful Payments. Fig. 6-2 suggests that overall, Spider routes more of the demand successfully, it does not capture whether the successful payments are extremely small ones or large transactions included. This is especially relevant because of the extremely skewed nature of the transaction sizes (Fig. 6-1b). To understand this better, we break down the successful transactions by size in Fig. 6-3 for the middle channel size in each of the topology and channel size distribution scenarios described above. Every shaded region denotes a different range of transaction sizes, with the smallest ones being denoted by lighter regions. Each range of transaction sizes roughly corresponds to an equal total number of transactions. Each point represents the fraction of successful transactions in that size interval that were routed successfully over all 5 runs with different circulations. Spider

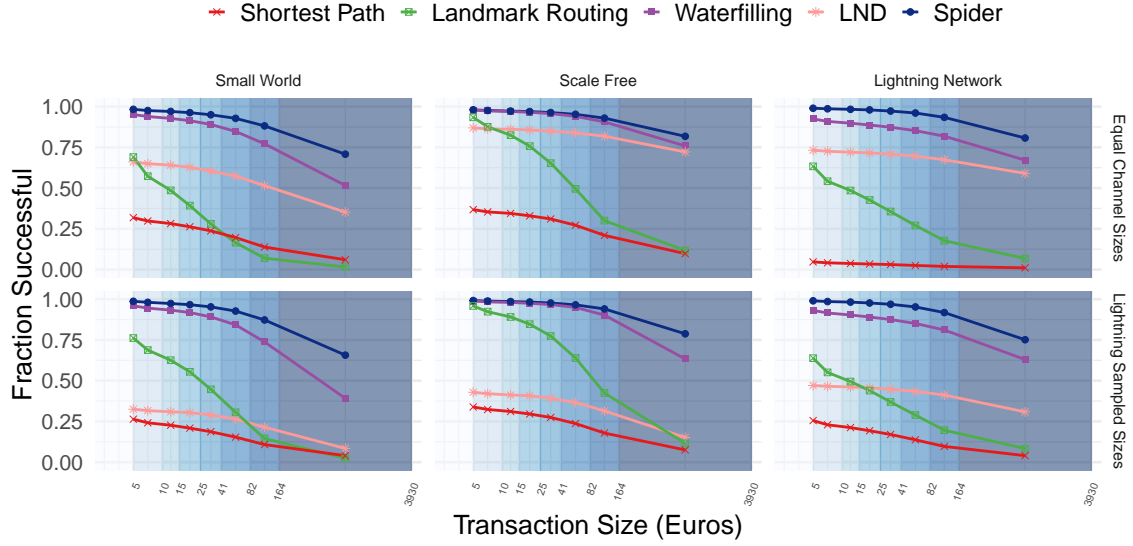


Figure 6-3: Breakdown by size across of performance of different schemes across all topology and capacity distributions. Each point reports the fraction of successful transactions whose size belongs to the interval denoted by the shaded region. Each interval corresponds roughly to 12.5% of the CDF denoted in Fig. 6-1b. The graphs correspond to the (right) midpoints of the corresponding channel sizes in Fig. 6-2

outperforms LND across all sizes on both real and synthetic topologies, and with both channel size distributions. In particular, it is able to route 5-60% more of the largest transactions when compared to LND. Waterfilling is also able to route most of the small transactions as well as Spider does, but Spider’s benefits are most pronounced at larger transaction sizes where pro-active rate control, response to queues and the presence of windows enables more successful transactions.

Fairness. Lastly, we break down the success by flows (sender-receiver pairs) in order to understand whether all pairs of nodes transacting on the PCN experience improvement in throughput or if some of them are starved. Fig. 6-4 shows a CDF of the absolute throughput in €/s achieved by different protocols on a single circulation demand matrix where each sender sends an average of 30 transactions/s on synthetic and real topologies with channel sizes sampled from the LCSD. The mean channel sizes for the synthetic topologies and the real topology are 4000€ and 16880€ respectively. There are two things to note here: (a) Spider achieves close to 100% throughput in all three scenarios, (b) Spider is fairer to small flows (most vertical line) and doesn’t hurt the smallest flows just to benefit on throughput.

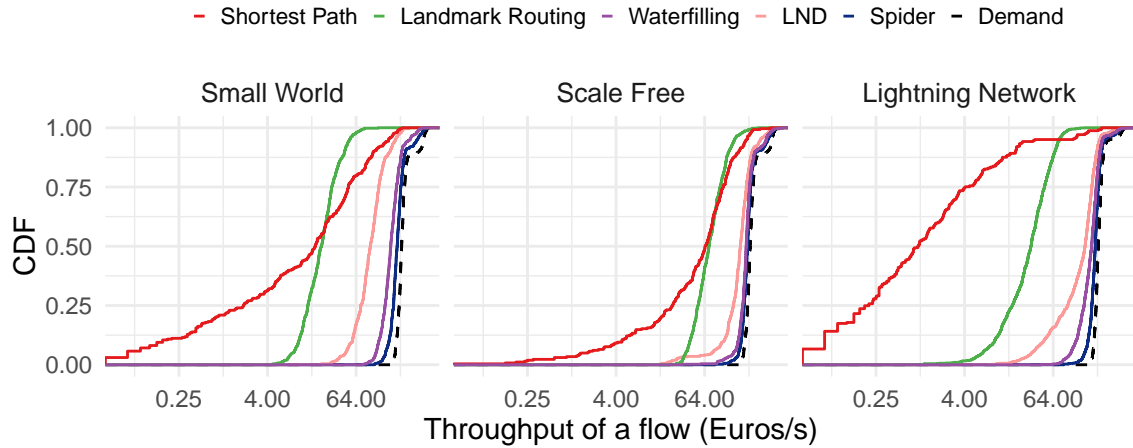


Figure 6-4: CDF of normalized throughput achieved by different flows under different schemes across topologies. Spider achieves close to 100% throughput given its proximity to the black line denoting demand of flows. Spider is a more vertical line than the baseline LND scheme showing that it is fairer than other schemes: it doesn't hurt the throughput of smaller flows to attain good overall throughput.

This is not as true for the baseline LND scheme.

6.5 Effect of Adding DAGs

While §6.4 suggests that Spider outperforms other schemes under a circulation demand, in reality, transaction demands tends to have DAG components in them: certain nodes (typically consumers) are more likely to spend than others and merchant nodes are more likely to receive than others. To simulate this, as mentioned in §6.3, we draw senders and receivers from two independent exponential distributions with controlled skews to generate a fully DAG demand. We repeat this process 5 different times to generate 5 different DAGs. We add this to the circulation demand described above, weighing the two parts appropriately. We vary the skew and the weight to generate effectively 5%, 20% and 40% DAG in the total traffic demand matrix. As described in §4.2.1, the maximum achievable throughput in each of these cases is 95%, 80% and 60% respectively. Note that when transaction sizes are variable the metric to compare to this optimal throughput is the normalized throughput and not the success ratio.

We run the schemes from §6.3 for 1000s on both the synthetic and real topologies when

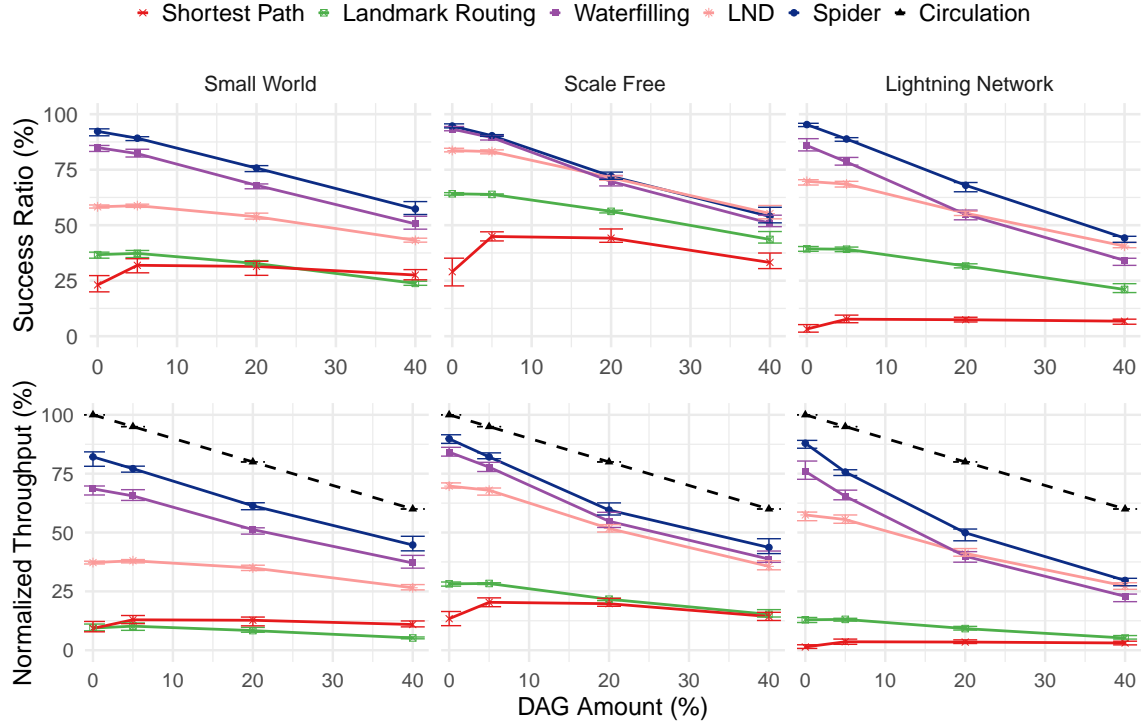


Figure 6-5: Performance of different algorithms across all topologies as the DAG component in the transaction demand matrix is varied. As the DAG amount is increased, the normalized throughput achieved is further away from the expected optimal circulation throughput. The gap is more pronounced on the real topology.

the total transaction arrival rate is 30 transactions/s at every sender. The channel sizes are sampled from LCSD, and the mean channel sizes for the synthetic topologies and the real topology are 4000€ and 16880€ respectively while the transaction sizes are sampled from Fig. 6-1b. As before, we measure the successful transactions between 800-1000s.

Fig. 6-5 shows the success ratio and normalized throughput that the different schemes achieve. We immediately notice that no scheme is able to achieve the maximum throughput. However, the achieved throughput is closer to the maximum when there is a smaller component of DAG in the demand matrix. Together, these suggest that not only is the DAG unroutable itself, it also alters the PCN balances in a way that affects the circulation from going through. Further, the more DAG there is, the more affected the circulation is. One possible explanation for this is that the DAG causes a deadlock §3.

To see if this is indeed the case, we run a more controlled experiment with a circulation demand for 3000s. We remove all other effects by using 1€ transactions arriving in a

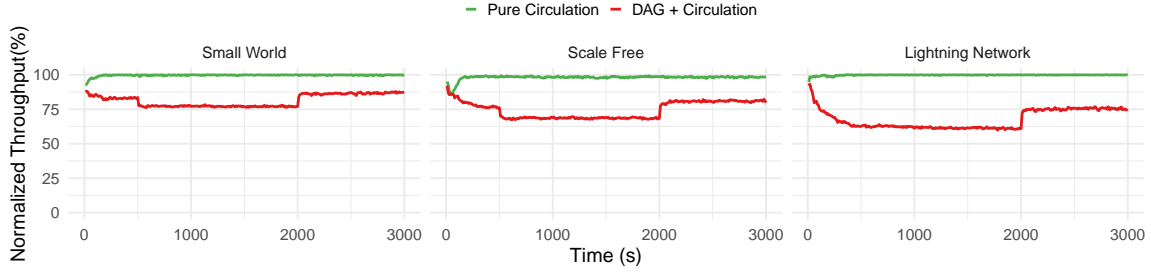


Figure 6-6: Comparing throughput when a pure circulation demand is run for 3000s to a scenario where a circulation demand is restored for 1000s after 2000s of a demand with 20% DAG. The throughput achieved on the last 1000s of circulation is not the expected 100% even after the DAG is removed

poisson manner at a rate of 200 transactions/s. In addition to the circulation workload, we run a second scenario where we add a DAG component such that the DAG comprises 20% of the demand only until 2000s. The last 1000s of this scenario has a pure circulation demand. We run this special workload on the synthetic and real topologies and observe a time-series of the normalized throughput (or success ratio in this case also) over the entire 3000s for both cases. The channel sizes, sampled from the LCSD, have a mean 2750€ per channel for the synthetic topologies and 16880€ for the Lightning Network topology.

Fig. 6-6 shows as expected that there is a brief convergence period when the demand is introduced or modified. However, once steady state is reached, the throughput achieved is the expected 100% for the first scenario of 3000s of pure circulation demand in all three topologies. However, in the second scenario with the dag, it affects the different topologies differently. Since the effective DAG added is 20%, we expect a normalized throughput of 80% for the 0-2000s period. However, this doesn't happen on any of the topologies. This implies that the DAG eats into some of the circulation throughput also. Further, even once the circulation demand is restored for the last 1000s, the throughput achieved is no longer 100%. In fact, the further the achieved throughput in the first 2000s is from the expected throughput, the further it is from the expected 100% in the last 1000s also. This implies that a deadlock occurs in a part of the network due to the presence of the DAG that much like Fig. 3-1c ensures that even the circulation demand cannot get through.

As described in §3, the only solution to this problem involves replenishing funds via an on-chain rebalancing scheme since DAG demands continuously direct money from sources

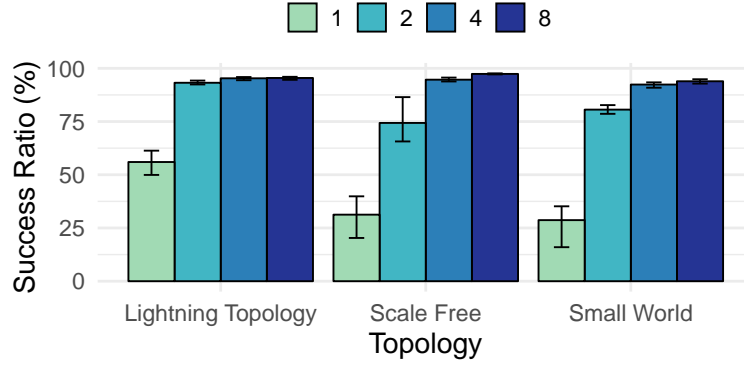


Figure 6-7: Performance of Spider as the number of edge-disjoint shortest paths considered per sender-receiver pair is varied on two different topologies.

to sinks. This one-way flow implies that the edges close to the sources will eventually run out of tokens and need some reinforcement. Designing an efficient rebalancing scheme is beyond the scope of this thesis; future work in this space should consider this a factor for PCN routing.

6.6 Spider’s design choices

We now delve deeper into two of Spider’s design choices in terms of its choice of paths. For this, we evaluate Spider on all three topologies: the LND subgraph with its original payment channel capacities and small world and scale free 50 router graphs with channel sizes sampled from LCSD as described in §6.3. The mean channel sizes for the synthetic topologies and the real topology are 4000 € and 16880€ respectively. Every sender sends out transactions at a rate of 30 transactions/s and the size of transactions is sampled from Fig. 6-1b. We look at two aspects of the choice of paths: the number as well as nature of paths considered.

Number of Paths. We first vary the maximum number of edge-disjoint widest paths Spider allows from 1 to 8. Fig. 6-7 shows that as the number of transactions successfully completed increases as the number of paths is increased. This is intuitive as more paths allow us to better exploit the available capacity in the network. However, while moving from 1 to 2 paths and from 2 to 4 paths gives significant improvements in throughput, the same isn’t true when moving to 8 paths. One reason for this is that all three PCN topologies

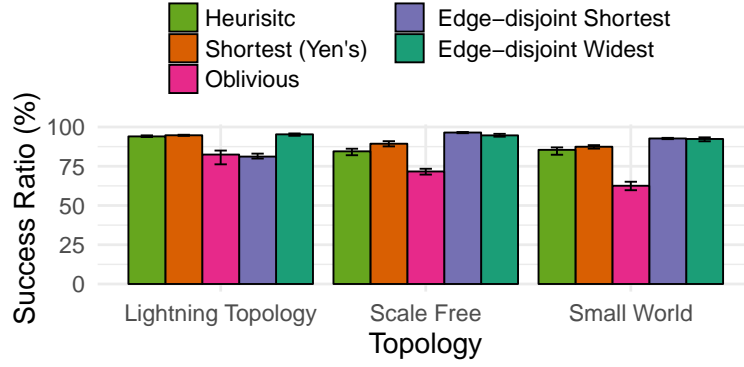


Figure 6-8: Performance of Spider as the type of paths considered per sender-receiver pair is varied.

that we evaluate on are relatively sparse and very few flows have 8 edge-disjoint widest paths even. Most flows have utmost 5 or 6 paths. A second reason is that Spider prefers paths with smaller RTTs since they are able to receive feedback faster for both increases and decreases in window. As more paths become available, some of which might be much longer than others, Spider doesn't exploit the longer paths as much, effectively reducing the working set of paths to much fewer than 8. Thus, the benefit in allowing a maximum of 8 paths isn't as pronounced.

Choice of Paths. We also investigate the role that the type of the chosen paths for Spider plays. In Fig. 6-8, we replace the baseline 4 edge-disjoint widest paths with 4 edge-disjoint shortest paths, Yen's shortest paths [61], and oblivious paths [49]. To compute the widest and oblivious paths, the channel size acts as the edge weight. We also attempt a simple heuristic where we rank all paths between a sender and receiver based on the metric of bottleneck balance/RTT and choose the top 4. The edge-disjoint widest paths perform best on the Lightning Network topology without performing that much worse than any of the other choices on the synthetic topologies. The main reason for its better performance on the Lightning Network is because of its ability to find the largest payment channels. When the channel sizes are as skewed as depicted in Fig. 6-1a, this becomes especially relevant. However, Fig. 6-8 does suggest that there might not be a one-size-fits-all solution with regards to the choice of paths: different path choices might be suitable for different topologies and channel size distributions.

Chapter 7

Related Work

PCN Routing Algorithms. As described in §6.3, in current implementations for the Lightning Network, each node maintains a local view of the network topology and source-routes transactions [47]. Paths are chosen based on shortest path algorithms [8, 3]. A classical alternative solution for this problem is to use a distributed Ford-Fulkerson algorithm [28] to find source-destination paths that support the largest transaction volume for each transaction. The computation complexity of max-flow based algorithms make it impractical for the Lightning Network that has over 5000 nodes and 30,000 channels [9, 5]. Recent proposals have used a modified version of max-flow that differentiates based on the size of transactions [59]. However, inferring the size of payments is hard in an onion-routed network like Lightning.

Two main alternatives to max-flow routing have been proposed: landmark routing and embedding-based routing. In *landmark routing*, select routers (landmarks) store routing tables for the rest of the network, and nodes need only route transactions to a landmark [56]. This approach is used in Flare [48] and SilentWhispers [41, 43]. Spider does not use landmarks, but like SilentWhispers, it splits transactions over multiple paths [41]. Our results (§6.4) show that Spider routes 5-40% more transactions successfully than a landmark routing based approach.

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 [50] and SpeedyMurmurs [51] use embedding-based routing. Computing and updating the embedding dynamically as the topology and link balances change is a primary challenge of these approaches. Prior work [52] shows that embedding-based approaches tend to underperform when compared to landmark routing approaches.

Utility Maximization and Congestion Control. A popular framework for decentralizing transport protocols (for traditional data networks) that optimize a fairness objective is Network Utility Maximization (NUM) [36]. The protocol described in §4.3 follows a very similar approach. NUM uses link “prices”, much like the primal-dual approach, that reflects congestion. These prices are derived based on the solution to the utility maximization problem and senders compute rates based on these router prices at each sender. The protocol typically requires many repeated iterations of the price updates and rate computations to converge to an optimal allocation. As we noticed with the protocol in §4.3, these algorithms can be difficult to tune and stabilize. Variations of NUM attempt to alleviate this by predetermining weights for senders and using rate control mechanisms that target the weight [44].

Amongst the vast literature on congestion control for the Internet, schemes that are most closely related to Spider also try to minimize queue length via explicit feedback from the routers [25, 53, 18]. Most of these schemes use a one-bit ECN mark to indicate congestion to the senders. This is the same approach we adapt for Spider. Most of Internet congestion control focuses on links with fairly stable link capacities. The main contribution of this thesis is that the same family of algorithms works for state-dependent capacities where link capacities are a function of the sending rate and queue sizes themselves.

Blockchain Scalability Solutions. A number of alternate consensus protocols have been proposed over the last few years that enable faster blockchain transactions [27, 29, 19]. Payment channels have become popular as layer 2 solutions that work independent of the underlying consensus algorithms. There have been a number of implementations of payment channels themselves that differ primarily in their use of layer 1 as a punishment mechanism or arbiter [24, 47, 40, 32]. Spider can be modified to suit any of these implementations or consensus algorithms. Channel factories [22] and rebalancing techniques [38] have been

proposed to further reduce layer 1 use, by making it easier to add more funds to payment channels. We view Spider and such techniques as complementary that together improve blockchain scalability.

Chapter 8

Discussion

This thesis motivates efficient routing on PCNs in order to make the most use of the collateral or the tokens in the payment channels and discusses how far we can get on a circulation demand matrix with *balanced routing algorithms*. However, as shown in §6.5, if there is a portion of the demand matrix that is not a circulation, there can be significant degradation to throughput. That begs the question: what can we do to route the remaining DAG portion and to avoid deadlocks in the presence of DAGs? This is especially important given that in practice, most natural demands are DAG-like in nature; rarely do we receive money from the merchants we pay to.

DAG demands need some external rebalancing on-chain in order to be satisfiable since they continuously drain funds from the sources of the DAGs. Rebalancing transactions remove funds from one payment channel on a node and move it to another payment channel on the same node [38]. Alternatively, one might choose to add new funds to an existing payment channel or open a new payment altogether. While this might be useful in some cases, depending on how long it takes to rebalance and how usable the payment channel is while rebalancing and so on, this may or may not be an easy resort. This in turn depends on what Layer 1 (underlying consensus mechanism) the payment channel operates on top of. One can explicitly model rebalancing and its costs, and account for it along with the routing decisions that a node makes in order to ensure high throughput. We leave this to future work.

The protocol presented in §5.2 hinges upon router nodes accurately marking transaction-

units based on queueing delay. This means that sometimes routers are going to actively ask for a reduction in transaction-units sent via them implying a loss of revenue for themselves. It would be interesting to see how the routers financial incentives line up with the network's goal of maximizing throughput and efficiency, and whether there exist any reasons for routers to deviate from the protocol.

While this protocol was designed with blockchains in mind, its use cases extend beyond that. Any payment network that needs to maintain roughly balanced payments for faster settlements would be able to use this algorithm. In particular, we believe it is also applicable to international transfer mechanisms at big banks including Swift [16] and newer technologies like Interledger [54]. It would be interesting to study its impact/usefulness in such a scenario.

Chapter 9

Conclusion

This thesis discusses the challenges in using PCNs effectively to truly enable faster transactions than today's blockchains offer. In particular, we discuss the shortcomings of existing routing solutions, namely their circuit-switched nature and the lack of balance-aware routing that make it hard to use PCNs sustainably. We propose a packet-switched architecture for PCNs that splits transactions into smaller units called transaction-units that are easier to route and load-balance over. We introduced the notion of queues at router payment channels that improve network efficiency. We then use a fluid-model to understand the balance and capacity constraints for PCN routing which leads us to a protocol that achieves high throughput for PCNs. We draw from years of research on congestion control for traditional data networks to develop a multipath transport protocol called Spider that responds to queue buildup at payment channels. We show how queue buildup captures both the need for balance and capacity via a teaching example and evaluations on a PCN simulator. Our scheme outperforms the state-of-the-art routing schemes across a wide range of topologies and payment channel sizes.

Bibliography

- [1] <http://omnetpp.org/>.
- [2] <https://inet.omnetpp.org>.
- [3] Amount-independent payment routing in Lightning Networks. <https://medium.com/coinmonks/amount-independent-payment-routing-in-lightning-networks-6409201ff5ed>.
- [4] AMP: Atomic Multi-Path Payments over Lightning. <https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html>.
- [5] Blockchain caffe. <https://blockchaincaffe.org/map/>.
- [6] c-lightning: A specification compliant Lightning Network implementation in C. <https://github.com/ElementsProject/lightning>.
- [7] HighSpeed TCP for Large Congestion Windows. <https://tools.ietf.org/html/rfc3649>.
- [8] Lightning Network Daemon. <https://github.com/lightningnetwork/lnd>.
- [9] Lightning Network Search and Analysis Engine. <https://1ml.com>.
- [10] NS-2. <http://www.isi.edu/nsnam/ns/>.
- [11] NS-3. <http://www.nsnam.org>.

- [12] Onion Routed Micropayments for the Lightning Network. <https://github.com/lightningnetwork/lightning-onion>.
- [13] Raiden network. <https://raiden.network/>.
- [14] SpeedyMurmurs Software. <https://crysp.uwaterloo.ca/software/speedymurmurs/>.
- [15] The NewReno Modification to TCP's Fast Recovery Algorithm. <https://tools.ietf.org/html/rfc6582>.
- [16] The Society for Worldwide Interbank Financial Telecommunication. <https://www.swift.com/>.
- [17] Why south korea is ?crypto crazy? and what that means for the rest of the world, 2018.
- [18] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). *ACM SIGCOMM computer communication review*, 41(4):63–74, 2011.
- [19] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath. Deconstructing the blockchain to approach physical limits. *arXiv preprint arXiv:1810.08092*, 2018.
- [20] S. Banerjee, R. Johari, and C. Riquelme. Pricing in ride-sharing platforms: A queueing-theoretic approach. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, pages 639–639. ACM, 2015.
- [21] S. Banerjee, R. Johari, and C. Riquelme. Dynamic pricing in ridesharing platforms. *ACM SIGecom Exchanges*, 15(1):65–70, 2016.
- [22] C. Burchert, C. Decker, and R. Wattenhofer. Scalable funding of bitcoin micropayment channel networks. *Royal Society open science*, 5(8):180089, 2018.
- [23] C. N. Cordi. *Simulating high-throughput cryptocurrency payment channel networks*. PhD thesis, 2017.

- [24] C. Decker and R. Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*, pages 3–18. Springer, 2015.
- [25] N. Dukkupati. *Rate Control Protocol (RCP): Congestion control to make flows complete quickly*. Citeseer, 2008.
- [26] 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.
- [27] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59, 2016.
- [28] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956.
- [29] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.
- [30] D. Goldschlag, M. Reed, and P. Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.
- [31] U. M. L. Group. Credit card fraud detection, 2018. <https://www.kaggle.com/mlg-ulb/creditcardfraud>.
- [32] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais. Sok: Off the chain transactions. *IACR Cryptology ePrint Archive*, 2019:360, 2019.
- [33] C. V. Hollot, V. Misra, D. Towsley, and W.-B. Gong. On designing improved controllers for aqm routers supporting tcp flows. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of*

the IEEE Computer and Communications Society (Cat. No. 01CH37213), volume 3, pages 1726–1734. IEEE, 2001.

- [34] P. Hu and W. C. Lau. A survey and taxonomy of graph sampling. *arXiv preprint arXiv:1308.5865*, 2013.
- [35] V. Jacobson and M. J. Karels. Congestion avoidance and control. In *SIGCOMM 1988*, Stanford, CA, Aug. 1988.
- [36] F. Kelly and T. Voice. Stability of end-to-end algorithms for joint routing and rate control. *ACM SIGCOMM Computer Communication Review*, 35(2):5–12, 2005.
- [37] F. P. Kelly, A. K. Maulloo, and D. K. Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, 49(3):237–252, 1998.
- [38] R. Khalil and A. Gervais. Revive: Rebalancing off-blockchain payment networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 439–453. ACM, 2017.
- [39] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff. A brief history of the internet. *SIGCOMM Comput. Commun. Rev.*, 39(5):22–31, Oct. 2009.
- [40] T. Lundqvist, A. De Blanche, and H. R. H. Andersson. Thing-to-thing electricity micro payments using blockchain technology. In *2017 Global Internet of Things Summit (GloTS)*, pages 1–6. IEEE, 2017.
- [41] 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.
- [42] R. McManus. Blockchain speeds & the scalability debate. *Blockspain*, February 2018.

- [43] P. Moreno-Sanchez, A. Kate, M. Maffei, and K. Pecina. Privacy preserving payments in credit networks. In *Network and Distributed Security Symposium*, 2015.
- [44] K. Nagaraj, D. Bharadia, H. Mao, S. Chinchali, M. Alizadeh, and S. Katti. Numfabric: Fast and flexible bandwidth allocation in datacenters. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 188–201. ACM, 2016.
- [45] D. P. Palomar and M. Chiang. A tutorial on decomposition methods for network utility maximization. *IEEE Journal on Selected Areas in Communications*, 24(8):1439–1451, 2006.
- [46] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. Pie: A lightweight control scheme to address the bufferbloat problem. In *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*, pages 148–155. IEEE, 2013.
- [47] J. Poon and T. Dryja. The Bitcoin Lightning Network: Scalable Off-chain Instant Payments. *draft version 0.5*, 9:14, 2016.
- [48] 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.
- [49] H. Racke. Minimizing congestion in general networks. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 43–52. IEEE, 2002.
- [50] 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.
- [51] 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.

- [52] V. Sivaraman, S. B. Venkatakrisnan, M. Alizadeh, G. Fanti, and P. Viswanath. Routing cryptocurrency with the spider network. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, HotNets '18, pages 29–35, New York, NY, USA, 2018. ACM.
- [53] C.-H. Tai, J. Zhu, and N. Dukkupati. Making large scale deployment of rcp practical for real networks. In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications*, pages 2180–2188. IEEE, 2008.
- [54] S. Thomas and E. Schwartz. A protocol for interledger payments. URL <https://interledger.org/interledger.pdf>, 2015.
- [55] 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/>.
- [56] 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.
- [57] N. Varshney. Lightning network has 1% success rate with transactions larger than \$200, controversial research says. HardFork, June 2018.
- [58] Visa. Visa acceptance for retailers. <https://usa.visa.com/run-your-business/small-business-tools/retail.html>.
- [59] P. Wang, H. Xu, X. Jin, and T. Wang. Flash: efficient dynamic routing for offchain networks. *arXiv preprint arXiv:1902.05260*, 2019.
- [60] 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.

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

Appendix

9.1 Throughput Bounds

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

Proposition 1. *Consider a payment graph H with a maximum circulation graph C^* . Let $\nu(C^*)$ denote the total demand in C^* . Then, on a network in which each payment channel has at least $\nu(C^*)$ units of escrowed funds, there exists a balanced routing scheme that can*

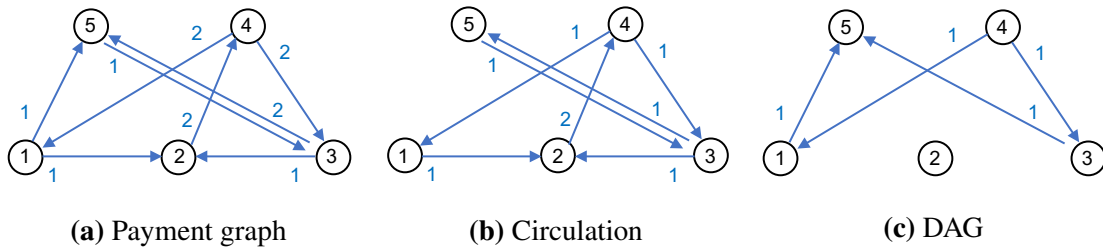


Figure 9-1: Example payment graph (denoted by blue lines) for a five node network (left). It decomposes into a maximum circulation and DAG components as shown in (b) and (c).

achieve a total throughput of $\nu(C^*)$. However, no balanced routing scheme can achieve a throughput greater than $\nu(C^*)$ on any network.

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

Next, to see that no balanced routing scheme can achieve a throughput greater than $\nu(C^*)$, assume the contrary and suppose there exists a balanced routing scheme SCH with a throughput greater than $\nu(C^*)$. Let $H_{\text{SCH}} \subseteq H$ be a payment graph where the edges represent the portion of demand that is actually routed in SCH. Since $\nu(H_{\text{SCH}}) > \nu(C^*)$, H_{SCH} is not a circulation and there exists a partition $S, V \setminus S$ such that the net flow from S to $V \setminus S$ is strictly greater than the net flow from $V \setminus S$ to S in H_{SCH} . However, the net flows routed by SCH across the same partition $S, V \setminus S$ in G are balanced (by assumption) resulting in a contradiction. Thus we conclude there does not exist any balanced routing scheme that can achieve a throughput greater than $\nu(C^*)$.

□

9.2 Primal-Dual Algorithm Derivation

In this section, we present a formal derivation of the decentralized algorithm for computing the optimum solution of the fluid-model optimization problem (Eq. (4.1)–(4.5)) with

$U(x) = x$. Consider the partial Lagrangian of the LP:

$$\begin{aligned}
L(\mathbf{x}, \lambda, \mu) = & \sum_{i,j \in V} \sum_{p \in \mathcal{P}_{i,j}} x_p \\
& - \sum_{(u,v) \in E} \lambda_{uv} \left[\sum_{\substack{p \in \mathcal{P}: \\ (u,v) \in p}} x_p + \sum_{\substack{p' \in \mathcal{P}: \\ (v,u) \in p'}} x_{p'} - \frac{c_{u,v}}{\Delta} \right] \\
& - \sum_{(u,v) \in E} \mu_{uv} \left[\sum_{\substack{p \in \mathcal{P}: \\ (u,v) \in p}} x_p - \sum_{\substack{p' \in \mathcal{P}: \\ (v,u) \in p'}} x_{p'} \right] \\
& - \sum_{(u,v) \in E} \mu_{vu} \left[\sum_{\substack{p \in \mathcal{P}: \\ (v,u) \in p}} x_p - \sum_{\substack{p' \in \mathcal{P}: \\ (u,v) \in p'}} x_{p'} \right], \tag{9.1}
\end{aligned}$$

where μ_{uv}, μ_{vu} are Lagrange variables corresponding to the imbalance constraint (Eq. (4.4)) when written out separately in the u - v and v - u directions respectively. λ_{uv} is a Lagrange variable corresponding to the capacity constraint (Eq (4.3)). Since the λ variable does not have a direction associated with it, to simplify notation we use λ_{vu} and λ_{uv} interchangeably to denote λ_{uv} for channel $(u, v) \in E$. The partial Lagrangian can be rewritten as

$$\begin{aligned}
L(\mathbf{x}, \lambda, \mu) = & \sum_{i,j \in V} \sum_{p \in \mathcal{P}_{i,j}} x_p \left(1 - \sum_{(u,v) \in p} \lambda_{uv} - \sum_{(u,v) \in p} \mu_{uv} \right. \\
& \left. + \sum_{(v,u) \in p} \mu_{vu} \right) + \sum_{(u,v) \in E} \lambda_{uv} \frac{c_{uv}}{\Delta}. \tag{9.2}
\end{aligned}$$

Define $z_{uv} = \lambda_{uv} + \mu_{uv} - \mu_{vu}$ to denote the price of channel $(u, v) \in E$ in the u - v direction, and $z_p = \sum_{(u,v): (u,v) \in p} (\lambda_{uv} + \mu_{uv} - \mu_{(v,u)})$ to be the total price of a path p . The partial Lagrangian above decomposes into separate terms for rate variables for each source/destination pair $\{x_p : p \in \mathcal{P}\}$. This suggests the following iterative primal-dual algorithm for solving the LP:

- **Primal step.** Supposing the path price of a path p at time t is $z_p(t)$. Then, each sender-receiver pair (i, j) updates its rates x_p on each path $p \in \mathcal{P}_{i,j}$ as

$$x_p(t+1) = x_p(t) + \alpha(1 - z_p(t)) \quad (9.3)$$

$$x_p(t+1) = \text{Proj}_{\chi_{i,j}}(x_p(t+1)), \quad (9.4)$$

where Proj is a projection operation on to the convex set $\{x_p : \sum_{p:p \in \mathcal{P}_{i,j}} x_p \leq d_{i,j}, x_p \geq 0 \ \forall p\}$, to ensure the rates are feasible.

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

$$w_{uv}(t) = \sum_{\substack{p \in \mathcal{P}: \\ (u,v) \in p}} x_p(t) + \sum_{\substack{p' \in \mathcal{P}: \\ (v,u) \in p'}} x_{p'}(t) - \frac{c_{uv}}{\Delta} \quad (9.5)$$

$$y_{uv}(t) = \sum_{\substack{p \in \mathcal{P}: \\ (u,v) \in p}} x_p(t) - \sum_{\substack{p' \in \mathcal{P}: \\ (v,u) \in p'}} x_{p'}(t) \quad (9.6)$$

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

$$\lambda_{uv}(t+1) = [\lambda_{uv}(t) + \eta w_{uv}(t)]_+ \quad (9.7)$$

$$\mu_{uv}(t+1) = [\mu_{uv}(t) + \kappa y_{uv}(t)]_+ \quad (9.8)$$

$$\mu_{vu}(t+1) = [\mu_{vu}(t) - \kappa y_{uv}(t)]_+. \quad (9.9)$$

The parameters α, η, κ are positive "step size" constants, that determine the rate at which the algorithm converges. Using standard arguments we can show that for small enough step sizes, the algorithm would converge to the optimal solution of the LP in Eq. (4.1)–(4.5).

The algorithm has the following intuitive interpretation. λ_{uv} and μ_{uv}, μ_{vu} are prices that vary due to capacity constraints and imbalance at the payment channels. In Eq. (9.7), λ_{uv} would increase if the total rate on channel (u, v) (in both directions) exceeds its capacity,

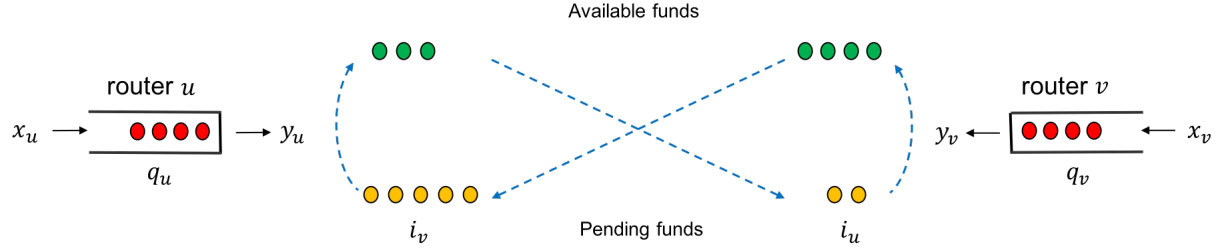


Figure 9-2: Update figure. Routers queue transaction units and schedule them across the payment channel based on available capacity and transaction priorities. Funds received on a payment channel remain in a pending state until the final receiver provides the key for the hashlock.

and would decrease to 0 if there is excess capacity. Similarly, μ_{uv} would increase (resp. decrease) if the net rate in the u - v direction is greater (resp. less) than the net rate in the v - u direction (Eq. (9.8), (9.9)). As the prices vary, an end-host with a flow on path p would react according to Eq. (9.3) by increasing its sending rate x_p if the total price of the path p is cheap, and decreasing the rate otherwise. The net effect is the convergence of the rate and price variables to values such that the overall throughput of the network is maximized. We remark that the objective of our optimization problem in Eq. (4.1) can be modified to also ensure fairness in routing, by associating an appropriate utility function with each sender-receiver pair [37]. A decentralized algorithm for such a case may be derived analogously as our proposed solution.

9.3 Estimating the Demand-Capacity Gap at the Routers

In this section, we explain how this algorithm estimates the total amount of demand on a channel at any time, for updating the capacity price λ in Eq. (4.7). From the description of the primal-dual algorithm for the fluid model in Appendix 9.2, we see that updating λ_{uv} at a channel $(u, v) \in E$ requires estimating

$$\sum_{\substack{p \in \mathcal{P}: \\ (u,v) \in p}} x_p(t) + \sum_{\substack{p' \in \mathcal{P}: \\ (v,u) \in p'}} x_{p'}(t) - \frac{c_{uv}}{\Delta} \quad (9.10)$$

at the channel (Eq. (9.7)). While the total rate at which transactions are arriving at u ($\sum_{p \in \mathcal{P}: (u,v) \in p} x_p(t)$) and at v ($\sum_{p' \in \mathcal{P}: (v,u) \in p'} x_{p'}(t)$) are straightforward to estimate, estimating Δ —the average time taken for transactions to reach their destination from the channel, and for their hashlock keys to arrive at the channel—is difficult. We overcome this problem by estimating the quantity

$$\sum_{\substack{p \in \mathcal{P}: \\ (u,v) \in p}} x_p(t) \Delta + \sum_{\substack{p' \in \mathcal{P}: \\ (v,u) \in p'}} x_{p'}(t) \Delta - c_{uv}, \quad (9.11)$$

instead of trying to estimate the expression in Eq. (9.10). Eq. (9.11) is simply a scaling of Eq. (9.10), but can be estimated without having to first estimate Δ . To see this, let $\tilde{x}_u(t) = \sum_{p \in \mathcal{P}: (u,v) \in p} x_p(t)$ and $\tilde{x}_v(t) = \sum_{p' \in \mathcal{P}: (v,u) \in p'} x_{p'}(t)$ denote the rate of transaction arrival at u and v respectively. Similarly, let $\tilde{y}_u(t)$ and $\tilde{y}_v(t)$ be the rate at which transactions are serviced from the queue at each of the routers (see Fig. 9-2 for an illustration). Eq. (9.11) can now be rewritten as $\tilde{x}_u(t) \Delta + \tilde{x}_v(t) \Delta - c_{uv}$

$$= \left(\frac{\tilde{x}_u(t)}{\tilde{y}_u(t)} \right) \tilde{y}_u(t) \Delta + \left(\frac{\tilde{x}_v(t)}{\tilde{y}_v(t)} \right) \tilde{y}_v(t) \Delta - c_{uv} \quad (9.12)$$

$$= \left(\frac{\tilde{x}_u(t)}{\tilde{y}_u(t)} \right) i_u(t) + \left(\frac{\tilde{x}_v(t)}{\tilde{y}_v(t)} \right) i_v(t) - c_{uv}, \quad (9.13)$$

where $i_u(t)$ and $i_v(t)$ are the amount of funds that are currently locked at routers v and u respectively (Fig. 9-2). Since the funds used when servicing transactions at router u require Δ seconds on average to become available at v , by Little's law the product of the average service rate $\tilde{y}_u(t)$ and average delay Δ is equal to the average amount of pending transactions $i_u(t)$ at v . Thus, Eq. (9.13) follows from Eq. (9.12). However, each of the terms in Eq. (9.13)—the transaction arrival rates $\tilde{x}_u(t), \tilde{x}_v(t)$, service rates $\tilde{y}_u(t), \tilde{y}_v(t)$, amount of pending transactions $i_u(t), i_v(t)$ —can now be readily estimated at the channel.

Intuitively, since $i_u(t)$ is the amount of pending funds at router v when transactions are being serviced at a rate $\tilde{y}_u(t)$, $\tilde{x}_u(t) i_u(t) / \tilde{y}_u(t)$ is an estimate of the amount of transactions that will be pending if transactions were serviced at a rate $\tilde{x}_u(t)$. As the total amount of pending transactions in the channel cannot exceed the total amount of funds escrowed c_{uv} ,

the difference $\tilde{x}_u(t)i_u(t)/\tilde{y}_u(t) + \tilde{x}_v(t)i_v(t)/\tilde{y}_v(t) - c_{uv}$ is exactly the additional amount of funds required in the channel to support the current rates of transaction arrival. Denoting $\tilde{x}_u(t)i_u(t)/\tilde{y}_u(t)$ as $m_u(t)$ and $\tilde{x}_v(t)i_v(t)/\tilde{y}_v(t)$ as $m_v(t)$, the equation for updating λ at the routers can be written as

$$\begin{aligned} \lambda_{uv}(t+1) = [\lambda_{uv}(t) + \eta (m_u(t) + m_v(t) - c_{uv} \\ + \beta \min(q_u(t), q_v(t)))]_+, \end{aligned} \quad (9.14)$$

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