Evaluating Bitcoin's Lightning Network: The impact of Atomic Multi-path Payments & Just In Time Routing

Fergal O'Connor

CK401 Final Year Project Report
Supervisor: Prof. Ken Brown

Department of Computer Science
University College Cork, Ireland

April 2020

Abstract. Bitcoin, in its current state cannot scale, limited to just seven transactions per second. The purpose of this project is to evaluate Bitcoin's Lightning Network (LN), an off-chain solution intended to overcome this scalability issue. The solution is composed of a mechanism known as payment channels that allow two mutually distrustful parties to transact almost instantaneously. These channels may be linked together to form a network of payment channels, allowing parties who do not share a direct payment channel to transact in a trust-less manner.
In this work, we developed a simulator of the network to investigate how the usage of two community-proposed extensions of LN, Atomic Multi-path Payments and Just In Time Routing affects the success rate of payments on this network. In addition, various network conditions are considered to evaluate where the network can scale and to identify what tipping points cause the system to start to break down. Various search strategies have also been compared to investigate the types of paths that are more likely to cause payment success or failure.
Our findings support the adoption of both Atomic Multi-path Payments and Just In Time Routing. We also have found that the network can and cannot scale in certain areas. The network performs well when concurrently routing a large number of payments, but the routing of large payment amounts is problematic. We also note that a more balanced ratio of payment start-points and end-points leads to a higher success rate. In general, routing by the shortest path has been observed to yield a notably higher success rate than other search strategies.

Keywords: Bitcoin · Blockchain · Payment channel networks

1 Introduction

Bitcoin is a digital currency built on a peer-to-peer, decentralised network [14]. It serves as a public payment system in which anyone can join and issue transactions to a distributed ledger known as a blockchain. A key attribute of a blockchain is that there is no central governing authority involved, which means transactions are validated by global consensus of the network participants. This greatly restricts its scalability in terms of transaction throughput which in turn impairs its adoption as a usable payment system. On average, Bitcoin processes seven transactions per second [7] - by contrast, the widely used Visa payment system has been observed to have handled a peak of 47,000 transactions per second in 2013 [26].

In attempts to alleviate this scalability issue, secondary protocols have been designed that are built on top of the original blockchain layer. These are known as Layer 2 protocols [25], [21], [13], [8] and are typically developed to improve its transaction rate. Fundamentally, they allow parties to transact locally "off-chain" without consensus, and employ the blockchain as a method to settle disputes where necessary. Perhaps the most popular Layer 2 solution developed today are payment channel networks, which is the technology that powers the Lightning Network (LN) [21]. A payment channel essentially allows two distrustful parties to transact almost instantaneously and these channels may be tied together to form a payment channel network (PCN) where payments may be routed across the network through intermediary nodes for a nominal fee. In the case of LN, the channels are bidirectional.

Additionally, a substantial fee must be paid to confirm transactions on the blockchain. The confirmation of transactions is performed by miners[1] who in exchange for their resource-intensive services charge this considerable fee. Moreover, the on-chain fee rates do not depend on the transaction amount. This restricts Bitcoin's usability as a micropayment system, remaining more suited to larger transaction amounts. Currently, the fee rates on LN are of stark contrast to its on-chain counterpart. The rates are so negligible that it has been concluded that participation in this network for monetary gain is irrational [3]. This fact supports LN's application as a platform for micropayments.

Yet, the Lightning Network is not without its faults. While ever-improving, it still has flaws to overcome before wide spread adoption. Perhaps the most prominent issue it faces is its routing problem [22]. This is inherently associated with the volatility of the network as payment channels may be opened or closed at any given time. Consider the network as a directed graph, where each edge

[1] Miners are machines who solve difficult cryptography-based mathematical problems to confirm transactions and record them on the blockchain.

indicates the funds allocated to one side of a payment channel and for each edge, there exists a partnering edge that together forms the entire bidirectional payment channel. For a given path to be capable of routing a payment, each hop must have at least as many funds available on that edge as the payment amount (including fees). Pathfinding for payments also typically occurs at the source node with no clear understanding of the optimal way in which such payments should be routed.

Nevertheless, this project tackles the routing problem from a different angle; that is, even given a path comprised of responsive, non-malicious parties, there is no guarantee that the payment can be completed. This is primarily as a result of the privacy-preserving nature of LN - it is impossible to know if a given route is even capable of supporting your payment request. To be specific, the total funds allocated to a payment channel is publicly visible, but the distribution of funds between the two parties is private. To be certain a path is unconditionally capable of routing a payment, the distribution of funds would need to be known for each hop. To evaluate LN, we must consider its success rate of payment deliveries and identify the network conditions in which it can scale in this regard. To accomplish this, we have built a simulator of LN.

LN is ever-improving due to the community behind it that constantly drives its adoption with the proposals of innovative extensions to the protocol. An initial shortcoming of LN was its limitations concerning the splitting up of payments into smaller, more manageable chunks. The release of Atomic Multipath Payments (AMP) [17] has resolved this limitation, but has had limited maturity in practice. As of yet, it is difficult to estimate how payments should be split and how they are expected to perform. As such, experiments have also been conducted with multi-part payments to evaluate its performance.

In a similar regard, another limitation of LN is associated with how funds are allocated to channels. Once created, the total amount of funds locked up in a channel is fixed and cannot be changed without the closure and reopening of the channel, which is costly as it requires slow and expensive on-chain transactions. Over time, some channels may become unbalanced, leaving one side depleted and ineffective - this is problematic for routing payments. However, channels can be re-balanced by circular rotations of funds, where a given node sends a payment to itself in a cycle [6]. Another community proposed technique, labelled Just In Time Routing (JIT Routing) [19] performs a re-balance once it is required to complete a payment at that given moment. That is, if a payment cannot be forwarded, a re-balance is performed and if successful, the original payment may now be successfully forwarded. The simulator has also been used to investigate the impact of the adoption of this technique by all network participants.

Through a series of experiments, our findings support the adoption of both Atomic Multi-path Payments and Just In Time Routing. An insight into the most appropriate manner in which payments should be split is also provided, at least when split evenly into four or less partial payments. Variable size splits, and splits of five or more are beyond the scope of the experiments conducted. Our experiments have concluded that JIT Routing is a technique that all network participants should adopt. The usage of JIT Routing allows to network to remain operational for longer, lessening the need for the closure and re-opening of unbalanced channels.

Our experiments also reveal where LN can and cannot scale. Increasing the number of payments initiated per second does not congest the network as much as originally expected, likely due to the high degree nature of participants of the network. We do however note that the increase in payment amounts has a direct decrease in payment success rate. In practice, we feel the success rates of lower payment amounts are more important, due to LN's application as a micropayment system.

In general, we have found that routing by the shortest path tends to yield the best results in terms of payment success rate. Utilising longer paths, even if they are cheaper (in terms of fees) does not appear to be a reliable search strategy.

The remainder of this paper is structured as follows. In Section 2 we provide a detailed understanding of how LN works, coupled with a review of the publications surrounding LN and other PCNs. In Sections 3 and 4 we present the design and implementation of the simulator. The experiments conducted using this simulator are introduced in Section 5, accompanied by the results and an evaluation of each. Finally, we conclude in Section 6.

2 Background

The Lightning Network is the most prominent solution to Bitcoin's scalability concerns available today. In this section, we hope to deliver an understanding of how LN works in practice, coupled with a detailed outline of the operation of Atomic Multi-path Payments and Just In Time Routing. Hopefully, this will serve as a grounding point in delivering the simulator. In addition, related publications have been reviewed in attempts to identify the role of this project in augmenting the growing literature of this area.

Blockchains A blockchain is a distributed, append-only ledger. Each block stores some data, a hash value of contents of the block and the hash value of the previous block in the chain. In the case of Bitcoin, the data stored are a number of transactions that send bitcoins from one address (wallet) to another [14]. If a block is tampered with, its hash value will change. This secures the blockchain - if a block changes, its hash value will change, invaliding all subsequent blocks as each block stores the hash of the previous block. To change a block, one would need to re-calculate the hashes of every subsequent block - the security concerns here are mitigated by requiring a proof-of-work to be calculated to confirm blocks, which in Bitcoin's case, takes ten minutes [7].

Blockchains are also secured by their distributed property. Rather than using a central governing entity, blockchains are used on a peer-to-peer network that is open to anyone to join. Each participant has a copy of the blockchain, and validation of newly confirmed blocks is done by global consensus of the network participants. In this way, to tamper with a block, a malicious node would need to do all of the required proof of work, and have control of more than half of the network. While this ensures the system is very secure, it becomes the root cause of its scalability issues.

2.1 Prerequisite building blocks

To understand how LN works, it is fundamental to understand the following building blocks as they are crucial in the network's operation.

Unconfirmed transactions Transactions are central to Bitcoin's operation and they consist of inputs and outputs. The inputs represent the addresses from which the bitcoins are sent from, and the outputs are the destinations of the funds. Both the inputs and outputs have accompanying requirements that must be fulfilled to complete the transfer of funds, such as signatures that must be provided to prove ownership. Note that the outputs of a transaction can only be spent from once, as Bitcoin has protection against double-spend attacks [14].

A transaction is unconfirmed if it is not broadcast to the blockchain but instead stored locally, available to be broadcast at any point in time. Most of the transactions on the Lightning Network are unconfirmed, happening off-chain. This alleviates the burden on the blockchain.

Time-locks As previously stated, transactions have accompanying requirements that must be satisfied to allow spending from. A time-lock on a transaction means the funds only become spendable from the output of that address after a certain time period has passed. There are two different types of time-locks, *a*) $CheckLockTimeVerify$ (CLTV) which refers to a specific date and time in the absolute sense, and *b*) $CheckSequenceVerify$ (CSV) which is relative, whereby once the transaction is recorded on the blockchain, the outputs only become spendable after a certain amount of blocks have been confirmed since that point in time.

2.2 Mechanisms behind the Lightning Network

The Lightning Network is based on a mechanism known as payment channels; that is, two mutually distrustful parties can establish a payment channel between each other to send funds back and forth in an almost instantaneous manner.[2] In the case of LN, such channels are bidirectional, which is not always the case [25].

All transactions of this manner occur off-chain (unconfirmed), aside from an initial set up transaction known as a funding transaction, and a final closing transactions known as a settlement transaction. Essentially, to open a channel, both parties deposit a certain amount of bitcoins in a 2-of-2 multi-signature address[3] which requires a signature from each party to be "unlocked" and spent from. The channel can be thought of as a balance sheet and transactions update this balance sheet, provided both private keys are yielded. The channels may be kept open for as long as either party wishes, allowing for an unlimited number of re-distributions of the balance of the funds held within that multi-signature address. While in the channel, funds are "locked up" in that channel until closed. Finally, the settlement transaction (on-chain) is used to distribute the funds back to each owner's original wallet in accordance with the final balance sheet.

Security & incentive to play fair The process of opening channels and updating the balance within channels is not quite as straight forward as explained above. To present how the system works in a more digestible manner, we label the parties in the common way - Alice and Bob, who wish to lock up a total of θ bitcoin (BTC) - Alice locks up δ BTC and Bob locks up $\theta - \delta$ BTC. Before broadcasting the funding transaction, which moves the funds into the multi-signature address, a new transaction known as a "commitment" transaction must be created by both parties to allow spending from that multi-signature

[2] Distrustful in this case means parties do not need to trust each other to set up a channel - their funds are secured.

[3] A multi-signature address is a Bitcoin address that requires multiple private keys to spend from - in most cases for LN, this requires two of the two possible keys, hence 2-of-2.

address [21]. This is done before signing and broadcasting the funding transaction in case the parties become uncooperative, possibly locking up the funds forever. To do this, Alice and Bob both create a secret and exchange the hashed values of the secrets with each other.[4] In the case of Alice, she creates a commitment transaction which sends δ BTC to herself, and $\theta - \delta$ BTC to another more complex multi-signature address: which can be unlocked by Bob after 1000 new blocks have been confirmed on the blockchain (CSV time-lock), or by Alice, but only if she knows Bob's secret. Alice transfers this commitment transaction to Bob - Bob does the opposite and gives his commitment transaction to Alice. At this point, the original funding transaction is broadcast on the blockchain and the channel is officially opened.

Essentially, transactions on this channel are nothing more than a chain of commitment transactions that spend the outputs of the original funding transaction. As such, to update the balance of a payment channel, both Alice and Bob need to create new commitment transactions. By design, the broadcast of old commitment transactions imposes risk, allowing the counterparty to claim all of the funds as a penalty - hence, the incentive here is to play fair. Every time Alice and Bob wish to update the balance sheet, the process is repeated and a new commitment transaction is formed with a new distribution of the funds (θ BTC), and each time using newly formed secrets. Old secrets are exchanged to invalidate the older transactions.

To understand the security here, imagine Bob is paid a greater amount in an older commitment transaction and wishes to broadcast it. Once he does this, Alice gets sent her original δ BTC, and Bob may get his $\theta - \delta$ BTC after 1000 confirmations on the blockchain - but now Alice knows Bob's secret associated with this commitment transaction and can claim the remainder of the funds before they get to Bob. With an average of a ten minute block time, 1000 blocks takes some time. As such, Alice is left with the entire θ BTC, and Bob with nothing. The transaction Alice broadcasts to claim the remaining funds from Bob is known as a penalty transaction (sometimes referred to as a justice transaction).

Naturally, this requires Alice to keep an eye on the blockchain to penalise Bob if he is not playing fair. It is possible to delegate this to a third party by giving them the penalty transaction that must be broadcast in the event of misplay, perhaps in return for a fee [21]. As the third party only has access to the penalty transaction, they cannot force close the channel - they can only act in response to malicious actions. These third parties are often referred to as watchtowers.

[4] The secret is a random integer hashed using ECDSA - private, public key pair.

The settlement transaction used to close the channel is the present commitment transaction. To cooperatively close the channel, Alice and Bob can agree to create an exercise settlement transaction, which is a special type of commitment transaction that has no time-locks on the outputs [21].

A network of channels As previously mentioned, channels may be strung together to form a network. This allows two parties that do not share a direct payment channel to transfer funds between each other in a trust-less manner. We introduce a new party, Carol, who has a direct payment channel with Bob. In this scenario, Alice is attempting to send funds to Carol by routing her payment through Bob. To accomplish this, she must pay Bob who in turn must pay Carol; however, she has no way of knowing if Bob will actually pay Carol. The security measures described up until now only permit the secure transfer of funds within a channel, and cannot account for the security of an entire network of channels.

To understand how this network remains trust-less, the idea of a hash time-locked contract (HTLC) must be introduced. When Alice wants to send funds to Bob as part of a multi-hop payment, she must create an HTLC. This is essentially a new multi-signature address that holds the funds temporarily. The funds from this HTLC may be spent in one of two ways: either by Bob, if he produces the preimage of a predefined hash, or by Alice after a certain deadline (which is a CLTV time-lock) [21]. This means if Bob cannot produce the preimage after a certain period, the funds are returned to Alice. To gain the preimage, Bob must forward the payment to Carol.

The preimage is created ahead of time by the destination node, Carol. To allow Alice to construct an HTLC with Bob, Carol shares the hash of the preimage with Alice. This is the hash used in the HTLC. Bob does the same, creating an HTLC with Carol. As Carol created the preimage, she can immediately claim the funds by revealing the preimage. In this way, a series of HTLCs are constructed along the path of the payment until they reach the destination. The payment is then complete, and each node claims the funds in the HTLC as the preimage propagates back towards the sender, Alice. When the preimage reaches Alice, it serves a cryptographic receipt or proof of payment. This can be observed from Figure 1.

To be clear, while a commitment transaction is in effect a balance sheet of funds between Alice and Bob, when an HTLC is constructed the funds are split with an additional output - some to Alice, some to Bob and some to this new HTLC address. However, HTLCs must be closed out and settled once either of the requirements have been met. If not, even after the time-lock has surpassed, Bob can at a later time produce the preimage and claim the funds by broadcasting to the blockchain, or vice versa - even if Bob produced the preimage on time, after the time-lock is over, Alice will be able to claim the funds. Hence, to settle the HTLC, a further commitment transaction is created where the balance sheet is updated to reflect the rightful claiming of the HTLC funds. If a party is not co-operative in this manner, the counterparty may close the channel and the HTLC will be settled on the blockchain.

The time-out assigned to each HTLC in a payment must be decreasing in value for each hop towards the destination. This is to avoid a situation where Alice can claim back her funds, but Bob cannot claim his from Carol yet. This would be dangerous as Carol might suddenly produce the preimage, leaving Bob's funds effectively stolen. Each node can broadcast a *cltv_expiry_delta* value, so the timeout of a given HTLC must be at least as many blocks as the value for the subsequent HTLC plus the *cltv_expiry_delta* [4] - hence this is calculated in a reverse fashion from destination to source.

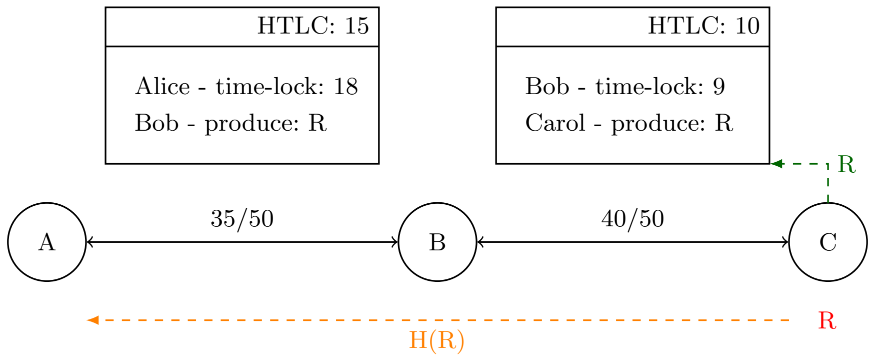
Fees In addition, on a per-channel basis, nodes may declare a fee rate for forwarding on payments. Fee rates are comprised of a base fee (in a way, accounting for the work done) and a multiplier on the amount being forwarded (in a way, accounting for the risk associated with time-locks of HTLCs). This can be observed in Equation 1.

$$txFee = baseFee + feeRate * txAmount \tag{1}$$

It is important to note that the *txAmount* that is multiplied with the fee rate changes at each hop - the fees are in a way compounding. For any given hop, the *txAmount* is the amount sent over the subsequent hop (including fees). There are no fees on the last hop, so the original payment amount is all that is sent. Hence, the second-to-last hop is the only definite hop in which the fees are calculated using the original payment amount. Much like the HTLC deadlines, fees must also be calculated in reverse from destination to source.

Fees are effectively taken by forwarding on a lower amount than received - for example, if Bob receives 2 BTC from Alice, and sends Carol 1 BTC, he has then claimed a fee of 1 BTC. The funds allocated to each payment channel have not changed, but the total funds that Bob has access to has increased. This is also illustrated in Figure 1.

Pathfinding In general, all pathfinding occurs at the source node with each node holding knowledge of the entire graph. A node may use any algorithm and criteria seen fit, such as the shortest path in terms of hop count, or the cheapest path in terms of total fees paid. There are many different factors that



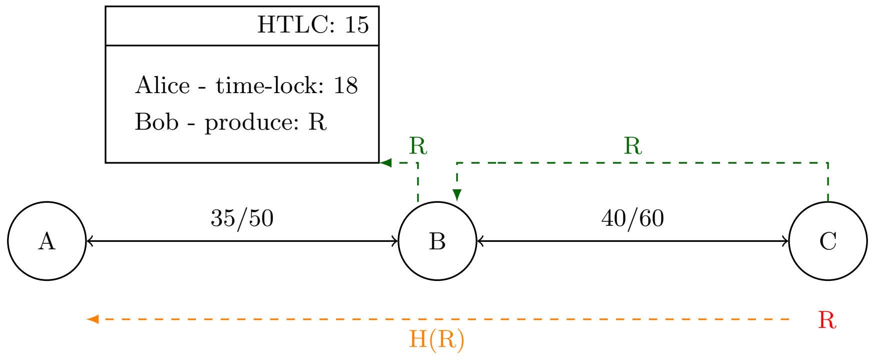


Fig. 1. A payment between Alice and Carol - two HTLCs are constructed using H(R). Before **(top)** and after **(bottom)** Carol settles the HTLC using the preimage R, claiming her funds (10). The time-lock on Alice and Bob's HTLC is longer. If Bob settles the other HTLC, he can claim his funds (15), earning a total of 5 in fees. The units of time on the HTLC deadlines are confirmations on the blockchain (CLTV time-lock).

one could consider. As previously mentioned, fees are compounded at each hop and for this reason, the path must be found in the reverse direction - starting from the target node. A route must be found for which sufficiently supports the funds to be sent at each hop (including fees), but this is problematic as the distribution of funds within a channel is private information. All that is publicly available is the total funds allocated to a channel. This is often referred to as the channel capacity and can be found by observing funding transactions broadcast to the blockchain. As such, one has no idea if the path they choose to route their payment is even capable of supporting it in the first place.

While LN currently uses source routing, this may not always be the case in the future with protocols such as the Ant Routing protocol [9] being established to address LN's routing problem.

The way payments are propagated is also privacy-preserving. All payments are onion routed [10], whereby any node on the path can only know its predecessor and successor. Nodes do not even know if they are forwarding to the final destination in the path, or receiving from the original sender.

Failed payments We have already introduced the notion of a payment being successfully routed - the preimage is released and propagated back towards the sender and all HTLCs become fulfilled. If Bob receives the preimage and has an HTLC set up with Alice, they agree is update the channel (with a new commitment transaction) to close out the HTLC with a new distribution of funds where Bob receives the funds that were allocated to the HTLC.

We can use this notion further when confronted with failed payments. If Bob realises he does not have sufficient funds to forward onto Carol, he can cancel the payment with Alice by tearing down the HTLC with a new commitment transaction that reverts to the original balance before the HTLC was set up. In this way, funds are not locked up for unnecessary periods of time provided nodes are cooperative.

Nodes going offline An intermediary node in a payment may go offline before the payment reaches that particular node, in which case the payment can fail in the usual sense - if Carol is offline, Bob cannot create an HTLC with her and then agrees to tear down the HTLC set up with Alice. It is also possible that a node will go offline after it has forwarded on a payment, which is the more interesting case. Imagine a scenario where Alice pays a new node Dave, along the path Alice → Bob → Carol → Dave - if the payment successfully reaches Dave, he will reveal the preimage to Carol and claim his funds. Yet, if Bob suddenly goes offline, Carol will not be able to settle the HTLC with Bob. In this case, Carol can either wait for Bob to return, or if the time-lock on the HTLC has nearly expired, she will have to close the channel to safe-keep her funds, using the blockchain to settle the HTLC. In this sense, Carol is always safe as long as she acts accordingly. Additionally, any other cooperative nodes thereafter will be able to use the preimage to claim their funds as it will have been publicly broadcast to the blockchain by Carol.

The real risk involved is for the uncooperative node, Bob. If Bob stays offline, Alice will eventually be able to claim back her funds by closing their channel and as such, Carol will have been paid by Bob but he will have never been paid by Alice.

2.3 Atomic Multi-Path Payments

Several problems arise when attempting to send large amounts of funds in a single payment. There are fewer available paths and the closer a payment amount is to the total capacity of a channel, the more likely it is to fail. Secondly, the funds available to a node may be distributed across multiple channels, where no single channel on its own has adequate funds to support the payment. Finally, there is currently a limit on the total amount of funds that can be sent in a single payment - 2^{64} millisatoshi [4].[5]

Hence, it is only natural to support the idea of multi-part payments, which in LN has been labelled Atomic Multi-path Payments (AMP) [17]. Atomic, in this case, refers to the nature in which the recipient receives sub-payments - funds are only to be claimed from a partial payment if all partial payments have reached the destination - all or nothing.

In the initial proposal, the atomic nature of these payments is accomplished by creating two unique preimages for each partial payment, r_i and s_i, at the source node - this would have normally occurred at the destination node. To each partial payment, attached is the hash of r_i, which will be used to set up the HTLCs as normal and s_i. By taking the XOR ($\oplus$) of all of the preimages s_i, a base preimage $BP = s_1 \oplus s_2 \oplus ... \oplus s_n$ is produced. This base preimage can be used to calculate each r_i as $r_i = SHA256(BP \parallel i)$, where $\parallel$ refers to the concatenation operator. In this sense, the receiver will have to collect every partial payment in order to redeem any of them. The problem with this approach is that as the sender is creating the preimages and the cryptographic receipt or proof of payment is lost which would otherwise accompany non-split payments.

To alleviate this problem, a simpler approach has been considered which is known as Base AMP [24], [30] where the same preimage is used for all partial payments and generated by the receiver as normal, but the receiver does not release the preimage until all partial payments have arrived. Intuitively, in a real-world scenario, the provider of a service would not give a receipt for the partial fulfillment of a payment. As such, when the preimage is finally released, the sender can assume all partial payments have been received, providing proof of payment. If all partial payments have not been received within a certain predefined time period, typically 60 seconds [24], all are cancelled as normal.

[5] A bitcoin is divisible by the eight decimal into a satoshi. This is the smallest unit available for Bitcoin transactions, but LN supports millisatoshi. When channels are closed, funds are rounded down to the nearest satoshi.

This solution is currently supported by major LN protocol implementations such as c-lightning.[6]

2.4 Just In Time Routing

Despite the name, Just In Time Routing (JIT Routing) is not a method of finding routes for payments, but rather a method of re-balancing channels when required ”just in time”, or at least attempting to in order to fulfil a payment that would have otherwise failed [19].

Re-balancing When a payment channel becomes unbalanced, we refer to one side as depleted. Depleted edges in a payment path cause it to become more likely to fail as the sender may not have any knowledge of how depleted the edge is.

Yet we can still re-balance the distribution of funds - all it would take is for the non-depleted edge to send funds to the depleted edge. Using this fact, the idea of circular re-balancing can be established, where a node sends a payment to itself with a cyclic route with the aim of the last hop of the payment causing a re-balance [6].

JIT Routing JIT Routing integrates the idea of a circular re-balance by performing it immediately once the next hop of a path is too depleted to continue a payment. If successfully executed, the payment is resumed, otherwise the payment fails as usual.

The intuition here is that we cannot predict the future so there is no point in attempting to keep the network constantly balanced. If a channel is unbalanced, this might not be a problem for the immediate payments to follow - they might still be supported by the depleted edge, or payments may be coming from the other direction which in turn automatically re-balances the channel anyway. Some payments are better suited to unbalanced channels if they require almost the entire capacity of the channel to be at one end. By re-balancing just in time, the problems that arise due to depleted edges can potentially be overcome, with no extra unnecessary re-balancing of channels. A successful application of JIT Routing can be observed in Figure 2.

Furthermore, when a path is normally found, the sender only has knowledge of the distribution of funds allocated to the channels it participates in. With

[6] c-lightning is one of the most widely used implementations of the Lightning Network protocol for running nodes, along with the Lightning Network Daemon (LND).

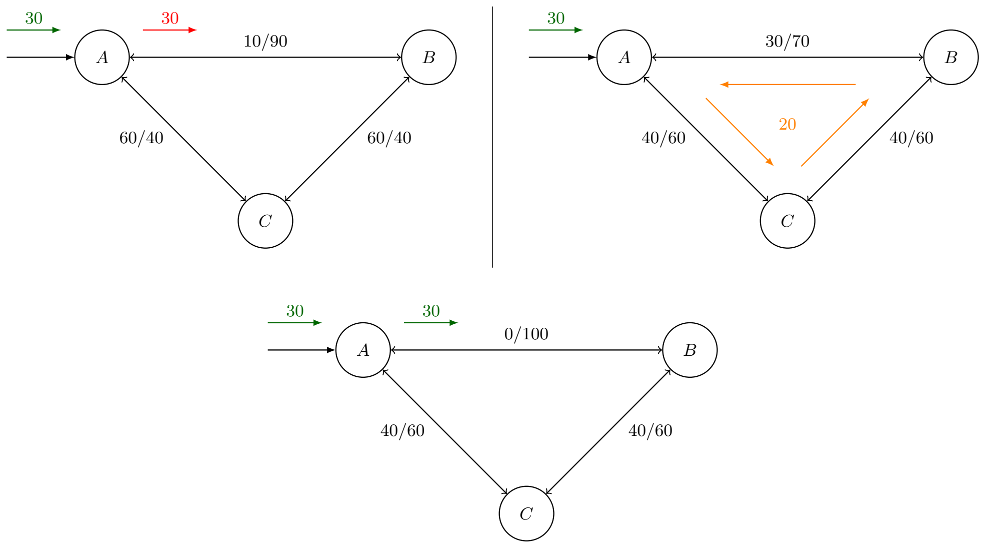


Fig. 2. Re-balancing the channel between Alice and Bob (via Carol) to support the incoming payment of amount 30 that Alice must forward to Bob - before **(top left)**, after re-balancing **(top right)** and after forwarding to Bob **(bottom)**.

JIT Routing, more channel balance information can supplement the pathfinding process without leaking privacy. The node performing the re-balance holds local channel information that the source node does not.

2.5 Problem statement

The Lightning Network is released and in-use, but currently very little has been experimented with in terms of AMP and JIT Routing - hence, this project will evaluate the performance of both techniques, individually and when combined, on the metric of payment success rate. This will help determine how payments should be split up (if at all), and whether or not re-balancing just in time is a method that should be adopted by all participants of the network.

It is also valuable to establish how the network performs with regards to the successful delivery of payments when various network conditions are considered. This will help expose under what circumstances the network can scale, and to identify the tipping points in which the system begins to break down and become unusable.

Finally, the optimal pathfinding algorithm is not clear as of yet - hence, this project will also compare a number of search strategies to determine the best performer.

2.6 Related works

This method of empirical research is not the first of its kind in this area and the following are publications detailing simulators of the Lightning Network and payment channel networks. This list is by no means exhaustive as the body of literature in the area continues to grow. To the best of our knowledge, there has been no formal publications studying the adoption of JIT Routing specifically, although the idea of re-balancing has been investigated [11], [20].

Sprites and State Channels: Payment Networks that Go Faster than Lightning Miller et al. [13] proposed a payment channel network known as Sprites which unlike the Lightning Network, allows for partial withdrawals and deposits of funds without channel closure, and reduces the worst-case amount of time funds may be locked up for in a payment. They built a simulation framework to model both LN and the Sprites network for comparison in their experiments, showcasing the benefits accompanying Sprites. While LN is the most popular PCN to surface, it does not necessarily mean it performs the best and this paper demonstrates how other PCNs can overcome some issues related to LN. The generation of their simulation network and handling of payments has been notably influential in the development of this project.

A Cryptoeconomic Traffic Analysis of Bitcoin's Lightning Network To empirically study LN's transaction fees as well as privacy provisions, Béres et al. [3] delivered a simulator of the network and found that the current fee rates are far too low to provide economic incentive to participate in the network.

They also discovered that the small-world nature of the network allows for the statistical inference of source and destination nodes of a payment. As such, they propose purposely lengthening payment routes to increase privacy, claiming that the fee rates are too low for this to be an issue; however, they have not considered how this affects the success rate of payments.

The network is simulated by using snapshots of LN, rather than simply generating an estimate of the network topology, which is an interesting approach.

Split Payments in Payment Networks Piatkivskyi and Nowostawski [18] developed a simulator to evaluate the performance of payments when split under the same metric as this project; however, in this case, split payments refer to those that are not atomically split in the same manner as with AMP. Split payments disperse partial payments over a longer time period and allow for the partial receiving of payments, rather than sending all partial payments at once as is the case with AMP.

The entirety of the experiments conducted compare split payments to AMP, offering no comparison between non-split payments. The results indicate a favour towards split payments over AMP as expected; however, the atomic property of AMP is important. As such, we feel the comparison between non-split payments and atomically split ones is a more significant investigation, particularly as AMP currently becomes the standard in which nodes split payments on the network itself. Note that elsewhere in this report, where we refer to multi-part payments, AMP is assumed.

3 Design

In this section, we present a high-level overview of the simulator we have built of the Lightning Network. A simulator is necessary to evaluate how the network performs and to investigate the implications of adopting Atomic Multi-path Payments and Just In Time Routing. In particular, due to the privacy features of LN, there is no information available on transactions between participants, so a simulator is necessary to investigate its operation.

This simulator is used to process a fixed number of payments throughout the network, and monitor how many payments are successfully delivered. The goal is to evaluate the payment delivery success rate under various network conditions to expose where the system can scale and where it cannot.

3.1 Graph topology formation

It is difficult to estimate exactly how the Lightning Network will emerge in practice as it continues to grow. Ideally, a topology will materialise that is as decentralised as possible - yet, this is unlikely, with the emergence of a hub-and-spoke style topology appearing to be a more realistic outcome [12]. Two different topological patterns are assumed in other networks, such as those presented in Section 2.6. These two models are known as small-world and scale-free. The snapshots taken by Béres et al. of LN also show a small-world structure. A small-world graph is one in which the average path length between any two given nodes is at most $O(nlogn)$. This can be generated using the Watts-Strogatz (WS) algorithm [16], which also ensures that the generated small-world graph is not scale-free. On the other hand, a scale-free graph follows a power lower for degree distribution [2] which results in the formation of hubs. This preferential attachment process is modelled using the Barabási-Albert (BA) algorithm [1]. All experiments are run on both of these graph types, with a network size of 2,000 nodes, which is the same number of nodes used in experiments conducted by Miller et al. [13]. The average degree of nodes is ever-increasing on LN, but for this simulator, a degree of 10 is chosen which was akin to the average degree of nodes on the actual LN at the time when experiments were conducted.[7]

3.2 Node & payment channel generation

The generated graph, denoted G, is directed, with each directed edge representing the funds held on a particular side of a payment channel. To distribute initial funds, each edge is labelled as either a "high" edge with a 20% probability, or a "low" edge with an 80% probability. Such edge types are initialised with a starting balance of 8,000,000 satoshi and 500,000 satoshi, respectively. The selected values are similar to those used by Miller et al. [13].

Additionally, we also assign a latency distribution between nodes. Where a payment channel exists, the two nodes are assigned a latency which follows the distribution of 92.5% having a 100ms latency, 4.9% having a 1-second latency, and the remaining 2.6% having a 10-second latency. These are based on estimates by a study conducted by Neudecker et al. [15] who measured timing on LN.[8]

The previously mentioned fee rates must also be considered. The current default values for LN are a 1 satoshi base rate and a 0.000001 satoshi multiplier. The majority of users maintain this default rate [23], as low as it is - hence, for simplicity, all directed edges by default have been assigned these values.

Each node has been assigned a role as either merchant with a 2/3 probability, or consumer with a 1/3 probability, again, akin to Miller et al. [13]. This is to model the natural flow of funds. We additionally introduce another variable, β, which represents a percentage chance to behave as the opposite role to allow merchant to merchant, consumer to consumer and consumer to merchant payments. As a default, $\beta = 0$.

3.3 Generating payment requests

It is difficult to model a realistic distribution of payments within the network, particularly because the privacy-preserving features of LN restrict studying existing patterns of payments.

For simplicity, a merchant is paired with a consumer (again, with a β% chance for either node to behave as the opposite role). The amount of satoshi

[7] Source: `https://1ml.com` - Accessed Mar. 21, 2020.
[8] This includes internet latency as well as any work done to forward on the transaction.

to send within a payment is a random integer from 1 to α, where α is the set maximum payment amount. By default, $\alpha = 200000$ satoshi, which has been arbitrary chosen.

Payments are repeatedly sent at a particular frequency to control the 'activity' level of the network, until 7,000 payments have either succeeded or failed in total. Béres et al. [3] also used a total of 7,000 payments in their experiments. The frequency in which payments are sent is represented by λ, the number of payments sent per second. By default, $\lambda = 50$.

Route-finding In general, all route-finding occurs at the source node. By default, payment routes are chosen by finding the shortest path, but we also explore the cheapest path in terms of fees, and the path which holds the highest average 'slack'. Slack here, on a per-hop basis, refers to the total channel capacity minus the amount of funds being sent over that channel. The intuition is that a payment of amount θ is likely to fail on a channel whose total allocated funds is $\theta + \delta$ if δ is comparably small.

3.4 Communication & Timing

Communication on LN happens via standard TCP connections between nodes. LN itself is completely peer-to-peer, but to control the timing of messages being sent in the simulator, a central communication object is used to relay message packets between nodes, ensuring the correct latency is applied. Hence, when each packet (message) relayed, a future timestamp is assigned to the packet to indicate when it is ready to be processed by the other node.

Timing A central priority-ordered queue of packets based on their timestamp is maintained to process packets. To ensure the correct ordering in which packets are processed, particularly when there is latency involved, a simulation time is maintained. New payments are initiated as the timer is advanced and new payments are due. The timer is temporarily stopped during pathfinding for simplicity.

3.5 Atomic Multi-path Payments

To accommodate multi-path payments which are to be split into k partial payments, the funds being sent are divided and sent as k independent payment packets. Packets that are sent as part of a larger payment are handled in the same manner by intermediary nodes; however, a receiving node cannot claim the funds of a partial payment until all have reached the destination within the time constraints, which will be detailed in Section 4.5. If a payment is not successful, any received partial payments must be returned to the sender as failed. Additional functionality for the re-sending of partial payments is also supported.

It is impossible to say for sure the best manner in which to split payments - by design, the simulator divides the payment equally into k parts and the pathfinding algorithm is optimised to find k independent shortest paths if possible.

3.6 Just In Time Routing

To support JIT Routing, a separate pathfinding algorithm must be developed to support finding routes for re-balancing. In this case, the algorithm finds cycles of length three hops, and the last hop must be the payment channel that is intended to be re-balanced. Then, when a node is unable to forward on a payment due to insufficient funds, it can initiate a re-balance, which if successful will allow for the forwarding of the original payment. If it not successful, the original payment can fail as normal. In reality, the cycles may be of any length, but are limited to three hops in this simulator for simplicity.

We propose the following extension to the protocol: if Alice wishes to re-balance her channel with Bob and can route through either Carol or Dave - let Bob inform Alice whether she should route through Carol or Dave for the best chance of success - presume Bob says Dave. The concern that arises here is privacy, but if Alice attempts the re-balance and it is successful using Bob's advice, all she has learnt is that Dave had enough funds with Bob to support the payment. This is the same knowledge she would have learnt by attempting the payment anyway. The caveat here is Bob should only continue to provide advice if Alice actually attempts the payments using the given advice thereafter.

A more pressing concern is if the re-balance fails using Bob's advice - if he was truthful, then it means the re-balance also would not have worked by routing through Carol. This is extra information for Alice. A counter-argument here is that Alice has no guarantees that Bob is being truthful, and also has no guarantees that Bob's original advice still holds by the time the re-balance reaches Dave. If nodes maintained an honesty factor where they do not always tell the truth, the risk of nodes learning fund distributions of others can be mitigated.

4 Implementation

To model the network, a graph developed with Python's NetworkX package is used,[9] which is a package used for the creation and manipulation of complex networks. Python was chosen due to its substantial usage throughout the Computer Science course in UCC, and it felt the most comfortable for the task at hand. Furthermore, after investigation, NetworkX surfaced as a highly recommended, well documented and mature library that was suitable for this project.

4.1 Graph components

In the simulator, nodes represent participants of the network and edges represent payment channels. To be precise, the graph is directed and each edge represents one side of a channel - this means each edge has a partner that together forms the entire bidirectional payment channel.

Payment channel generation As described in Section 3.1, all experiments have been conducted on two generated network topologies: *a*) small-world and *b*) scale-free. The respective algorithm implementations for both of these network types are built into the NetworkX library; however, they required minor adjustments for the project and were adapted accordingly. Each directed edge has an associated amount of available funds, advertised fee rates and the initial starting funds. The initial amount of funds allocated to a channel is maintained so that other nodes can use this information when pathfinding, as the amount of available funds is private information. The starting balance and fee rates of edges are assigned in accordance with the chosen simulation parameters.

Node initialisation Participants of the network are classified as either merchants or consumers, where consumers are in general more likely to initiate payments. The actual implementation also includes a spend frequency and a receive frequency to distribute the initiation of payments across consumers and distribute the receiving of payments across merchants. The set of final experiments conducted does not take advantage of this feature however. This is in part due to the absence of a well-qualified dataset of existing payments to model the simulations on, which was the case with the simulator built by Miller et al. [13]. As such, the distribution was uniform with nodes being selected from the merchant and consumer groups with equal probability.

[9] See: `https://networkx.github.io`

4.2 Packets & Communication

To handle the propagation of payments throughout the network, nodes communicate with each other using packets. The packets are relayed along the path of a payment and at each hop, a communication object known as the *NodeCommunicator* is used to handle the sending of packets between nodes. Even if a payment is a single hop, direct balance sheet update, a packet is used.

Packets on the Lightning Network are onion encrypted [5], [10] - at each hop, nodes can see where to forward the packet to and the expected amount of funds to send. To simulate this, a dictionary is used to map the ID of a node to the index where its relevant information can be found within the packet. To begin with, we will first consider a packet in the case of a non-split payment with JIT Routing disabled. Packets can be of type *pay*, *pay_htlc*, *preimage*, *cancel*, or *cancel_rest*.

- *pay*: Used to update the balance sheet of a payment channel - no hash time-locked contracts are constructed.
- *pay_htlc*: Used to lock up funds into an HTLC for a payment.
- *preimage*: Can be used to release funds in an HTLC. Once a payment has successfully reached its destination, the receiver propagates the preimage back to release HTLC funds.
- *cancel*: If a payment cannot be forwarded, it can be cancelled by tearing down the set-up HTLCs and returning the locked-up funds.
- *cancel_rest*: A special type of cancellation, where there was no HTLC set up with the previous node. If an intermediary node receives a *cancel_rest* packet, it updates the packet type to *cancel* and sends the packet on to continue the cancellation process.

Nodes receiving a packet can use the type (and their portion of the information on the packet), combined with local knowledge of their payment channels to process the packet further. Each packet also has an associated timestamp which indicates if is it ready to be processed yet. Ready in this case refers to whether or not it is 'arrived' at its destination node after accounting for latency.

The NodeCommunicator Class To simulate the transfer time of packets, they are passed between nodes by a *NodeCommunicator* object. When this object receives a packet, it updates the associated timestamp based on the latency between the two nodes and adds the packet to a priority-ordered queue which is shared by all nodes. The priority-ordered queue holds packets of on-going payments. The class is slightly unwarranted due to its simplicity, yet, it helps make the simulator more understandable from a packet communication standpoint.

Timing In the initial implementation of the simulator, UTC-based timestamps were used (the time package in Python). Each node maintained a priority-ordered queue of incoming packets that were to be processed and the simulator iterated over nodes in a round-robin fashion, processing the head of the queues. The number of packets processed at each iteration was in proportion to the number of payment channels associated with that node.

The round-robin approach to selecting the next node raised issues where packets were being processed out of order. To alleviate this problem, the use of a secondary priority-ordered queue was considered - the queue would be used to order which nodes to select next, based on which node had the next available packet to process, timestamp-wise. While in theory this solution would resolve the problem, a shift to maintaining a custom simulator time felt more appropriate as it is a computationally more efficient solution.

The distributed network simulator is based on those by Zivan and Meisels [29], and Wahbi and Brown [27]. The simulator time is initialised as time zero and advanced manually. In doing so, the CPU processing time does not affect the operation of experiments. To handle the ordering of packets, all active packets across the network are stored in a single priority-ordered queue based on their timestamp - this way, the next available packet is always processed first. As the processing of packets is handled by nodes themselves, once popped, a packet is passed to the associated node for processing.

Additionally, a timestamp τ is maintained for the next time to send a packet, initialised as time zero. Whenever there are no more packets available to process at a given time, we check if it is time to initialise a new payment. If so, the payment is sent and τ is advanced in relation to the number of payments that are sent per second, λ. For example, if $\lambda = 50$ then $\tau = \tau + 0.02$. Then, the simulator time is advanced to the next packet processing time, or next payment initialise time, whichever is closest. A visual representation of this is presented in Figure 3.

4.3 Pathfinding for single payments

As a reminder, the path must be found from destination to source due to the calculation of fees.

Default search strategy In the implementation of this simulator, the primary pathfinding algorithm used is Dijkstra's algorithm for finding the cheap-

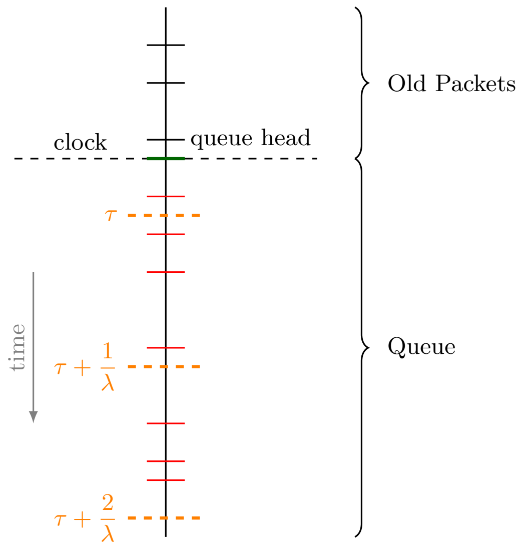


Fig. 3. Time handling of packets in the simulator: packets are inserted into a queue which is ordered by timestamps (nearest first). Solid horizontal lines represent packets - packets of completed/failed payments are black, ready to process packets are green and packets that are not ready yet are red; therefore, from the clock time (black dashed line) down is the queue - anything above this line is off the queue and completed. Once processed, packets are re-inserted into the queue if they are still required for an on-going payment. The orange dashed lines represent the initiation time of a new payment. The clock is always advanced to the next red or orange line.

est path (in terms of fees). As previously stated, the default search strategy is shortest path (hops) and not cheapest path; however, by default all fee rates have been initialised to the same value - the default on LN itself. As such, the resultant path found is of the shortest length in terms of hops. By default, breadth-first search is not used as Dijkstra's algorithm is necessary when pathfinding for multi-part payments. To find the shortest path in experiments that compare search strategies, where the fee rates have been altered, breadth-first search has been used.

NetworkX has a built-in implementation of Dijkstra's algorithm, but it had to be modified to accommodate some LN-related constraints, such as whether or not a channel is capable of supporting a particular payment. The adapted version can be observed in Algorithm 1. A channel is considered capable of supporting a payment if the amount required to send at that hop is less than or equal to the total channel capacity.[10] An exception to this case is for the first hop of the payment (which is the final hop when pathfinding, as it is

[10] In practice, nodes tend to never deplete ends of a channel under a certain point, called the channel reserve - 1% is recommended. This simulator does not impose this recommendation for simplicity.

reversed), where the sending node is aware of the distribution of funds on that channel.

Additionally, it is important to note that when calculating the fees of a hop while pathfinding, that the fee rates of the opposite edge are used because this is the direction the actual payment will be travelling. To increase the speed of the pathfinding, one might also consider running bidirectional Dijkstra, but this is once again impeded by the compounding fees.

Algorithm 1: Dijkstra's algorithm adapted from NetworkX for the Lightning Network.

```
function dijkstraReverse ($src, tar, amnt, avoidEdges$);
Input  : Source and target nodes $src$ and $tar$ and amount of funds to be sent, $amnt$. A
         set of edges to avoid for finding independent paths, $avoidEdges$.
Output: Cheapest path (by fees) between $src$ and $tar$, and the fees paid on that path, or
         False if no path exists.
$src, tar \leftarrow tar, src$ ;                                    // Swap search direction
$dist \leftarrow \{\}$;
$fringe \leftarrow$ heapQueue();
$paths \leftarrow \{src : [src]\}$ ;                         // Maps nodes to cheapest paths
$seen \leftarrow \{src : 0\}$ ;                       // Maps nodes to total fees paid so far
push($fringe$, $(0, src)$);
while len($fringe$) $\neq 0$ do
    $d, v \leftarrow$ pop($fringe$);
    if $v \in dist$ then continue;                          // Already searched this node
    if $v = tar$ then break;                                               // Path found
    $dist[v] \leftarrow d$;
    foreach $w \in$ successors($v$) do
        if $v = src$ then $cost \leftarrow 0$ ;                          // No fees on last hop
        else $cost \leftarrow$ fees($wv$, $amnt + seen[v]$) ; // Reverse direction for fee rates
        $vwDist \leftarrow dist[v] + cost$;
        if $(w, v) \in avoidEdges$ then
            $vwDist \leftarrow vwDist + 100000$ ;             // Arbitrarily large but not $\infty$
        if $w \notin seen \lor vwDist < seen[w]$ then
            if $(w = tar \land amnt + cost <=$ edgeFunds($w$, $v$)$) \lor (w \neq$
              $tar \land amnt + cost <=$ channelFunds($w$, $v$)$)$ then
                $seen[w] \leftarrow vwDist$;
                push($fringe$, $(vwDist, w)$);
                $paths[w] \leftarrow paths[v] + [w]$;
if $tar \in dist$ then
    return reverse($paths[tar]$), $dist[tar]$;
else
    return False;
```

Cheapest path To compare how well the cheapest path performed in comparison to the shortest path, the fee rates set on payment channel edges were distributed as described in Section 5.2 and the same pathfinding algorithm utilised. When comparing the shortest path to the cheapest path, a breadth-first search implementation is used for finding the shortest path.

Highest average slack path To find paths with the highest average slack, where slack on a per-hop basis refers to the total channel capacity minus the amount to be sent, an additional adapted version of Dijkstra's algorithm had to be developed. This can be observed in Algorithm 2. The only change is that instead of comparing the fees imposed at that hop, the average slack is determined and used for comparisons. At every hop, the slack is calculated, and averaged into the overall slack value for the entire path so far.

Initial experiments revealed a very poor performance of this search strategy as it found exceedingly large paths. This is a problem not only because more hops impose more risk of failure, but on LN there is a restriction on the maximum number of hops per payments, which is 20 [5]. To combat this, paths over a length of 20 hops were automatically assigned a slack value of negative infinity.

Negative fees The prospect of negative fees must also be considered in practice - that is, nodes *paying* to be routed through. This may seem nonsensical at first, but if a node wishes to re-balance a depleted channel, they may want nodes to route through them if the payment will result in a re-balancing. The fees paid here are likely to be far less than if they were forced to close and re-open the channel with on-chain transactions. As such, LN protocol implementations such as c-lightning use the Bellman-Ford algorithm for pathfinding, as opposed to Dijkstra's algorithm. For simplicity, negative fees have been ignored in this simulator.

4.4 Processing of incoming packets

Packets, when removed from the priority-ordered queue are ready to be processed by the receiving node of that packet. For now, we will consider the case of a singular, non-split payment without re-balancing.

Direct balance sheet updates When a node wishes to pay a neighbour whom they have a direct payment channel with, a packet of type *pay* is sent. Upon receiving a *pay* packet, if there are enough funds available to complete

Algorithm 2: Highest average slack pathfinding algorithm for the Lightning Network.

```
function highestSlackReverse ($src, tar, amnt$);
Input  : Source and target nodes $src$ and $tar$ and amount of funds to be sent, $amnt$.
Output: Path between $src$ and $tar$ that has the highest average "slack" per hop, and the
        fees paid on that path, or False if no path exists.
$src, tar \leftarrow tar, src$ ;                                  // Swap search direction
$dist, slacks \leftarrow \{\}$;
$fringe \leftarrow$ heapQueue();
$paths \leftarrow \{src : [src]\}$ ;                         // Maps nodes to cheapest paths
$seen \leftarrow \{src : 0\}$ ;                          // Maps nodes to total fees paid so far
push($fringe$, $(0, src, 0, null, 0)$);
while len($fringe$) $\neq 0$ do
    $slackVal, v, d, u, lenPath \leftarrow$ pop($fringe$);
    $slackVal \leftarrow -slackVal$ ;        // Negated on heap so largest is popped first
    if $v \in dist$ then continue;                        // Already searched this node
    if $v = tar$ then break;                                         // Path found
    $slacks[v] \leftarrow slackVal$;
    $dist[v] \leftarrow d$;
    foreach $w \in$ successors($v$) do
        if $v = src$ then $cost \leftarrow 0$ ;                        // No fees on last hop
        else $cost \leftarrow$ fees($wv$, $amnt + seen[v]$) ; // Reverse direction for fee rates
        $vwDist \leftarrow dist[v] + cost$;
        if $v = tar$ then
            $new\_slack\_val = slack\_val$
        else
            $vwSlack =$ channelFunds($w$, $v$) $- (vwDist + amnt)$;
            $newSlackVal = \frac{lenPath * slackVal + vwSlack}{lenPath + 1}$;
        if $lenPath >= 20$ then $newSlackVal = -\infty$ ;       // Discourage long paths
        if $w \notin seen \vee newSlackVal > slacks[w]$ then
            if $(w = tar \wedge amnt + cost <=$ edgeFunds($w$, $v$)$) \vee (w \neq$
              $tar \wedge amnt + cost <=$ channelFunds($w$, $v$)$)$ then
                $seen[w] \leftarrow vwDist$;
                $slacks[w] \leftarrow newSlackVal$;
                push($fringe$, $(-newSlackVal, w, vwDist, v, lenPath + 1)$);
                $paths[w] \leftarrow paths[v] + [w]$;
if $tar \in dist$ then
    return reverse($paths[tar]$), $dist[tar]$;
else
    return False;
```

the payment, the balance sheet is updated and the payment is recorded as successful. Otherwise, the packet type is updated to *cancel_rest* which is sent back to the source node.

Requests to forward a payment Multi-hop payments use *pay_htlc* packets as opposed to the previously described *pay* packet. When an intermediary node receives such a packet, an HTLC is created between the two nodes provided there are sufficient funds available, and the packet is forwarded to the next node in the path. If enough funds are not available, the packet is sent back with type *cancel_rest*.

To mimic the creation of HTLCs, funds are reduced on the sender's edge and the receiver keeps track of the HTLC in a dictionary, storing details such as the amount and timestamp of the associated CLTV time-lock. The keys of this dictionary are IDs of the HTLCs which are generated using the UUID package in Python. The ID is also stored in the packet.

The total funds available to the two parties is now reduced, but this change is not publicly visible to other nodes who may be pathfinding. This is the reason why the initial funding of channels is also maintained in the states of the corresponding edges.

Receiving a payment If a node is the target of a payment and receives a *pay_htlc* packet, then the payment has reached its destination - provided there are enough available funds on the last hop. In practice, another HTLC is constructed before the receiver reveals the preimage to claim the funds. In the simulator, this step was skipped for simplicity and the balance sheet on the last hop is directly updated to reflect the payment. The packet type is then updated to *preimage* and sent back along the path. As before, if there are not enough available funds on the last hop, the payment is sent back with packet type *cancel_rest*.

Forwarding preimages Once a node receives a packet of type *preimage*, they can now claim funds from their associated HTLC. As such, their side of the payment channel is increased by the amount in the HTLC - the ID of which can be found in the packet's state. The packet is then forwarded on.

Payment receipt If the source node of the payment receives a *preimage* packet, the payment has now fully completed and the preimage serves as a form of receipt. The payment is marked as successful.

Teardown of HTLCs To cancel a payment, the HTLCs must be torn down in a reverse order by propagating a cancellation message towards the source. If a node receives a *cancel_rest* packet, it updates the packet type to *cancel* and forwards it. If a node receives a *cancel* packet, it claims the funds allocated in the HTLC before forwarding on back towards the source. Additionally, once the source node receives the cancellation packet, the payment is recorded as a failure.

The simulator allows nodes to delay the forwarding of cancellation packets. The delay can be any amount of time up until the time-lock of the HTLC has expired. This feature was disabled for the conducted experiments, primarily due to the considerable length of the average time-lock. By default, every CLTV time-lock in an HTLC is nine block confirmations longer than the HTLC ahead of itself in the path [4] - hence, the time-locks quickly get into the range of hours, much longer than the run-time of experiments. To simulate a delayed cancellation, instead of processing the packet as normal, the node changes the timestamp to account for the delay and the packet is put back into the packet queue. This can only happen once per hop of a payment - so the next time the packet is removed by that node, it will be processed as normal.

4.5 Atomic Multi-path Payments

When payments are initiated, they may be split into smaller partial payments. In practice, they may be split in any manner appropriate, but in the simulator they are split evenly into k parts. The simulator supports the re-sending of partial payments that fail but this feature is disabled for the experiments conducted as the experiments are used to investigate the likelihood of payments succeeding on the first attempt. There is a timeout in which the receiver expects to receive all partial payments, which is 60 seconds by default [24], initiated when the first partial payment is received.

To initiate a payment, a path is found for each partial payment and is then sent as normal. The ID of the overall payment, the index of the partial payment and the number of partial payments sent in total are included in each packet. The sender also keeps track of any already failed routes for each on-going partial payment.

Pathfinding It is difficult to determine the best pathfinding solution for multi-part payments. If a partial payment is sent down a route, then you know that the route has become depleted to a degree. In the simulator, we attempt to find completely independent paths. This is achieved by temporarily increasing the weights of edges used in previously found paths - so when running Dijkstra's algorithm, they are avoided. This detail is also included in Algorithm 1. Note that if k independent paths do not exist, this will prioritise paths that are partially independent or if necessary, re-use old paths. This is crucial because it means payments do not automatically fail if it cannot find k independent paths. The implementation of Dijkstra's algorithm uses a heap, so the computational complexity for finding k paths becomes $O(k(m + nlogn))$, where n is the number of nodes and m is the number of edges.

Algorithm 3 demonstrates how to call Dijkstra's algorithm for multi-part payments. When multi-part payments are initiated with k splits, the generator automatically yields k paths; thereafter, if re-sending of partial payments is enabled, the generator will retrieve the next available path that has not already been tried, if there are any.

An alternative algorithm that was originally implemented in the simulator is Yen's algorithm which is used in conjunction with an underlying cheapest path algorithm (Dijkstra's in this case) to find the K cheapest paths in a graph [28]. This differs from temporarily increasing weights of previously used edges because it does not try to find fully independent paths - its goal is to find the K cheapest paths, irrespective of independence. The computational complexity of Yen's algorithm is comparably higher however, emerging as $O(kn(m + nlogn)$. This meant pathfinding quickly became very slow and was not feasible for usage in this simulator.

Receiving a partial payment Once packets of partial payments have been sent out, intermediary nodes process them as normal. If a partial payment successfully reaches its destination, the receiving node stores the associated packet provided it is still within the deadline. If not, the packet is returned to the sender with type *cancel*, along with all other packets that have been stored for that particular payment. If all partial payments successfully reach the destination before the deadline, the receiver can now claim funds from each partial payment. The packets are then set to type *preimage* and sent back to the sender. Once the first partial payment has returned, the overall payment is recorded as a success.

Failed partial payments If a partial payment returns to the sender as failed, it can be re-sent if this feature is enabled provided the deadline has not surpassed. The failed path is recorded and the partial payment is re-initiated with a newly found path.

If re-sending is disabled, or the deadline has been surpassed, the payment is recorded as a failure whenever the first partial payment returns. Whenever

Algorithm 3: Find the next cheapest path available, or the first k cheapest paths.

```
function findPath ($src, tar, amnt, failedPaths, k = False$);
Input  : Source and target nodes $src$ and $tar$ and the amount of funds to be sent, $amnt$.
         A list of previously attempted failed paths, $failedPaths$, or an integer $k$
         (default: False) which specifies to return the first $k$ paths, ignoring $failedPaths$.
Output: A generator that produces a list of possible paths from best to worst, or False if
         none are found. If $k$ is specified, $k$ paths will be returned, otherwise 1.
$listA \leftarrow \{\}$;
$listB \leftarrow PathBuffer()$ ;            // Built-in NetworkX object - heap for paths.
if $\neg k$ then $toFind \leftarrow |failedPaths|$;
else $toFind \leftarrow k - 1$;
while $|listA| <= toFind$ do
    $avoidEdges \leftarrow \emptyset$;
    if $k$ then $toAvoid \leftarrow listA$;
    else $toAvoid \leftarrow failedPaths$;
    foreach $p \in toAvoid$ do
        $avoidEdges \leftarrow avoidEdges \cup p.edges$
    $pathAttempt \leftarrow$ dijkstraReverse($src, tar, amnt, avoidEdges$);
    if $pathAttempt$ then
        $path, length \leftarrow pathAttempt$;
        push($listB$, ($length, path$))
    if len($listB$) $\neq 0$ then
        $path =$ pop($listB$);
        $listA \leftarrow listA \cup path$;
        yield $path$;
    else
        return False;
```

a partial payment returns, the destination node releases any potentially stored partial payments with type *cancel*.

4.6 Just In Time Routing

To support JIT Routing, a re-balancing mechanism has been implemented. As previously described, the re-balance simply involves a circular payment where the source and destination node is the same. The aim here, is that the last hop of the payment will re-balance a payment channel that is too depleted. Some minor adjustments to how packets are processed is required to perform the re-balance at the correct time. No specific adjustments are required to integrate with multi-part payments.

Pathfinding An additional modified version of Dijkstra's algorithm was necessary to accommodate the re-balancing. As the source and target nodes are the same, the distribution of funds on the first and last hop of the payment are known. The path is also constrained to be of length three. Most importantly, the aim is to re-balance a particular payment channel - as such, this must be the last hop of the path returned. As the pathfinding is reserved, that particular payment channel must be the first hop when searching for a route. This can be observed in Algorithm 4.

In Section 3.6, we described an extension to the proposal which allowed the node to find the optimal path to re-balance by. To simulate the effect of this, a flag can be set to allow full knowledge of the distribution of funds on all three hops of the cycle. In hindsight, a better approach would have been to simply ensure the route found was the most likely to succeed, rather than granting full knowledge as this would not be the case in practice. Where no possible path exists, the simulator will not attempt a re-balance, whereas, in practice, a path destined to fail might be attempted. The only difference is one HTLC would be set up, which is minor.

Initiating a re-balance When an intermediary node receives a *pay_htlc* packet but cannot forward it on due to insufficient funds, it can then attempt a re-balance. Normally, this is the point in which the packet is sent back with type *cancel*. The simulator attempts to find a path that can re-balance the channel by exactly the correct amount to support the original payment. In practice, one may want to re-balance by a larger amount so the edge won't immediately be depleted to zero.

If a path that can potentially support the re-balance is found, the node sends the payment to itself in the usual fashion, but the packet contains the ID of the re-balance payment. Additionally, the node also stores the original packet in a dictionary where the key is the same ID as stored in the packet. This way, once the re-balancing has completed, the node can easily retrieve the original payment packet.

Processing of re-balance packets Once a destination node receives a payment marked as a re-balance, the ID stored in the packet can be used to retrieve the packet of the original payment from the node's state. Now, the original payment can be continued as normal. There is however a chance that while re-balancing, the edge in question could have been depleted even further. To combat this, re-balancing is continuously re-attempted. Note that this only happens if the previous re-balance was successful.

If a re-balance fails, the original payment is cancelled in the usual way. At this point, trying to re-balance again using an alternative path would be possible but this was not implemented in the simulator.

Algorithm 4: Dijkstra's algorithm adapted from NetworkX to find cycles length three for the Lightning Network.

```
function jitDijkstraReverse ($src, jitTar, amnt, fullKnowledge$);
Input  : A source node $src$ and another node $jitTar$ - where the edge between $src$ and
         $jitTar$ is depleted and to be re-balanced by $amnt$. A flag $fullKnowledge$ for
         enabling full visibility on all fund distributions.
Output: Cheapest cyclical path (by fees) between $src$ and itself, where $tar$ is the second
         last node in the path, and the fees paid on that path, or False if no path exists.
$src, tar \leftarrow tar, src$ ;                                        // Swap search direction
$dist \leftarrow \{\}$;
$fringe \leftarrow$ heapQueue();
$paths \leftarrow \{src : [src]\}$ ;                               // Maps nodes to cheapest paths
$seen \leftarrow \{src : 0\}$ ;                            // Maps nodes to total fees paid so far
push($fringe$, $(0, src)$);
$srcAlreadyPopped \leftarrow$ False;
while len($fringe$) $\neq 0$ do
    $d, v \leftarrow$ pop($fringe$);
    if $srcAlreadyPopped \wedge v \in dist$ then continue;   // Already searched this node
    if $srcAlreadyPopped \wedge v = src$ then break;                         // Cycle found
    if $v = src$ then $srcAlreadyPopped \leftarrow$ True ;
    $dist[v] \leftarrow d$;
    foreach $w \in$ successors($v$) do
        if $v = src \wedge w \neq jitTar$ then          // Must include hop from $src$ to $jitTar$
            continue
        if $v = jitTar \wedge \neg$edgeExists($w$, $src$) then        // Only cycles of length 3
            continue
        if $v \neq jitTar \wedge v \neq src \wedge w \neq src$ then         // 3rd hop must return to $src$
            continue
        if $v = src$ then $cost \leftarrow 0$ ;                                // No fees on last hop
        else $cost \leftarrow$ fees($wv$, $amnt + seen[v]$) ; // Reverse direction for fee rates
        $vwDist \leftarrow dist[v] + cost$;
        if $w \notin seen \vee vwDist < seen[w] \vee seen[w] = 0$ then
            if $((w = jitTar \vee w = src \vee fullKnowledge) \wedge amnt + cost <=$
             edgeFunds($w$, $v$)$) \vee (\neg(w = jitTar \vee w =$
             $src \vee fullKnowledge) \wedge amnt + cost <=$ channelFunds($w$, $v$)$)$ then
                $seen[w] \leftarrow vwDist$;
                push($fringe$, $(vwDist, w)$);
                $paths[w] \leftarrow paths[v] + [w]$;
if $|paths[src]| > 1$ then                                               // Full cycle found
    return reverse($paths[src]$), $dist[src]$;
else
    return False;
```

4.7 Testing

As with any piece of software, rigorous testing is necessary to ensure the system operates as designed and in a bug-free manner. To begin with, the testing of a non-split payment with re-balancing disabled is considered.

Network initiation The two graph topologies were generated by modifying code taken from the NetworkX library. The adaptions are very minor - the adjustments made are to ensure the nodes of the graph are instances of the custom *Node* class and that the edges are both directed and contain the necessary information regarding initial funding. As such, no specific testing was required here. The initialisation of both nodes and edges is straightforward as they are only assigned starting values dependent on the parameters set for the simulator.

To ensure the network is performing as intended, the generated test graph must be the same each time. The NetworkX implementations of both of these algorithms allow for the passing of a *seed*, which is an integer used for the random number generator. An arbitrary seed was selected for testing - in this case, 10.

Pathfinding & fees To path-find, the calculation of fees must first be considered. To unit test this function, a set of predefined paths and payment amounts is passed. The important factor here is that there must be no fees on the last hop, and it must be calculated in a reverse order.

To establish that the pathfinding implementations of Dijkstra's algorithm and breadth-first search are correct, various nodes are selected to initiate payments between, ensuring that every edge case is covered. To make sure the correct paths are being found, these tests are conducted on manually created networks, not the seeded graph described above. All tests are conducted for the shortest path, cheapest path, and highest average slack path. As this was mostly adapted code from the NetworkX library, the most important aspect to ensure was paths were not be travelled down that could not support the payment amount with fees included.

Packet processing during single payments The processing of packets is in many ways the most complex aspect of this project. To test this area thoroughly, a series of different payment scenarios are considered for the seeded graph. The tests are designed to consider all possible cases that can arise, especially edge cases, in hopes of a complete code coverage of the packet handling function. The primary concern here is that the allocated funds are being updated correctly and that packets are responding to the funds available in the correct manner - hence, the distribution of funds is compared to the expected values, both mid-payment and post-payment. The following payment scenarios have been examined:

- Single hop payments to neighbours: success and failure, no HTLC should be set up.
- Successful multi-hop payment: HTLCs should be set up at every hop once it reaches the destination, and the HTLCs settled as the preimage propagates back to the sender.
- Failed multi-hop payment: HTLCs should be set up as far as possible until the failure point, and torn down as the cancellation message propagates back to the sender.
- Correct cancellation initiation: if an HTLC is set up and the payment cannot be forwarded, the packet is changed to type *cancel*. If the HTLC could not be set up, the packet type is changed to *cancel_rest*. The difference depends on the timing when packets are processed.

Correct ordering of packets processed The above payment scenarios should be sufficient in assessing the timing of the system, but an additional set of tests were conducted to elevate the confidence of the packet processing order. These tests created packets between nodes that would result in a different processing ordering than in which they were created due to differing latency times.

Atomic Multi-path Payments The pathfinding algorithm for multi-part payments is tested to ensure independent paths were found where possible. As with non-split payments, various payment scenarios have been established to expose any faults with the handling of packets in the case of multi-part payments, both with and without the re-sending of packets enabled. In particular, once partial payments are sent out, they are treated like a non-split payment by intermediary nodes - the necessary testing is the packet handling at the source and destination.

- A multi-part payment where all k partial payments deliver correctly: HTLCs should be set up correctly, and preimages for any part only released once all have reached the destination.
- A multi-part payment where some partial payments reach the destination, but at least one fails. Once the failed partial payment returns, the partial payments received at the destination should be sent back as cancelled, tearing down all of the set-up HTLCs.
- Re-sending: payment with failing partial payments, that cause the payment to succeed when re-sent with new routes.
- Failure with re-sending: payment that fails even with re-sending available due to not being able to find an alternative path or is taking too long.

Just In Time Routing It is important to ensure the constraints imposed on the pathfinding algorithm designed for re-balancing are adhered to. Additionally, we assert that the most appropriate path is found when the knowledge of the third edge's funds distribution is available.

Continuing with the same testing methodology, the correct re-balancing and timing of such a re-balance is considered with the following set of payment scenarios. As there is no direct interconnection between JIT Routing and multi-part payments, meaning the re-balancing occurs without the knowledge of how the payment was initiated, testing for JIT Routing is covered in the case of a single non-split payment.

- Payment succeeds that would have otherwise failed without JIT Routing enabled.
- Payment fails as normal as no re-balance route could be found.
- Payment fails as normal as the attempted re-balance fails.
- Re-balance is successful, but the funds have been depleted even further while re-balancing, so another re-balance is attempted - tested with both successful and failed second re-balance attempts.

5 Experiments

To reiterate, the following experiments are used to investigate the various network conditions that cause payments to succeed or fail. All experiments are conducted 20 times to ensure that results are of high quality and free of noise. The experiments have been run on the Google Cloud Platform to complete all within the time constraints of this project. The default parameters for the simulator are formally presented as follows:

- G, a directed graph of 2,000 nodes with an average degree 10, where each edge represents an end of a payment channel.
- γ, the total number of payments that are sent throughout the experiment - 7,000.
- α, the maximum amount for a payment - 200,000 satoshi.
- β, the percentage chance to behave as the opposite role (merchant/consumer) - 0%.
- λ, the number of payments initiated per second - 50.
- ϵ, the percentage of "high" balances initiated on payment channel ends - 20%.
- k, the number of splits for each payment - 1.
- Search strategy - shortest path.
- Re-balancing - disabled.
- Epochs, the number of times each experiment was run - 20.

5.1 Network performance experiments

Before investigating multi-part payments or JIT Routing, several experiments have been conducted to evaluate the network's performance to see how well it scales and to help identify under what conditions the network breaks down.

Primarily, α, the maximum payment amount is modified to give insight into what to expect when sending payments of a higher value. These experiments also provide a baseline to evaluate multi-part payments.

Secondly, λ, the number of payments sent per second is adjusted to examine how active the network can be while maintaining usability and not becoming overly congested. Any in-progress payment may have a certain amount of funds locked up in HTLCs which reduces the funds available for other payments.

As it is difficult to measure the expected ratio of merchants to consumers on LN in practice, β is adjusted to see how the robustness of the network changes as the flow of funds becomes more evenly distributed. If $\beta = 50\%$, every node is equally likely to be selected as either a merchant or consumer - hence, the distribution is uniform.

Finally, the ratio of "high" to "low" starting edge funds may not be representative of how LN emerges in practice so we also modify ϵ in experiments. The default value for ϵ is taken from Miller et al. [13] who make use of an anonymised dataset of credit card transactions provided by a bank. As ϵ is increased, not only does the total available funds in the network increase, but the distribution of funds becomes more evenly distributed - at 50% there are as many high-value edges as low-value.

The following are the various choices of parameters used for these experiments, including the defaults.

- λ: 10, 50, 100, 250, 500, 1000
- α: 100000, 200000, 300000, 400000, 500000, 800000, 1000000
- β: 0%, 10%, 20%, 30%, 40%, 50%
- ϵ: 10%, 20%, 30%, 40%, 50%

The results of the experiments can be observed from Figure 4. In general, both scale-free and small-world networks yield the same trend in results.

Interestingly, the vast increase in transactions initiated per second (λ) does not disrupt the network as much as originally thought. While the increase in λ reduces the total funds available for subsequent payments, the drop in the number of successful payments is comparably low. We presume the relatively high degree nature of nodes mitigates this problem. Even when on average half of the participants in the network initiate a payment every second, still over 91% of payments succeed.

As expected, we observe a steady decline in payment success rate as the maximum payment amount (α) is increased. On average, each payment channel has a total capacity of two million satoshi - clearly sending half of that is not met with much success, but sending closer to 10% of this value shows great promise. The scalability of LN seems limited in terms of larger payment amounts, but in practice, a primary application of the Lightning Network is as a micropayment system. In this light, the results are promising. Large scale payments may be better suited to employing the blockchain directly, particularly as the fee rates on the blockchain are not affected by the payment amount.

The increase in chance to behave as the opposite role (merchant/consumer) correlates to an increase in payment success rate. This is expected as a 50% chance brings rise to a uniform distribution of payment start-points and end-points, so the network becomes less unbalanced.

Finally, as predicted, raising the percentage of edges that are initially funded with a higher amount (eight million satoshi) leads to a constant rise in payment success. This is primarily as the network now has a larger overall funding. For future work, it would be worthwhile exploring how this change behaves when the total network funding is kept constant - for example, as the percentage of high-funded edges increases, the amount assigned to each highly-funded edge is reduced.

5.2 Search Strategies

Payments are routed along the shortest path for the majority of experiments, but it is worth considering some alternatives to evaluate their performance. Each hop of a payment path holds an uncertainty in its success, so longer paths tend to be more likely to fail; however, for some, longer paths that cost less in fees may be more desired. In practice, many factors influence the quality of a path, but for simplicity, we have considered two other search strategies in conjunction with shortest path: *a*) Cheapest path, in terms of fees, and *b*) the path with the highest average "slack", which was previously explained in Section 4.3.

All three search strategies are compared in terms of payment success rate and average path length. The difference in compute time is negligible. To compare the cheapest and shortest path, the default fee rates must be changed. The distribution of fee rates used in these experiments can be observed from Table 1. They have been constructed to best represent the percentiles of chosen fee rates currently on LN.[11] For these experiments, a breadth-first search

[11] Source: `https://1ml.com` - Accessed Mar. 21, 2020.

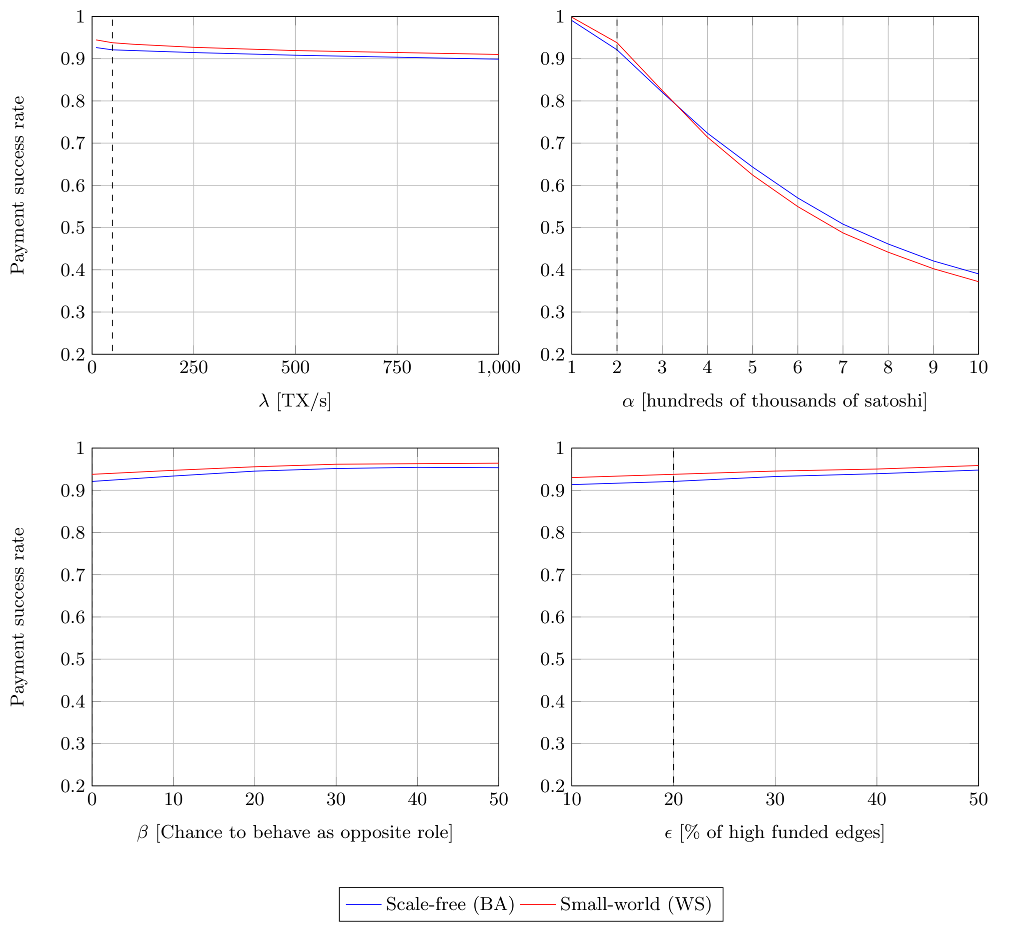


Fig. 4. Payment success rate for non-split payments under various network conditions, averaged over a total of 7,000 payments. The default parameters of the respective dependent variables are marked by the black dashed lines, which is zero or the y-axis in the case of β.

implementation is used to find the shortest path as the distribution of fee rates is no longer uniform.

Table 1. Fee rate distribution of the base rate **(left)** and the fee rate multiplier **(right)** that is assigned to edges of payment channels during experiments comparing search strategies.

Base Rate	Distribution
0	5%
0.0025	20%
1	70%
5	5%

Fee Rate	Distribution
0	5%
0.00001	45%
0.00055	25%
0.001	20%
0.003	5%

It is clear from Table 2 that the shortest path is a definite winner in terms of payment success rate. Cheapest path (in terms of fees) performs surprisingly poorly, despite on average being only about two hops longer. Each extra hop imposes a very high risk of failure; hence, finding the cheapest path does not prove worthwhile unless the current fee rates become substantially larger. On the other hand, while paths with the highest average slack tend to be even longer, they do not fall as short in success as the cheapest path strategy, especially in the case of the small-world graph topology.

It could also be argued that as edges are only initialised with one of two values, the slack-based pathfinding algorithm is unable to show its true performance. Every channel has only have one of three possible total funding amounts. This means the shortest path found has a higher chance of also holding the highest possible slack value. Also, the algorithm discourages paths that exceed the maximum hop distance. This aids performance not only due to the auto-failure of these long paths, but also due to the risk of including more hops. For future work, we propose calculating the length (number of hops), say l, of the shortest path, and then discourage paths that exceed a length of $l + \delta$, where δ is a relatively small integer, say 2.

Table 2. A comparison of different search strategies and their impact on payment success rate, along with the average hop length of paths found.

Graph	Search Strategy	Payment Success Rate	Average Hops Per Path
BA	Shortest path	0.92	4.23
WS	Shortest path	0.94	5.00
BA	Cheapest path	0.54	5.82
WS	Cheapest path	0.57	7.27
BA	Highest slack	0.69	8.52
WS	Highest slack	0.81	8.53

5.3 Atomic Multi-path Payments

A primary reason for utilising AMP is to route larger payments, so both k, the number of splits and α, the maximum payment amount are adjusted against each other to determine the performance of payments when they are split. Note that here α refers to the maximum payment amount of the entire payment, not the split amounts.

In Figure 5 an interesting observation arises - up until a certain point, payments perform better the more they are split, and thereafter they perform worse. This point is approximately the same for all experimented splits - about a sixth of the average total allocated funds to each payment channel. For very large payments, the reduction in performance as you increase splits is significant, but not severe. This is important as splitting up a payment may be the only option for a node.

With multi-part payments, a factor of k more hops are introduced that a payment is routed on. While the amount of funds being sent over every hop is lower, as they are split, the increase in risk from the number of hops seems to outweigh this. It is unclear why AMP behaves better for smaller payment amounts, but this further supports LN's application as a system for micropayments.

Therefore, large payments should not be split if a node wants to maximise their chances of a successful payment and have sufficient funds in a single channel; however, this might not always be the best practice as such a large payment may cause a significant depletion in the channel it is sent from.

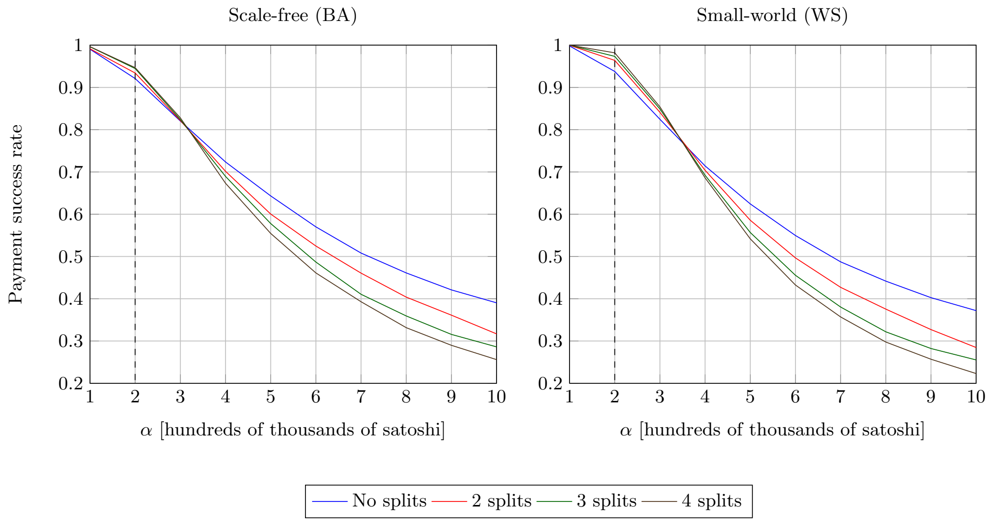


Fig. 5. The success rate of multi-part payments where the re-sending of partial payments is disabled. The default value for α is marked by the black dashed lines.

5.4 Just In Time Routing

To evaluate the impact of JIT Routing, α, the maximum payment amount is adjusted once again. In experiments, both the original protocol and the extended version (as proposed in this work) are considered. As a reminder, cycles are limited to length three when re-balancing. Furthermore, all experiments conducted for multi-part payments are also repeated with JIT Routing (original protocol only) enabled.

The benefits of adopting JIT Routing are observable from Figure 6, showing an increase in payment success rate of up to 7.88% in the small-world network. The extension to the protocol sees an even higher growth, albeit only slightly - it is arguable that the standard JIT Routing protocol is sufficient as it does not raise any privacy concerns.

Such benefits should motivate participants of the network to adopt this technique as it helps others route their payments which allows cooperating nodes to earn more fees. Additionally, if a specific re-balance would cause an undesired depletion in another channel, the node can simply decide against engaging in JIT Routing for that particular payment.

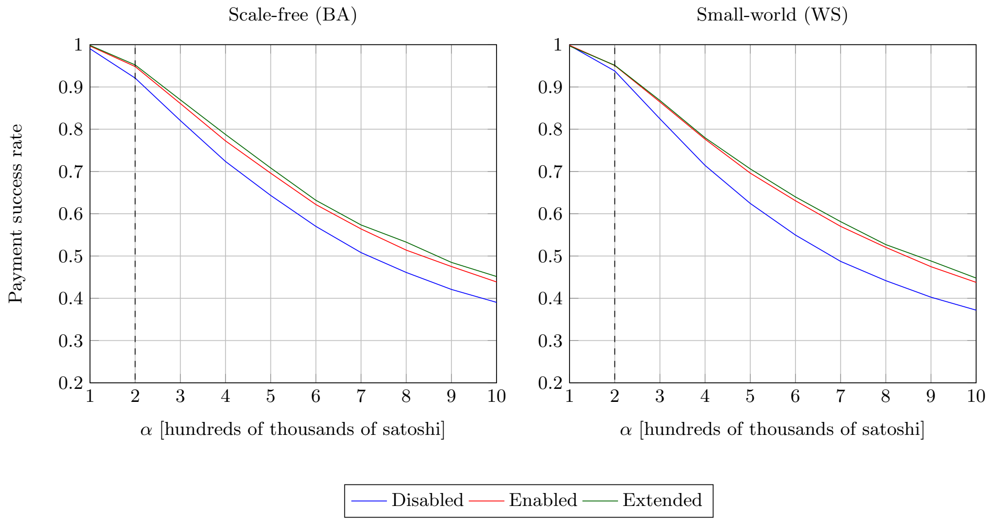


Fig. 6. The impact of Just In Time Routing on payment success rate, both with the standard JIT Routing protocol and the extended protocol (as described in this work). The default value for α is marked by the black dashed lines.

Graph stability over time The network tends to become more unbalanced over time causing the payment success rate to continuously decrease. To evaluate whether or not JIT Routing (original protocol only) keeps the network stable for a longer period, the simulator is left run until the average payment success rate drops below an arbitrarily chosen value, both with and without

JIT Routing enabled. The final value of γ, the total amount of payments sent is compared once the average payment success rate drops below 0.7.

The adoption of JIT Routing illustrates a significant increase in the time taken for the average payment success rate to drop below 70%. The scale-free network saw an increase from 25,843 to 29,063 payments successfully routed. This is almost a 12.5% increase in delivered payments. The small-world topology saw an even more considerable improvement, jumping from 28,857 to 35,905 payments when the standard JIT Routing protocol was enabled - an impressive 24.42% increase.

This demonstrates how this method of re-balancing can allow nodes to operate for longer without the need to close and re-open channels. This is significant due to the associated time and cost of closing and re-opening channels, the fees of which are likely to be notably higher than the accumulated re-balancing fees, which could be zero if free-free re-balancing [31] is also implemented.

AMP combined with JIT Routing In Figure 7, it can be observed that the benefits of JIT Routing still apply when payments are split. This behaviour is expected and the results are promising.

6 Conclusion

Undoubtedly, cryptocurrencies still face several challenges, most notably, its issues with scalability and transaction approval rates. Payment channels prove to be the most promising solution to these problems yet to surface, with Bitcoin's Lightning Network leading in popularity.

The experiments on LN revealed how well it can and cannot scale in certain areas. Concurrently routing more payments through the network did not result in an overly congested network, likely due to the high degree of nodes. The attempts to route larger payment amounts were not met with the same success however. In practice, we feel this is less significant due to LN's application as a micropayment system. The network also performed its best when the balance between consumers and merchants was more evenly distributed, but it's hard to estimate how this balance will emerge in practice.

Routing along the shortest path resulted in much of the success of the experiments. The scalability of LN is seemingly more fragile when using other search strategies, such as the cheapest path. Each extra hop of a payment increases the risk of failure by a significant amount. Currently, we do not view that as problematic as the fee rates of the network are so low that utilising a longer, cheaper path seems impractical.

Multi-part payments often resulted in a lower success rate - most likely due to the increase in the total hops travelled for each payment. The reduction is significant, but for scenarios where splitting payments may be the only option, they are still met with a degree of success. This further discourages the sending of larger payment amounts on LN - in this case, employing the blockchain directly may be a better option. Lower payment amounts performed better when split, further supporting LN's application as a micropayment system.

Our findings support the adoption of JIT Routing by all network participants, resulting in a significant increase in payment success. This is valuable for both the sender of the payment, and the node engaging in JIT Routing, as it allows them to earn more fees. Above all, it allows the network to operate for longer with less of a need to close and re-open unbalanced channels.

In general, the final simulator performed excellently and as required, providing high confidence in results; although it was not without its flaws along the way. Primarily, how the timing of the simulator was handled coupled with the initiation of new payments evolved in many ways throughout the course of the project. While the core functionality of the system, such as sending and receiving payments was assured through rigorous testing, the flaws accompanied by the simulation aspect of the project were less obvious. Above all, it proved to be a tremendous learning experience into the approach of empirical research and experiments, discovering what questions should be asked and how should one approach answering them.

The importance of studying publications became evident as the project progressed. Much misunderstanding of how the Lightning Network operates could have been avoided at an earlier point in time by analysing the relevant publications where necessary, such as the initial white paper [21].

6.1 Future Work

We have previously mentioned two items left for future work concerning an extension to our slack-based pathfinding algorithm, and maintaining a constant total network funding when adjusting ϵ, the percentage of initial high-value edges. The following further extensions to our simulator are outlined below.

Offline and unresponsive nodes As mentioned in Section 2, LN discourages nodes from going offline. The inclusion of nodes randomly going offline could benefit the experiments two-fold: by interrupting payments, and by reducing the available search space for possible routes for a payment. Additionally, while unresponsive nodes are possible given the current simulator, this could be extended to include unresponsiveness in the outward direction of a payment

- currently, nodes can only delay a cancellation. The previously mentioned issue of nodes having too much time to cancel could be mitigated by scaling down the response time.

Disappearing and appearing channels It would also be worthwhile to investigate what happens when channels close, re-open or even completely new channels becoming available. The current network becomes more unbalanced over time, and the success rate of payments constantly decreases. An insightful experiment would be to evaluate the conditions of channels appearing and disappearing which lead to a stable network that can run indefinitely.

Funds and payment distributions The initial funding amounts distributed to channels are very limited in diversity and likely not representative of a real-life scenario. Using snapshots of the Lightning Network to model the graph and the capacities of each channel would be a better approach. How merchants and consumers are selected is uniform within their respective groups for the conducted experiments - yet the simulator supports varying this distribution. For future work, it would be beneficial to find a representative distribution of payments.

Captured statistics The majority of experiments completed measured the fraction of successful payments over a predefined total number of sent payments (almost always 7,000). Perhaps a more practical measure would be throughput - where we capture the rate at which payments succeed over given time intervals and contrast it to the number of new payments initiated per second.

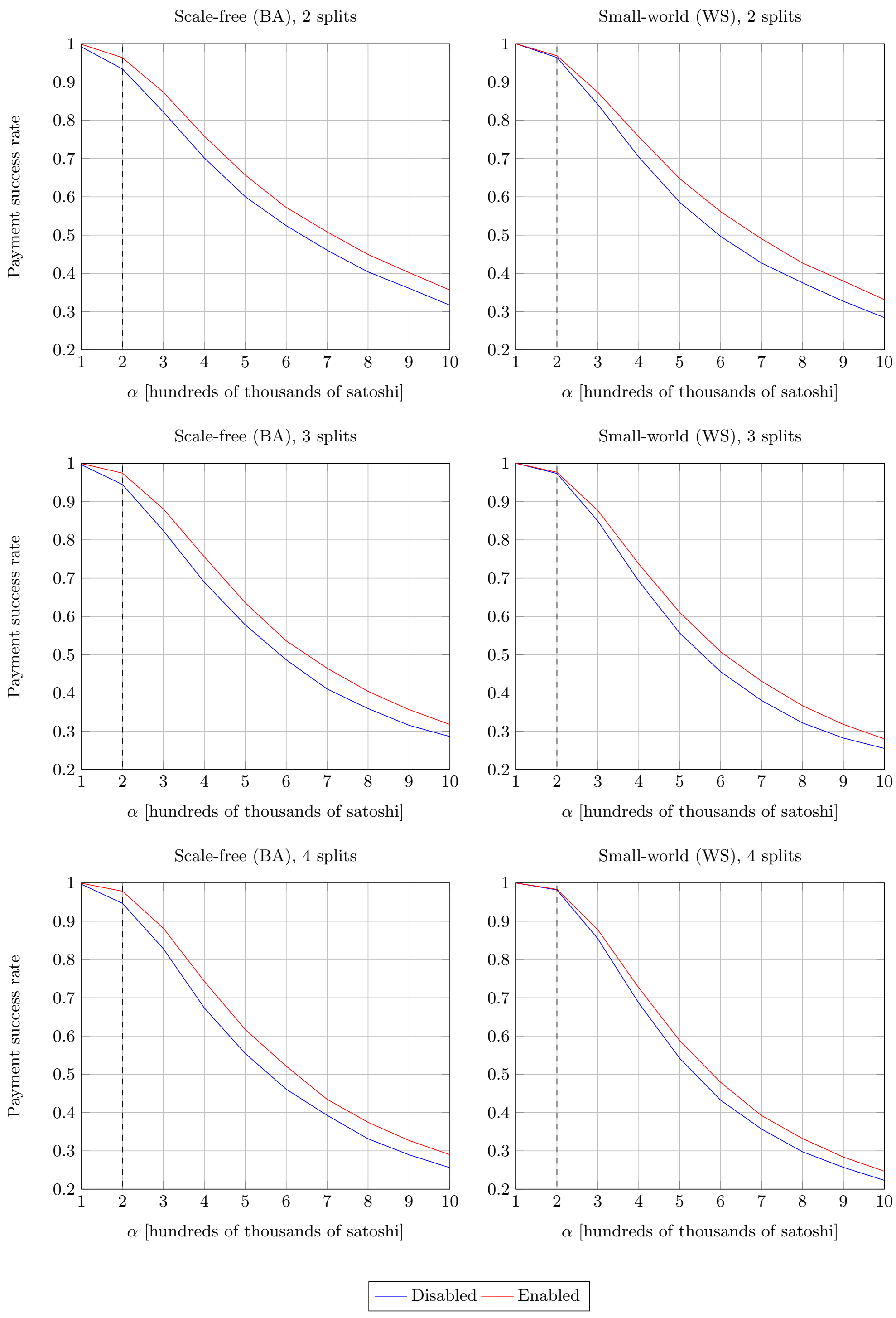


Fig. 7. The impact of JIT Routing (original protocol) on multi-part payments. The default parameters of the respective dependent variables are marked by the black dashed lines.

References

[1] R. Albert and A.-L. Barabási, “Statistical mechanics of complex networks,” *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[2] A.-L. Barabási and E. Bonabeau, “Scale-free networks,” *Scientific american*, vol. 288, no. 5, pp. 60–69, 2003.

[3] F. Béres, I. A. Seres, and A. A. Benczúr, “A cryptoeconomic traffic analysis of bitcoins lightning network,” *arXiv preprint arXiv:1911.09432*, 2019.

[4] (2020). Bolt #2: Peer protocol for channel management, [Online]. Available: `https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md/` (visited on 03/30/2020).

[5] (2020). Bolt #4: Onion routing protocol, [Online]. Available: `https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md/` (visited on 03/30/2020).

[6] C. Burchert, C. Decker, and R. Wattenhofer, “Scalable funding of bitcoin micropayment channel networks,” *Royal Society open science*, vol. 5, no. 8, p. 180 089, 2018.

[7] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, *et al.*, “On scaling decentralized blockchains,” in *International conference on financial cryptography and data security*, Springer, 2016, pp. 106–125.

[8] C. Decker and R. Wattenhofer, “A fast and scalable payment network with bitcoin duplex micropayment channels,” in *Symposium on Self-Stabilizing Systems*, Springer, 2015, pp. 3–18.

[9] C. Grunspan and R. Pérez-Marco, “Ant routing algorithm for the lightning network,” *arXiv preprint arXiv:1807.00151*, 2018.

[10] A. Kate and I. Goldberg, “Using sphinx to improve onion routing circuit construction,” in *International Conference on Financial Cryptography and Data Security*, Springer, 2010, pp. 359–366.

[11] R. Khalil and A. Gervais, “Revive: Rebalancing off-blockchain payment networks,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 439–453.

[12] S. Martinazzi and A. Flori, “The evolving topology of the lightning network: Centralization, efficiency, robustness, synchronization, and anonymity,” *PloS one*, vol. 15, no. 1, e0225966, 2020.

[13] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, “Sprites and state channels: Payment networks that go faster than lightning,” in *International Conference on Financial Cryptography and Data Security*, Springer, 2019, pp. 508–526.

[14] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Manubot, Tech. Rep., 2008.

[15] T. Neudecker, P. Andelfinger, and H. Hartenstein, "Timing analysis for inferring the topology of the bitcoin peer-to-peer network," in *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, IEEE, 2016, pp. 358–367.

[16] M. E. Newman, S. H. Strogatz, and D. J. Watts, "Random graphs with arbitrary degree distributions and their applications," *Physical review E*, vol. 64, no. 2, p. 026 118, 2001.

[17] O. Olaoluwa. (2018). Amp: Atomic multi-path payments over lightning, [Online]. Available: `https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html/` (visited on 04/01/2020).

[18] D. Piatkivskyi and M. Nowostawski, "Split payments in payment networks," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, Springer, 2018, pp. 67–75.

[19] R. Pickhardt. (2019). Just in time routing (jit-routing) and a channel rebalancing heuristic as an add on for improved routing success in bolt 1.0, [Online]. Available: `https://lists.linuxfoundation.org/pipermail/lightning-dev/2019-March/001891.html/` (visited on 04/01/2020).

[20] R. Pickhardt and M. Nowostawski, "Imbalance measure and proactive channel rebalancing algorithm for the lightning network," *arXiv preprint arXiv:1912.09555*, 2019.

[21] J. Poon and T. Dryja, *The bitcoin lightning network: Scalable off-chain instant payments*, 2016.

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

[23] R. Russell. (2019). Increasing fee defaults to 5000+500 for a healthier network? [Online]. Available: `https://www.mail-archive.com/lightning-dev@lists.linuxfoundation.org/msg01484.html/` (visited on 03/28/2020).

[24] (2019). Slp134 rusty russell - lightning multi part payments, [Online]. Available: `https://stephanlivera.com/episode/134/` (visited on 03/30/2020).

[25] J. Spilman. (2013). Anti dos for tx replacement, [Online]. Available: `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html/` (visited on 04/01/2020).

[26] M. Trillo. (2013). Stress test prepares visanet for the most wonderful time of the year, [Online]. Available: `https://www.visa.com/blogarchives/us/2013/10/10/stress-test-prepares-visanet-for-the-most-wonderful-time-of-the-year/index.html/` (visited on 04/01/2020).

[27] M. Wahbi and K. N. Brown, "The impact of wireless communication on distributed constraint satisfaction," in *International Conference on Principles and Practice of Constraint Programming*, Springer, 2014, pp. 738–754.

[28] J. Y. Yen, "Finding the k shortest loopless paths in a network," *management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[29] R. Zivan and A. Meisels, "Message delay and discsp search algorithms," *Annals of Mathematics and Artificial Intelligence*, vol. 46, no. 4, pp. 415–439, 2006.

[30] ZmnSCPxj. (2018). Base amp, [Online]. Available: `https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-November/001577.html/` (visited on 04/02/2020).

[31] ——, (2019). Fee-free rebalancing to support jit-routing, [Online]. Available: `https://lists.linuxfoundation.org/pipermail/lightning-dev/2019-July/002055.html/` (visited on 04/06/2020).