RUHR UNIVERSITÄT BOCHUM

RUHR-UNIVERSITÄT BOCHUM

Price of Anarchy in the Lightning Network

Sebastian Alscher

Master Thesis – March 20, 2023.
Chair for Information Security.

First supervisor:	Prof. Dr. Ghassan Karame
Second supervisor:	Prof. Dr. Jörg Schwenk

InfSec

Abstract

When sending payments in the Bitcoin Lightning Network, sending nodes choose their optimal payment delivery strategy to find a flow for the payment. The negative impact of choosing selfish strategies on the reliability of the payment network – its ability to successfully deliver payments – can be expressed in the Price of Anarchy. We define the Price of Anarchy as the ratio of the failure rates of the optimal selfish strategy for participants and the failure rate when participants follow a cooperative strategy. To ascertain the existence of a Price of Anarchy for such measure and establish its value, we simulate non-cooperative as well as cooperative payment delivery strategies in a model of the Lightning Network.

Our findings of the simulation show that depletion of channels, and thus payment failure, happens faster with payment delivery strategies based on fees compared to probability based payment delivery strategies, in particular multi-part payments. The strategy with the lowest failure rate is when learning or sharing of information about success or failure of payments is shared with a central coordinator or participants. We conclude that a Price of Anarchy exists under the measure of payment reliability, and we calculate the lowest possible Price of Anarchy with the results of our simulation as 1.5456. Our findings show furthermore that the failure rate of payments increases over time for selfish payment delivery strategies, and that betweenness centrality in the Lightning Network has a significant impact on the failure rate.

Affidavit

I declare that I have not already submitted a paper in the same or similar form for another examination at the Ruhr-Universität Bochum or another university.

I affirm that I have written this paper independently and that I have not used any sources other than those indicated. The passages that are taken from other sources in terms of wording or meaning, I have identified the sources. This also applies analogously to drawings, sketches, pictorial representations and the like used.

I also assure that the written version submitted by me corresponds to the digital version. I agree that the digital version of this work may be used for the purpose of checking plagiarism.

Eidesstattliche Erklärung

Ich erkläre, dass ich keine Arbeit in gleicher oder ähnlicher Fassung bereits für eine andere Prüfung an der Ruhr-Universität Bochum oder einer anderen Hochschule eingereicht habe.

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen, die anderen Quellen dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen kenntlich gemacht. Dies gilt sinngemäß auch für verwendete Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen.

Ich versichere auch, dass die von mir eingereichte schriftliche Version mit der digitalen Version übereinstimmt. Ich erkläre mich damit einverstanden, dass die digitale Version dieser Arbeit zwecks Plagiatsprüfung verwendet wird.

Date	Signature

I would first like to thank my supervisor Prof. Dr. Ghassan Karame of the Faculty of Computer Science at Ruhr-University Bochum. Prof. Karame was always there, whenever I had a question about my research or writing. Thank you for being supportive and open to this topic.

I am grateful to René Pickhardt for inspiration to this research question, the stimulating conversations during the past year, access, insights and guidance, regular challenges and objective critique.

I would also like to thank Summer of Bitcoin for introducing me to open source-development and providing me with mentorship of established bitcoin and lightning developers.

I must express my very profound gratitude to my family for continuing support and patience. I could not have undertaken this journey without you.

Contents

1 Introduction

The Lightning Network is a decentralized system where payments are sent over a network of micropayment channels. It is based on the bitcoin blockchain where the consensus of the balances of bitcoins in wallets is agreed upon. Given its character, the bitcoin blockchain cannot register an equally high number of payments per second, as would a current payment network. Different possibilities exist that could increase the maximum payment volume, like an increase in block size or more centralization. But for bitcoin to succeed, it needs to maintain its current advantages of for example decentralization. Using a network of micropayment channels linked to the bitcoin blockchain is a proposed suggestion to scale to billions of payments per day.

1.1 Motivation

However, to be used as such a payment network the Lightning Network needs to fulfill its promise to reliably send payments over its network of micropayment channels. It needs a minimum degree of reliability to acquire enough acceptance so that it can present itself as a viable alternative to current payment networks and electronic payment systems. Reliability refers to a high degree of confidence that payments between two counterparts don't fail.

Unfortunately this is currently not the case. The Lightning Network at the moment cannot provide the reliability needed to be regarded as an alternative to centralized payment networks.

Its unreliability originates in nodes not being able to send or receive payments. The main reasons for this state are nodes not being reachable and channels not being able to forward the expected amount. While the availability of nodes is more of an operational problem, providing enough liquidity to deliver a payment is generally a question of distribution of channel balances, which can be dealt with on a protocol level and on an implementation level of a lightning node.

When a payment is sent from sender/source to receiver/sink the sending node chooses the payment flow, which is dissected into paths. The flow of the payment describes the graph of lightning nodes and channels for the respective payment. Which flow is chosen is determined by criteria implemented in the lightning node software. Several strategies exist for flow selection.

Each payment changes the state of channels in the lightning network. This is because each successful payment changes the balances in channels and could deplete channels, making them unavailable for a future payment flow between the two nodes. In consequence, the path selection decision impacts the state of the Lightning Network and its reliability.

The flow of the payment from sender to receiver is chosen by the sending node according to the preferences implemented in the node software. These preferences are for example to minimize cost or maximizie the payment success probability. They are at the discretion of the sending node and generally don't follow the payment network's best interest. The sender aims to optimize the choice of the payment flow exclusively to its own interest. When routing payments through the network follows the optimum with regards to the sender's preference function, this is referred to as selfish routing [Rou05].

Optimizing one's payment delivery for an individual single payment is detrimental to the network as a whole, because for example using the path with the lowest fees promotes draining liquidity in the cheapest channels in the network. This will in return lead to unusable channels and a cut in the network more frequently. Those payment channels cannot relay the payment amount anymore and no path might exist to pass the payment from sender to receiver.

Assuming other strategies exist, even though they might not be followed by network participants, these might avoid such network deficiency and even create a higher welfare for the network. The higher welfare originates in less channels being depleted and thus the network being usable more broadly.

The existence of these strategies could allow for calculating the cost of selfish routing. If a quantitative measure for the inefficiency can be derived, this measure is referred to as the Price of Anarchy.

When the cost of the different strategies is measured as unreliability of the network, the Price of Anarchy can represent a measure of reliability of the network, as it is calculated as the relationship between cost from selfish routing and cost of the most beneficial strategy for the network.

1.2 Contribution

This thesis gives an overview of existing payment delivery strategies of lightning nodes and analyzes them in the context of how beneficial these strategies are for the Lightning Network. One measure of how much worse failure rates are when participants use strategies that they consider the most beneficial for themselves versus what optimum could be achieved with different strategies, for example those that require cooperation, is the Price of Anarchy. We implement a model for the Lightning Network and possible strategies to empirically establish a Price of Anarchy as a measure of how much welfare is lost by selfish strategies.

The Price of Anarchy has been discussed in flow networks in particular in the context of network traffic and routing [Rou05], or in relation to topology and network creation games [AHWW19]. We extend the previous discussions by applying the idea of a Price of Anarchy to a particular flow network, the Lightning Network, where the participants don't have enough information about liquidity constraints in network channels. We test a model Lightning Network for different payment delivery strategies and the existence of a Price of Anarchy.

We contribute the code, which has been developed for the implementation of the probabilistic payment delivery in our simulation, to the open source library for Pickhardt Payments [Pic23].

1.3 Organization of this Thesis

This thesis analyzes the cost of selfish routing in the Lightning Network. Initially we describe the Lightning Network and provide a formal model. We also examine the payment flow construction and the delivery mechanism for the most common strategy, optimizing for minimal fees, and an alternative probabilistic model. Subsequently we describe the concept of the Price of Anarchy and how it applies to payment delivery strategies in the Lightning Network. We investigate, if a Price of Anarchy exists and how it can be defined and measured.

We then test the findings against a model of the Lightning Network, using a simulator we developed. The results of this simulation will help us demonstrate the persistent problems in the Lightning Network and to what extent the Price of Anarchy can be a measure for the loss of welfare of the network.

2 The Lightning Network

In this chapter we will outline the structure of the Lightning Network and formalize the Lightning Network graph. We explain the requirements of a payment and that a payment is a flow on the graph. We also show how the cost for uncertainty to successfully deliver a payment can be calculated. Finally we present the steps necessary to send a payment from a sending node to the receiving node.

2.1 Bitcoin and the Lightning Network

The bitcoin blockchain is a system for electronic transactions without relying on trust [Nak09]. It is a gossip protocol whereby all state modifications to the ledger are broadcast to all participants. It is through this "gossip protocol" that consensus of the state, everyone's balances, is agreed upon [PD16]. The bitcoin blockchain can serve as a source of a universal truth as well as an arbiter when counterparties involved in exchanging bitcoin for other goods are in disagreement.

The Lightning Network evolved in 2015 as a solution to transact more quickly and allow for more capacity, because the number of payments registered in the bitcoin blockchain would not be sufficient to establish bitcoin as a more broadly adopted means of payment. The Lightning Network can be described in two ways: On the one hand as a peer-to-peer network consisting of payment channels, which are implemented as smart contracts on the bitcoin blockchain. On the other hand the Lightning Network can also be regarded as a communication protocol that defines how participants set up and execute these smart contracts. [AOP21]

The payment channels are known as Poon-Dryja channels. One of the channel partners opens a channel by funding the channel and sending an amount of bitcoin to a 2-of-2 multi signature address. This is called the funding transaction. The funding transaction sets the financial capacity that is bound in the channels. This will be referred to as the *channel capacity.*

The channels share the capacity of the initial funding transaction amongst each other. Once some amount is sent from the sender to the receiver, the liquidity is reduced at the sender's side. This transmitted amount now can be used to send payments from the receiver to the sender.

Most lightning nodes implement pathfinding strategies that optimize purely for fee rates. It is a popular opinion even amongst lightning developers that finding optimal flow in the Lightning Network should be guided by fee rates. This represents the belief in an efficient market where all participants instantaneously adjust fees in correspondence to available liquidity in the channels they manage. As was discussed on the lightning developer mailing list "feerates (...) are always going to be something that senders optimize for, because obviously senders will have a maximum amount they will be willing to pay in fees" [Zmn22a] and following a particular understanding of market mechanics "fees gets [sic] you basic economics of supply and demand, and is a natural throttle in all markets, including liquidity markets." [Zmn22b].

Even though failure rates of payments are an obvious problem for Lightning Network participants, this perception leads lightning node operators to sticking with solutions by „better“ or „faster“ adjustment of fees, to avoid channels being unusable. This practice has lead to a trial and error attitude when it comes to solving a quite fundamental deficiency, and has not yet been addressed from a research and systemic perspective. Consequently, there has been very little research on the Lightning Network flow problems in the past.

2.2 The Network - Defining the Problem

The Lightning Network is a network of channels with the participating nodes V and the payment channels as edges E. More precisely, the Lightning Network is a directed multigraph with pairs of channels between nodes, one for each direction, described by a directed graph $G = (V, E)$. With two nodes v_s and v_t and a channel pair $e_{s,t}$ and $e_{t,s}$ between the nodes, there exists a function l, assigning the liquidity to each edge e in the graph.

$$l : E \longrightarrow \mathbb{N}_0$$

The liquidity is split up in a set of back and forth channels between two nodes. It is constraint to the amount of the funding transaction, the *channel capacity*. The relationship between the channel capacity cap and the liquidity $l(e_{s,t}) = l_{s,t}$ of edge $e_{s,t}$ can be defined as

liquidity split: $l_{s,t} = cap_{s,t} - l_{t,s} = cap_{t,s} - l_{t,s}$

The channels share the capacity of the initial funding transaction amongst each other. Once some amount is sent from the sender to the recipient, the liquidity is reduced at the sender's side and increased on the recipient's side. This means that the liquidity $l_{s,t}$ of the channel $e_{s,t}$ is reduced by the amount m and liquidity $l_{t,s}$ in the channel $e_{t,s}$ is increased by the amount m.

Because the sending node only knows the exact amount of liquidity in its own outgoing and inbound channels, it has to guess liquidity in the rest of the network. This leads to uncertainty where payment flows on channels can fail, because when constructing the flow the sending node might choose edges that do not carry enough liquidity. Consequently, the Lightning Network is inherently unreliable. This creates uncertainty in the network and decreases its overall usefulness.

However, a payment can fail not only because of a sub-optimally chosen payment flow. An at least equally grave problem arises when a set of payments is analyzed, instead of one single payment. For the Lightning Network to operate as a payment network it needs to be able to operate and execute a significantly larger number of payments than just one. This means that it is not enough to find a solution for a single payment to be delivered with high reliability, but it is also necessary to analyze the effect and interaction of a large number of payments.

Sending nodes take decisions when structuring the payment onion and aim to successfully execute their single payment. To what extent do their strategies deplete liquidity along edges that are crucial to workings of the network and as a consequence create a cut, which makes other payments fail?

2.3 Dynamics of a Payment

Every payment has a sending node (source) and a receiving node (destination) and an amount m that is passed from source to destination.

2.3.1 Requirements

Looking at what constitutes a payment one can find that the source has an excess supply of liquidity, the destination an excess demand. For each node along the flow the liquidity into the node should equal the liquidity out of the node plus an amount c as fees that the node retains.

For every node $v \in V$ let $b_v \in \mathbb{Z}$ denote its excess supply or demand. Typically b_v will be c_v except for the source node s with supply ($b_s > 0$) and the destination d with demand ($b_d < 0$).

Payments on the Lightning Network are flows on a finite graph. The flow is determined after solving an optimization problem, and is given by the function f, which represents the change in liquidity on the edges of the network graph.

We call a function $f : E \longrightarrow \mathbb{N}_0$ a *flow*, if the following conditions hold:

capacity constraint: For every edge $e \in E$ and $cap(e)$ the capacity of an edge we have:

$$0 \leq f(e) =: f_e \leq cap_e := cap(e)$$

and

conservation of flows: For every node $v_j \in V$ and the edges into the node $(i, j) \in E$ and edges out of the node $(j, k) \in E$ we have:

$$\sum_{(i,j)\in E} f_{ij} - \sum_{(j,k)\in E} f_{jk} = b_j + \sum_{(i,j)\in E} c_{ij} - \sum_{(j,k)\in E} c_{jk}.$$

with $b_j = 0 \, \forall \, v_j \notin s, d$ with s being the source and d being the sink.

Therefore the difference between all flows into a node and all flows out of a node is the balance plus costs. When a node is a source, meaning it sends money, then it has a positive balance b, and as a destination it has a negative balance b.

For payments from source to destination, the Lightning Network uses source routing which means the source node is calculating the entire route, from source to destination.

Each payment can be defined as a flow, derived from a set of two nodes and its amount. A payment P can be broken up into several payments. This is referred to as a multi-part payment, MPP. A payment P is a set of MPPs, where each MPP is an acyclic directed graph from s to t with amount f_i

$$P = \{MPP_1, MPP_2, MPP_3, \dots \mid MPP_i : f_i(e)\}$$

A flow can be one singular acyclic directed graph from source to destination or a set of acyclic subgraphs for an amount f_i where $\sum f_i = m$.

The information to construct the flow that is used for the payment – information on all public payment channels and their capacities – is taken from the gossip protocol.

2.3.2 Considerations Regarding Fees

In the Lightning Network nodes are compensated for relaying payments by having the possibility to charge a fee. The fee is usually far less than one percent of the payment amount. Usually the main factors to be taken into account when solving the optimization problem to find the route for the package, fees paid are one factor. Reliability of the network and the uncertainty cost in case of payment failure are another. However, because there will be no fees if a payment fails and considering the low amount of fees, the network's reliability has a significantly higher weight in our consideration regarding optimization than fees. Consequently we will not consider fees as transaction costs for the following analysis.

2.3.3 Constructing the Payment Package

When the sending node constructs the data for the network to initiate the flow, it constructs one or more onion packages. A sequence of hops is put together, linking channels from sender to receiver. The packet structure is defined in the BOLT4 specification. BOLT4 defines the packet structure for payments to hold a 1300-byte `hop_payload` section consisting of multiple, variable length, `hop_payload` payloads or up to 20 fixed sized legacy `hop_data` payloads. Thus, a singular acyclic directed subgraph contains at most 21 edges, as can be derived from the protocol. [Tea23b]

When constructing the payment flow, the sender has to make assumptions about the availability of a node and the liquidity of its edges.

Availability means the ability to reach the node at all times during the lifetime of a payment and have it reply according to the protocol.

Liquidity refers to the available amount of satoshis in the inbound and outbound edges. One satoshi (sat) is the 100 millionth of a bitcoin (BTC).

For the payment to be successful, the nodes along the flow need to be available and provide enough liquidity to cover the payment amount including the fees.

Thus, in addition to the information on the graph provided by the gossip protocol, a "belief network" exists with the sending node, which carries the assumptions on the availability of nodes and edges when constructing the onions for the payment.

2.3.4 Uncertainty Cost

A feasible flow of amount m needs to be found, so that the payment can be delivered. A feasible flow is a flow along edges that carry a minimum liquidity $l_e \geq f_e$. Having an unsuccessful payment leads to costs for the sender as well as for the network as a whole. The sender is punished by not having the payment delivered and/or having liquidity stuck in payment channels, until the hashed timelock contracts that serve to reserve liquidity in the channel are unstuck.

Hashed timelock contracts, in short HTLCs, are part of the Lightning Network Specification (BOLT2). HTLCs are unconfirmed transactions that are created when a node initiates a payment or forwards a payment to another node. The contract guarantees the relaying node to receive the flow from the sending node in exchange for the correct secret. Usually the incoming HTLC cannot be redeemed unless the outgoing HTLC can be redeemed. This allows the sender to chain channels that deliver the payment with the promise to pay them. This promise leads to the nodes retaining the liquidity, so that they can push it to the other node once they receive the payment preimage, which they can use to claim the inbound liquidity from the other node along the chain. So for as long as there are unresolved HTLCs, liquidity is frozen in the network and cannot be provided to others.

Stuck liquidity creates opportunity costs that can be quantified on an individual level. It also creates negative effects for the network, because channels do reserve liquidity in the channel until the preimage is received. This liquidity cannot be provided to others, which leads to the network suffering from less liquidity and in the worst case from a cut in the network graph. In consequence, the uncertainty about a payment to be delivered or not is a cost. This cost decreases with an increasing probability to successfully settle the payment.

This uncertainty cost depends on the probability for a channel to successfully relay an amount m from one node to the adjacent node and can be calculated. Uncertainty costs decrease with the probability of l_e being larger than or equal to f_e and increase with the probability of failure, l_e being less than f_e. Information about the success probability can be stored and used when calculating the optimal flow for a payment. This estimate about the current belief of the liquidity of each edge is held in a belief network. In addition to the belief of the liquidity of each edge it also holds a belief about the probability distribution of balances in the payment network.

Because the flow can consist of more than one edge, requirement for a successful payment is

$$l_e \geq f_e \forall e \in E.$$

The payment fails if any of the edges' liquidity is less than the payment amount and a node cannot pass on the flow through the next edge and cannot relay the payment. Consequently the probability of failure is 1 - the product of success probabilities of all edges, assuming an independent distribution.

The probability P_e for a single channel e to successfully relay a payment of size f_e for a random variable X_e can be defined as

$$P_e(X_e \geq f_e)$$

and the probability for an edge e to carry a balance less than flow f_e, the failure probability, is

$$P_e(X_e < f_e),$$

which for a uniform distribution leads to

$$P(X < f) = \sum_{x=0}^{f-1} P(X = x) = \sum_{x=0}^{f-1} \frac{1}{cap + 1} = \frac{f}{cap + 1}$$

A successful payment on an edge e is a payment that doesn't fail

$$P_e(X_e \geq f_e) = 1 - P_e(X_e < f_e) = \frac{cap_e + 1 - f_e}{cap_e + 1}. \tag{2.1}$$

The success probability of the flow is then defined as

$$P_f = \prod_{e \in E} P(X_e \geq f_e).$$

The goal to maximize the success probability can be translated into minimizing the negative logarithm of this probability, because the logarithm is a group homomorphism from the multiplicative group of positive real numbers to the additive group of real numbers:

$$-\log\left(\prod_{e \in E} P(X_e \geq f_e)\right) = \sum_{e \in E} -\log(P(X_e \geq f_e))$$

This sum of logarithms has the form of a separable cost function C for the flow f, which is a useful characteristic when regarding min-cost flows. We use this term to define our Uncertainty Cost:

$$C(f) := \sum_{e \in E} -\log(P(X_e \geq f_e)) \tag{2.2}$$

(see [PTBN21]).

So that in the Lightning Network a payment flow can be found with the goal to minimize uncertainty costs, a node calculates the likelihood for an amount f to be delivered successfully. This can be achieved by assigning the negative logarithm of the success probability as cost to the edges of the graph [PR21]).

The estimate of the success probability can be improved by retaining knowledge about the liquidity of edges that have been probed by sending a successful payment onion along this edge or from information about a failing onion. From a failing onion one can learn – and update the belief network accordingly – that the liquidity balance is less than the amount that was tried to channel along this edge. The difference between the channel capacity and this amount then by definition has to be in the return channel. From a successful delivery of the onion one can learn that the return channel now carries at least this amount of liquidity, while the maximum amount of liquidity in the channel itself is now the channel capacity minus the transferred amount.

2.3.5 Construction of the Flow

Taking a look at payments that are delivered on a **purely fee-based strategy**, it becomes immediately clear that payments will fail at some point, because all money is routed along the cheapest channels. They get saturated and become unavailable, if there are no payments offsetting this drain. Failure occurs at the first edge along the flow when liquidity is lower than the amount. If the edges happen to carry higher liquidity, the payment is delivered with an optimal fee for the sender.

If the payment, optimized for lowest fee, fails, then the next try is started with an onion along the path with the second lowest fee. Again, this might fail at the edge with liquidity lower than the amount.

In the Lightning Network protocol, there exists a feature called multi-part payments. This allows for a large payment to be split, so that each individual partial payment does not exceed the known capacity (or a fraction of it) of a channel, which might have been the case previously. A lower partial amount increases the set of possible edges that can be used for flow. But also, making use of different channels towards the receiver can avoid creating a cut as described above. However, following the strategy of the lowest fees, the chosen paths might always be the same for all participants and all partial payments will route along the same lowest fees edges, with their liquidity draining quickly.

When in contrast to the fee-optimizing strategy the payment is sent with **success probability of a payment** as the objective function for the payment delivery method, the multi-part payment flow with the highest success probability is looked for. This means that for a payment the expected value is calculated, which is a product of all probabilities of the channels in the flow.

This algorithm can be complemented by updating the probability distributions with the information gained from both successful and unsuccessful paths on prior payment tries. This leads to the payment with the most probable payment flow, based on the belief network of the node.

For a payment delivery strategy to send a payment as one amount, popular algorithms for finding a path in a graph are Dijkstra or A* (A star) algorithm, a variation of the Dijkstra algorithm in that it uses heuristics to decide upon the next best node.

For multi-part payments a minimum cost flow problem needs to be solved. This is a common problem for flow networks, and under certain circumstances solving the problem is not NP-hard and can thus be solved efficiently. This makes the use of minimum cost flow solutions in the context of the Lightning Network possible. Regarding the two preferences of minimum fee and maximum success probability, solving the minimum cost flow problem with the negative logarithm of the success probability as a cost function is more often applied. It is commonly referred to as pickhardt payments. The most basic version of pickhardt payments only considers uncertainty costs for reliability, but it is possible to also consider routing costs (fee rates) and optimize for both features to come up with reliable and cheap-ish multi-part payments [PR21].

2.4 Payment Delivery

Delivering a payment in the Lightning Network is a sequence of the following steps.

1. Analyze the network graph. If possible refer to belief network for additional information
2. Construct (multi-part payment) onion.
3. Send onions.
4. Successful payment or resending or payment partially stuck.
5. Update belief network.

The sending node looks up the information from the gossip protocol about the network graph, which includes the nodes in the network, the channel capacities, fees, but also the details and requirements of HTLCs for each channel. With this information, the node uses the flow finding algorithm to receive a sequence of channels. Using the sequence of channels that the flow finding algorithm returns and the information about the fees on each channel, the sending node constructs an onion along the nodes. When constructing the onion, the node takes into account the amount and the different fees along these hops and the payment hash, a special 256-bit secret value, allowing the receiver to recognize the incoming payment [AOP21].

The sending node then sends the payment onion package to the first node, which passes it on to the following nodes, until it arrives at the receiving node. Each node unpacks the dedicated onion part and processes its layer of the onion before forwarding. In more detail, the payment forwarding algorithm has the following steps:

1. Decrypting the outer layer of the onion and perform an integrity check.
2. Confirm wether the node can fulfill the routing/forwarding, depending on the fees and capacity it has with the outgoing channel.
3. Update channel state with the node from which it received the onion. This means to confirm the HTLC for the inbound channel.
4. Because the node changed the length of the onion while deconstructing its layer, the node adds padding to the end before passing it on to the next node.

5. Forward the onion package to the outgoing payment channel and send an `update_add_htlc` message with the payment hash and the onion.
6. After the onion arrived at the next node, it works with the next node to update the channel state on its outgoing channel.

Along the way an error could return on one node, due to the unavailability of the next hop or because liquidity in the outgoing channel is less than the amount in the onion. The payment forwarding is then interrupted and aborted. The node where the error is registered sends an error message wrapped in an onion back to the node from which it received its `update_add_htlc` message. The error onion is passed on to the sending node and along the path the nodes remove the pending HTLCs. When the error onion successfully arrives at the sending node, the sender has confidence that the payment failed and can then initiate another try.

Looking at the payment forwarding algorithm it is immediately clear that the worst case is a failing onion not sending the error message. Especially when sending a multi-part payment this means that the sending node has all partial payments outstanding, even those that successfully arrived at the receiver, but cannot finish the payment successfully, because the information about the state of one multi-part payment is missing. This is a very expensive situation for the sender and for the network, because liquidity is stuck, payments are locked and cannot be brought to use in channels.

3 Price of Anarchy of Selfish Payment Delivery

Participants in the Lightning Network use the network to send an amount of money from a sender to a receiver. The Lightning Network is a graph and money is sent through channels. When the money is passed on from one channel to another we are facing a transport problem, in which participants compete for liquidity in the channels. Each participant chooses its own payment delivery strategy considered to be the optimal strategy. A lack of coordination leads to a loss of benefit to the system. The question we want to ask is, how far away from the maximum achievable welfare of the network is the result of participants with selfish strategies?

3.1 Theory and Background

The payment is a flow. For a successful delivery, the payment amount needs to travel from edge to edge. The sender has to find a set of nodes that receive the payment amount via their inbound channels and they need to be able to move it on to the next node through one of their outbound channels. The nodes need to have a high enough amount of liquidity in a suitable outbound channel. The payment amount cannot be simply pushed along like a cart on a road. This metaphorical cart is effectively brought to a halt when reaching a node, and then a different cart, "different" satoshis, are sent on their way to the next node. So luckily satoshis are fungible, but still the satoshis sent onwards are not the same ones as the ones received. The main goal for successful payment delivery is to find a flow where nodes can receive in one hand and give in the other. This inability to pass on the identical satoshis makes relaying nodes a possibly scarce resource.

3.1.1 Competition and Cooperation

Finding the optimal flow in the network depends on a participant's preferences. A payment flow can be split up in several paths, a multi-part payment. The participant then tries to find its one optimal multi-part payment flow.

Participants have a self interest in having payments routed in a way that they maximize their individual welfare. However, this could result in a conflict with other participants, because they drain resources that others might wish to use as well. Liquidity is subject to consumption rivalry, and one participant optimizing for its own preference might reduce the welfare of one or more other participants. Participants compete for the scarce resource.

This immediately defines a game-theoretic framework, in which each participant has as many strategies as there are paths from sink to destination, and the cost to a participant needs to take into account the strategies of other agents, which lead to the current liquidity distribution in the network [KP99]. It is not uncommon that the pursuit of individually optimal strategies does not lead to an overall optimal outcome. One such prominent case is the prisoner's dilemma, a famous example in game theory. In this example one person behaves selfishly and tries to optimize its own outcome while reducing general welfare that could be achieved through cooperation.

3.1.2 Prisoner's Dilemma

The prisoner's dilemma is an example of a game analyzed in game theory. Two completely rational agents are forced into a dilemma:

> Tanya and Cinque have been arrested for robbing the Hibernia Savings Bank and placed in separate isolation cells. Both care much more about their personal freedom than about the welfare of their accomplice. A clever prosecutor makes the following offer to each: "You may choose to confess or remain silent. If you confess and your accomplice remains silent I will drop all charges against you and use your testimony to ensure that your accomplice does serious time. Likewise, if your accomplice confesses while you remain silent, they will go free while you do the time. If you both confess I get two convictions, but I'll see to it that you both get early parole. If you both remain silent, I'll have to settle for token sentences on firearms possession charges. If you wish to confess, you must leave a note with the jailer before my return tomorrow morning." [Kuh19]

This example shows, that an optimal outcome for both participants in the game is possible with coordination and trust. However, due to a lack of cooperation both participants most likely decide to confess and both are convicted, and consequently the equilibrium is not as good as an outcome for the participants compared to if they cooperated. It is a well-known fact that non-cooperative equilibria can be inefficient. [Dub86]

3.1.3 Braess's paradox

Another example for the inefficiency of non-cooperative equilibria and closer to our transport problem is Braess's paradox. Braess describes a network of roads with a known number of cars at each point. All participants will chose the cheapest possible path. The cost for each edge is determined by the time of travel.

From a mathematical perspective, this can be seen as a directed graph. The problem is different to a "classic" shortest path problem, in that not all travel times are independent of the traffic throughput on each edge. Some cost functions are constant, while others are determined by the number of cars choosing the respective edge. In this example Braess shows that there exists a flow that is optimal for all travelers, which maximizes the welfare of all participants. And he also shows that there exists a flow that will materialize, if each participant decides to optimize its own path, which deviates from this optimum.

In a simple four node graph he demonstrates that participants will believe to take the optimal path but in fact a better path could be established if there was a general flow control (see figure 3.1).

So let there be a directed graph $G = (A, U)$ with a set of nodes $A = \{a^i\}$ and a set of directed edges $U = \{u_\alpha\}$. φ_α is the flow on edge α, measured as cars per unit of time. As usual in traffic planning problems, flow is nonnegative.

$t_\alpha(\varphi)$ is the amount of time that is needed to pass edge u_α, if there is a flow of $\varphi = \varphi_\alpha$ on edge u_α. The total amount of time needed for a participant to get from a^i to a^k on path U_β is $T^{ik}_\beta(\Phi)$.

We are now looking for the highest cost on the graph. For this we look at all flows and find the flow with the highest cost $|T(\Phi)|$

$$|T(\Phi)| = max\{T_\beta(\Phi); \Phi_\beta \neq 0\}$$

The total flow in the graph Φ is optimal, if $|T(\Phi)| \leq |T(\Psi)|$ for all Ψ with $|\Phi| = |\Psi)|$

Braess defines the graph as follows, numbers on the edges are for reference:

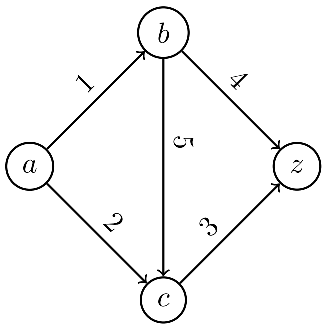

Figure 3.1: Braess's paradox, four node graph

In our example all cars want to drive from one source node a to one sink node z.

The cost functions for the five edges are:

$$\begin{aligned} t_1(\phi) = t_3(\phi) &= 10\phi \\ t_2(\phi) = t_4(\phi) &= 50 + \phi \\ t_5(\phi) &= 10 + \phi \end{aligned}$$

We find a unique solution for a flow of $|\Phi| = 2$ from a to z by sending all flow through $abcz$ with maximum cost of 52.

$$\Phi_{abcz} = 2, \quad \Phi_{abz} = \Phi_{acz} = 0, \quad |T(\Phi)| = 52 \tag{3.1}$$

We can also find unique solutions / equilibria for flows of $|\Phi| = 6$ and $|\Phi| = 20$ from a to z:

$$\Phi_{abcz} = 0, \quad \Phi_{abz} = \Phi_{acz} = 3, \quad |T(\Phi)| = 83 \tag{3.2}$$

$$\Phi_{abcz} = 0, \quad \Phi_{abz} = \Phi_{acz} = 10, \quad |T(\Phi)| = 160 \tag{3.3}$$

Every participant tries to find its optimal route by calculating how long it would take. In (3.1), following the route $abcz$ is individually optimal with a cost for the individual of 52, compared to for example abz, which would cost the participant 72. Total cost cannot be minimized, thus the flow of $\Phi_{abcz} = 2$ is optimal with $|T(\Phi)| = 52$.

In (3.3), half the participants following the route abz and the other half acz is optimal with a cost for the individual of 160. Choosing the route $abcz$ would cost the participant 480. Total cost cannot be minimized, thus the flow of $\Phi_{abz} = \Phi_{acz} = 10$ is optimal with $|T(\Phi)| = 160$.

However, in (3.2) we find another situation. A flow of 3 on each of the routes abz and acz is optimal, with cost for the individual being 83. But as participants gather experience, sooner or later one of the participants starting on *abz* will find out taking the faster road *abcz* might be more beneficial. While indeed their cost might decrease to $T_{abcz}(\Phi) = 81$, the other flows get more costly, with $T_{abz}(\Phi) = 82$ and $T_{acz}(\Phi) = 91$ resulting in

$$|T(\Phi)| = max\{T_{abcz}(\Phi), T_{abz}(\Phi), T_{acz}(\Phi)\} = 91.$$

With the next participant choosing the cheaper flow of *abcz*, the overall benefit decreases further, even on the previous shortcut: $T_{abcz}(\Phi) = 92$, $T_{acz}(\Phi) = 103$ and $T_{abz}(\Phi) = 81$, Leading to

$$|T(\Phi)| = max\{T_{abcz}(\Phi), T_{abz}(\Phi), T_{acz}(\Phi)\} = 103.$$

These examples are taken from [Bra68] and show that non-cooperation can lead to decreasing welfare in a network.

Choosing an individually optimal strategy often prevents finding an optimal outcome for all participants. Given that participants take rational decisions to achieve their individually best result, this decrease in welfare could only be avoided by general coordination or by changing circumstances such as their cost functions. This phenomenon can be observed in a lot of other domains as well. It does for example translate to network theory and routing. The term *selfish routing* was coined from analyzing traffic in congested networks.

3.1.4 Equilibria

In the situation with n participants wanting to send money through the Lightning Network, each of the n participants, or players or agents, as is the term in game theory, can choose among a set of strategies $S_i = 1, \dots, n$ that leads to an intended outcome. The intended outcome is the flow for the payment that maximizes the player's welfare or utility. To determine the players' utility, there are functions $u_i, i = 1, ..., n : S_1 \times \cdots \times S_n \mapsto \mathfrak{R}$ which assign to each such combined choice a utility for each player. The utility of a strategy of a player is mainly influenced by the cost for each player. $c_i : S \mapsto \mathbb{R}$ is the cost function of player $i \in N$. In state $s \in S$, player i has a cost of $c_i(s)$. u_i and c_i are generally inversely proportional, because costs reduce welfare. Game theory then analyzes what choice of strategies would be rational. One example of such concept of rationality is the *Nash equilibrium.*

Nash Equilibria

We denote by $s_{-i} = (s_1, ..., s_{i-1}, s_{i+1}, ..., s_n)$ a state s without the strategy s_i. This notation allows us to define a unilateral deviation of a player. For $i \in N$, let $s \in S$

and $s'_i \in S_i$, then $(s'_{i,s-i}) = (s_1, ..., s_{i-1}, s'_i, s_{i+1}, ..., s_n)$. A strategy s_i is called a best response for player $i \in N$ against a collection of strategies s_{-i} if $c_i(s_i, s_{-i}) \leq c_i(s'_i, s_{-i}) \forall s'_i \in S_i$.

Given what all other players are doing, a strategy is the best response if and only if a player cannot gain more utility from switching to a different strategy. And a state $s \in S$ is called a *pure Nash equilibrium* if s_i is a best response against the other strategies s_{-i} for every player $i \in N$ and thus $u_i(x_1, ..., x_i, ..., x_n) \geq u_i(x_1, ..., x'_i, ..., x_n)$ for all i and $x'_i \in S_i$. This describes a situation from which no player has an incentive to deviate. A game is in a Nash equilibrium if and only if all players are playing the best response to what the other players are doing.

Incomplete Information and Types of Players

In a Nash equilibrium, each player is assumed to know the equilibrium strategies of the other players. This is not the case in the Lightning Network, where routing nodes don't know which payment delivery strategies other nodes or participants follow and thus their preferred channels for payment delivery.

Harsanyi developed a theory for analysis of games where "some or all of the players lack full information about the "rules" of the game, or equivalently about its normal form (or about its extensive form). For example, they may lack full information about other players' or even their own payoff functions, about the physical facilities and strategies available to other players or even to themselves, about the amount of information the other players have about various aspects of the game situation etc.". [Har67]. Harsanyi described a game as having incomplete information when the players are uncertain about each other's types. A "type" is the belief not only over other players' actions, but also over these players' other characteristics. He developed a bayesian approach, where players will assign a subjective joint probability distribution to the player's type and then maximize the mathematical expectation of the player's own payoff over a game with these types of players.

Similar to the game with complete information we assume common knowledge about the set of players and the possible actions that they can take. In addition to this, we then have to refer to a probability distribution about the players' types, that determine their preferences/utility functions. While each player knows its own type, the other player's type is unknown. The probability distribution over the types, however, is common knowledge.

There exists a pure strategy Bayesian Nash equilibrium for each player and each realization of type $t_i \in T_i$ of player i's type, when there is an action that is a best response. The best response is the action that leads to an optimum expected probability weighted utility.

To translate this to the setting in the Lightning Network we can assume that the player currently deciding about the payment delivery strategy for its payment has another player doing the first move. This player can have two different types, type 1 having drained the channels that would be the preferred path for the player and type 2 having loaded the respective channels with liquidity. The best action for the sender would then be choosing the flow that maximizes the sum of the probability of meeting the type 1 player times the utility of failing on the payment and the probability of meeting type 2 player times the utility of a reliable payment delivery. If this action exists, then there is a Bayesian Nash equilibrium.

Berk-Nash Equilibria

In our game-theoretical setting of the Lightning Network we have a finite number of players, but players don't really know the strategies that the other agents are following. The requirement for a Bayesian Nash equilibrium, that the type of another player is unknown but the distribution of the types is known, is too strict. The participants in the Lightning Network don't know the distribution of payment delivery strategies and the (liquidity) state of the network before they decide on their own payment delivery. The standard assumption that people have a correctly-specified view of their environment does not hold. Consequently, participants in the network might build their subjective model, representing the own view of the environment they're acting in, before sending a payment, but their model might be misspecified. They can however learn and improve their view over time.

This leads to each player following a strategy that is optimal given their beliefs. Their beliefs are restricted to what they regard as the best fit among the set of beliefs they consider possible. The equilibrium that is based on a strategy profile such that, for each player, there exists a *belief* with support in the subjective model is called a Berk-Nash equilibrium [EP19].

The difference between the Berk-Nash equilibrium and the Nash equilibrium is that the players needn't have correct beliefs about the strategies of the other players but a belief that is sufficiently close to what they think the others would do. Berk-Nash equilibria refer to games where participants make simultaneous moves.

While Esponda and Pouzo are hesitant to translate their findings to sequential games [EP19], we find that Berk-Nash equilibria, such as simultaneous game considerations, still relate to the situation in the Lightning Network. This is because the outcome of previous players' moves are hidden to the player when making a decision. A player only learns about other players moves, which might have drained the liquidity in a channel that the player chose for the payment delivery, when actually sending the payment.

Finite Games and Infinite Games

The considerations regarding sequential games also lead to the question if the Lightning Network is to be treated like an infinite game. In fact one could see sending payments in the Lightning Network as an infinite game, because an unknown number of players sequentially sends unknown payments and change the state of the graph. However, in the context of our empirical analysis we simplify the Lightning Network to a finite game and consider this to be a sufficient approximation, a representative of all subgames in the original game. This is useful because a finite non-cooperative game always has at least one equilibrium point [NJ96] if the number of players and the number of strategies is finite. (They might not be pure Nash equilibria but could be mixed (strategy) Nash equilibria.)

Nonatomic Games, Anonymous Games

One could say that a player in the Lightning Network might only have an incremental effect on the state of the graph and as such cannot really influence the outcome. Consequently when other players choose their strategy, they would disregard the other player's behavior. This puts the focus on atomic and nonatomic games. In an atomic game the strategy choice of a player has an immediate and measurable effect on the outcome of the game for another player. In a nonatomic game "the single player has no influence on the situation, but the agregative behavior of "large" sets of players can change the payoffs." [Sch73].

This often goes hand in hand with anonymous games. In anonymous games it is not important for a player deciding on a strategy, who exactly the other player is (or players are), but how often a certain strategy is chosen. An example would be the question about congestion in traffic – how many players choose one road over the other – versus deciding about who to go to the movies with.

Cerreia-Vioglio et al. introduce a concept of approximate equilibrium for nonatomic anonymous games, which they call ε-estimated equilibrium. Berk-Nash equilibrium was developed for a finite game, finite-players framework and they extend this concept to nonatomic games. They demonstrate that a Berk-Nash ε-equilibrium exists for pure strategies for players when almost all players best-respond to their beliefs (optimality) and the beliefs are ε-close to the set of probabilistic models which are the best fit in the primitive set of the realized distribution (ε-fit) [CVMS20]. They make the important remark that each player's set of actions' distributions does not depend on the action played (but on the belief).

3.1.5 Price of Anarchy

"The question thus is, what happens if there is no central authority that designs, engineers and runs the Internet, but a master puppeteer, a benevolent dictator who, for example, micromanaged its operation, allocating bandwidth to flows so as to maximize total satisfaction? How much better would the Internet run? What is the price of anarchy?". This question was asked by Papadimitriou in relation to the internet [Pap01], but the same applies to considerations regarding networks in general and the Lightning Network in particular.

The extent to which the outcome of a competitive game with uncoordinated individual utility-maximizing decisions approximates the outcome of the same game with cooperation or coordination can be measured. The relationship is referred to as the *Price of Anarchy* and was analyzed in detail by Tim Roughgarden. He modeled classical multicommodity flow networks, described by the graph $G = (V, E)$, with node set V, edge set E and a set $(s_1, t_1), \ldots, (s_k, t_k)$ of source-sink node pairs [Rou05].

Roughgarden defines the Price of Anarchy of selfish routing in a network as the ratio between the cost of an equilibrium flow and the cost of an optimal (minimum-cost) flow, which is equivalent to putting the equilibrium with minimum cost or maximum utility in the non-cooperative game in relation to the cost of the outcome in a coordinated game. Roughgarden chooses the cost of an equilibrium flow in the Wardrop equilibrium, because such equilibrium has minimum cost, even though more than one equilibrium could exist. While this is appropriate for traffic and internet routing in congested networks and cost as travel time, we are reluctant to follow the definition for the Lightning Network. As we show above, there is a high degree of complexity in the Lightning Network, which makes it difficult to compare payment delivery to traffic routing, as we discuss in chapter 3.2. In our opinion an equilibrium is difficult to find. It still needs to be shown what equilibria exist for selfish routing in the Lightning Network.

We follow the definition of the Price of Anarchy in its original intention as the price of uncoordinated individual utility-maximizing decisions. The main prerequisite for the Lightning Network to be useful as a payment delivery network is its reliability. Consequently we can measure the overall utility of the network using the failure rate of payments, defined as the ratio of failing payments to the total amount of payments attempted. Selfish behaviour that reduces the networks reliability and increases failure rate, creates a cost for the network. Cost of selfish routing exist, are observable and can be measured using the failure rate of payment delivery strategies. They can be compared to the reliability rate in a network state micromanaged by a "benevolent dictator (. . .) maximizing total satisfaction", as explained by Papadimitriou [Pap01].

3.2 Anarchy in the Lightning Network

The protocol of the Lightning Network implements source-based onion routing. This means that the sender of a payment decides on the flow of the payment. As a consequence, there is significant room for selfish behavior in payment delivery. This is amplified by the fact that the Lightning Network is built as a network where no trust is needed and no cooperation is necessary. Another aspect is that every participant can set up its own node and then has full discretion on establishing a channel with someone else, when the other node agrees. In consequence, there is no control over the network topology.

The Lightning Network can be described as a flow network, with Nodes V and edges E, node capacities B and fees C, see chapter 2.2. Roughgarden established the Price of Anarchy for flow networks, having car traffic and network traffic in mind. However, the Lightning Network cannot be compared to a traffic network, even though similarities exist. A payment in the network is a flow, with constraints in liquidity along the edges. But it is different from a traffic network, because of its cost functions. In a traffic network, the main cost is travel time, which can for example be a fixed time or relative to the amount of traffic. In the Lightning Network, the utility is its reliability, the capacity to successfully send a payment. Consequently the dominant cost is unreliability, the failure rate when sending payments.

In the Lightning Network, nodes can charge a fee for using an edge, but if an edge does not carry liquidity, then no flow exists. Effectively the edge needs to be eliminated from the graph when solving for the optimal flow, but knowledge about effective liquidity does not exist for most edges. Thus, the lack of liquidity cannot explicitly be taken into account when constructing the intended flow, only assumptions about liquidity and thus the outcome of a payment delivery can be made. One such assumption, or belief, could be derived from the fees charged for a channel. Fees could be a signal for the liquidity and ability to relay a payment. With this signal the probability distribution that the sender builds over the available liquidity balances can be adjusted. Another cost function could be based on the probability distribution of liquidity balances gained from the channel capacity and the amount being sent. Both approaches require a different cost function for the Lightning Network when analyzing the Price of Anarchy than the cost functions for travel time in network problems as analyzed by Roughgarden [Rou05].

Similar to traffic analysis, in the Lightning Network there are instructive examples demonstrating that competitive behavior leads to a less optimal outcome compared to cooperative or coordinated behavior. The analysis to find the optimal strategy is more difficult compared to the traffic network case, because the participants decide on a strategy while the underlying network graph, given by nodes, edges and liquidity balances, is unknown. Thus, they form their own expectations and beliefs of the network graph, which is incorporated in the decision function on their optimal payment flow.

As pointed out above, we follow the definition of the Price of Anarchy ρ as the ratio between the cost of uncoordinated individual utility-maximizing behavior measured by its failure rate and reliability in a network managed by an omniscient router maximizing total satisfaction. This can also be seen as the ratio of the cost for the network of competitive strategies in relation to cooperative strategies.

$$\rho = \frac{\text{failure rate of uncoordinated individual utility-maximizing behavior}}{\text{failure rate of a network managed by an omniscient router}}$$

3.3 Optimal Decisions and Strategies

Because the Price of Anarchy is defined as the ratio of the cost of uncoordinated individual utility-maximizing behavior and the cost in the cooperative utility-maximizing strategy, it heavily depends on the definition of these optimal strategies. We describe what makes an optimal strategy for the participants in the Lightning Network.

3.3.1 Available Information for Finding an Optimal Solution

The gossip protocol provides information about the nodes in the network, the currently active channels, their capacity, the fees and further information like the maximum amount for HTLCs, that can be set in the channel. The sender of a payment knows the amount of the flow to be sent, the node address of the receiver of the payment and it also knows the liquidity that is in its own outbound channels.

The sender can also decide to not build one single payment onion but to split it up in several partial payments, called a multi-part payment.

The sender does not have any information on other payments that have been executed or that are currently being executed in the Lightning Network, which means that there are possibly other nodes updating HTLCs in the network. And because it is possible that flows of two or more payments are not disjoint, conflicts might arise. Consequently the sequence of payments plays a significant role when it comes to available liquidity in the selected flow, but this is completely hidden from the sender. The sender only knows its own flows.

Besides the information from the gossip graph, the node can manage its own belief about the network, for example trying to retain information about liquidity on the edges learnt from successful or failing payments. Another possible information is a score for the nodes to reflect experience of unavailability of nodes. There is no constraint to what a node decides to learn and include in its optimization strategy.

Apart from these constraints in information, the nodes are subject to further technical constraints with regards to bandwidth and processing power of a node.

With this information it can derive a flow that best matches the optimum of the sender's cost function. Based on this flow it can construct an onion and initiate the payment delivery.

3.3.2 What Makes an Optimal Solution

When the sender tries to find the optimal payment delivery, it first has to decide on its preference function. What is considered to be suitable and then optimal is different from participant to participant. After deciding about what is most important, a cost function needs to be found, that reflects this preference and the trade-off amongst different goals. The choice of cost function is at the discretion of the person running the sending node. It can range from using the default cost function of the chosen software implementation of the lightning node distribution up to some complex custom made variant.

Some senders might aim for a fast delivery, others might prefer to have as little hops on the network as possible, and then again others might have a strong preference for reliability of the network, just to mention a few optimization goals. Regarding possible costs for the sender, such goals can for example be

- **Low fees:** The sender has to pay a fee to the nodes along the way, which can either be a fixed fee, or a fee dependent on the amount on the channel, or a combination of both.
- **Speed:** Every lightning node in the internet has some latency when relaying a payment. This is due to the time to be reached the internet transport protocol and also from unpacking and packing the payment onion. Thus, speedy delivery can be aimed to achieve by choosing a delivery with as little hops as possible.
- **Reliability:** Because failing payments, especially in a multi-part payment, cause delays of the confirmation of the payment, choosing reliability might lead to less tries to settle a payment and thus have quicker confirmation about successful delivery of the payment.
- **Liquidity management:** Filling up channels where liquidity has been drained can be another goal. As a consequence there are particular requirements regarding the channels along the flow and the amount of liquidity, rather than fees or speed.

The cost function can reference one or more of these payment delivery goals. For example they can be combined linearly or exponentially. The choice of the cost function represents the preference of the sender regarding the payment delivery and has an impact on the possibility to solve the optimization problem on the graph in a limited and feasible amount of time.

Sending a payment means to send an amount from source s to sink t, which is a flow f with function $f : E \longrightarrow \mathbb{N}_0$ and the flow f_e being the flow f on edge $e \in E$.

The flow needs to be a *feasible flow*, which means that it needs to fulfill the requirements of capacity constraint and conservation of flows as described in chapter 2.3.1.

$\mathcal{C}$ is a set of diverse cost functions. The cost of sending this flow along the edge e_{uv} is $c_{uv}(f_{e_{uv}})$ with $c \in C$ and $f_{e_{uv}}$ the amount of flow on the edge.

The problem for the sender then is to minimize the total cost of the flow over all edges

$$C(f) = \sum_{e_{uv} \in E} c_{uv}(f_{e_{uv}}).$$

subject to

$$0 \leq f_e \leq cap_e \qquad \text{(capacity constraint)}$$
$$\sum_{(u,v) \in V} f_{e_{uv}} - \sum_{(u,v) \in V} f_{e_{vu}} = b_v \qquad \text{(conservation of flows)}$$

for $G = (V, E)$ being a directed graph with nodes $v \in V$ and edges $e \in E$, f being a feasible flow and $\sum_{v \in V} b_v = 0 \ \forall \ v \notin \{s, d\}$.

3.3.3 Algorithms to Find the Optimal Solution

To map the problem of finding the optimal payment flow to the preference function of the sender, a suitable optimization function needs to be found. The most common algorithms are the algorithms to solve the minimum-cost flow problem and Dijkstra.

MinCostFlow

The minimum-cost flow (MinCostFlow) problem describes the problem to find a feasible flow for a given value m in a flow network with minimum cost. In a directed graph with nodes V and edges E all edges have a cost and a capacity while the nodes have a balance representing supply and demand. The minimum-cost flow problem can be seen as a generalization of the shortest path and maximum flow problems.

There are several algorithms trying to solve the minimum-cost flow problem, for example the successive shortest path algorithm, the cycle canceling algorithm, and the network simplex algorihtm. To illustrate one possible process, we present the successive shortest path algorithm.

In the setup phase, the algorithm adds for all edges in the directed graph a reverse edge with liquidity 0, and the cost of the reverse edge is the negative of the cost of the original edge. A residual network is defined for a fixed flow F as the network containing only unsaturated edges and the residual liquidity R of each such edge is the previous liquidity minus the flow on this edge in the graph.

At each iteration of the algorithm we find the shortest path in the residual graph from s to t with the shortest path defined as the lowest cost of the path. If no path exists anymore, then the algorithm terminates, and the stream F is the resulting flow.

If a path is found, the flow along this path is increased as much as possible, i.e. the minimal residual liquidity R of the path is found and flow on this edge is increased accordingly. The reverse edges are reduced by the same amount. If at some point the flow reaches the value m, then the algorithm stops. Of course, in the last iteration of the algorithm the flow is only increased by an amount such that the total flow is m.

Dijkstra

The Dijkstra algorithm is an algorithm that can find the shortest path between the sending node and the receiving node in the lightning graph. Its name stems from computer scientist Edgar Dijkstra who published the algorithm. The Dijkstra algorithm belongs to the class of greedy algorithms, which stepwise choose the next solution promising the highest gain.

Dijkstra calculates the shortest path on the cost graph. For this, the edges get assigned weights which reflect the costs. Because the amount of the flow on the edge is known when running the algorithm, a defined fixed cost can be assigned to the edges. Costs on the edges cannot be negative in a Dijkstra algorithm, otherwise a switch to Bellman-Ford-algorithm is advisable. For example when running the Dijkstra algorithm on the cost graph with the fees as cost, the algorithm will find the path with the lowest fees.

Starting at the sending node, the algorithm compares the cost of the neighboring nodes. It then chooses as the next node for its comparison the node with the cheapest cost along the edges and compares the costs of the neighboring nodes of the current node. When progressing through the network and the first path arrives at the receiving node of the payment, all more expensive paths are dropped. For all paths where so far the costs are lower than the path that arrived at the receiving node, their neighboring nodes are explored as described above, and costs along the path are compared to the solution already found. Whenever a cheaper solution is found, the most recent one is dropped and the new solution treated as the optimal solution.

The Dijkstra algorithm is a speedy algorithm when applied to a graph with constant weights.

3.4 Example of Detrimental Selfish Routing

It is safe to assume that the most important criterion for the Lightning Network is its reliability, because with a high proportion of payments failing, its overall use as a payment network is sharply reduced. The following instructive example[1] demonstrates that a Price of Anarchy exists in the Lightning Network. In order to simplify the example, we ignore fees and channel reserves and optimize for success probability. We assume a uniform probability distribution for liquidity in the channel.

A graph with three nodes A, B, C represents a subgraph of the Lightning Network. The edges e_{AB}, e_{AC} and e_{CB} carry the capacities of 2, 3 and 3 respectively.

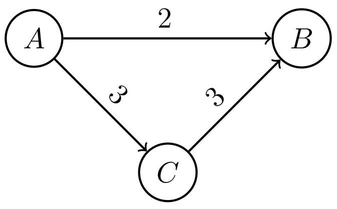

Figure 3.2: Example subgraph of Lightning Network

3.4.1 Selfish Routing

Let there be two payments of 1 each from s to t, passing the above subgraph.

The selfish senders would follow this decision process:

As shown in chapter 2.3.4, the probability that there is a successful payment from A to B along the edge e_{AB} with flow $f_{e_{AB}} = 1$ is

$$P_{e_{AB}}(X_{e_{AB}} \geq f_{e_{AB}}) = \frac{cap_{e_{AB}} + 1 - f_{e_{AB}}}{cap_{e_{AB}} + 1} = \frac{2+1-1}{2+1} = \frac{2}{3}. \tag{3.4}$$

[1]This example is taken from a discussion in fall 2022 with my mentor René Pickhardt on flow and game theory in the Lightning Network.

For sending the payment of 1 from A to B along the edges e_{AC} and e_{CB} the sender calculates:

$$P_{e_{AC}}(X_{e_{AC}} \geq f_{e_{AC}}) = \frac{cap_{e_{AC}} + 1 - f_{e_{AC}}}{cap_{e_{AC}} + 1} = \frac{3}{4} \tag{3.5}$$

$$P_{e_{CB}}(X_{e_{CB}} \geq f_{e_{CB}}) = \frac{cap_{e_{CB}} + 1 - f_{e_{CB}}}{cap_{e_{CB}} + 1} = \frac{3}{4} \tag{3.6}$$

The success probability of the flow from A to B along the edges e_{AC} and e_{CB} follows as:

$$P_f = \prod_{e \in E} P(X_e \geq f_e) = \frac{3}{4} * \frac{3}{4} = \frac{9}{16}. \tag{3.7}$$

Using a probabilistic payment delivery the sending node will try to route the payment along the path AB, because $\frac{2}{3} > \frac{9}{16}$.

The same will be the case for the other participant, who would take the same decision. Consequently there will be a flow of $f_{e_{AB}} = 2$. The probability for *both* payments to be successfully delivered via edge e_{AB} now decreases.

For a flow of 2 from A to B along the edge e_{AB}, the success probability is

$$\begin{aligned} P_{e_{AB}}(X_{e_{AB}} \geq f_{e_{AB}}) &= \frac{cap_{e_{AB}} + 1 - f_{e_{AB}}}{cap_{e_{AB}} + 1} = \frac{2+1-2}{2+1} \\ &= \frac{1}{3}. \end{aligned}$$

3.4.2 Global Coordination

How would a global coordinator solve the flow of the two payments of 1 each in the subgraph? A global coordinator could achieve a better solution by sending the two payments of 1 each along different paths, $f_{AB} = 1$ and $f_{ABC} = 1$. The probabilities are:

$$\begin{aligned} P_{AB}(X_{AB} \geq f_{AB}) &= \frac{cap_{e_{AB}} + 1 - f_{AB}}{cap_{e_{AB}} + 1} = \frac{2+1-1}{2+1} \\ &= \frac{2}{3}. \end{aligned}$$

$$\begin{aligned} P_{ACB}(X_{ACB} \geq f_{ACB}) &= P_{e_{AC}}(X_{e_{AC}} \geq f_{AC}) * P_{e_{AC}}(X_{e_{AC}} \geq f_{AC}) \\ &= \frac{3+1-1}{3+1} * \frac{3+1-1}{3+1} = \frac{3}{4} * \frac{3}{4} \\ &= \frac{9}{16} \end{aligned}$$

The success probability of the split payment is the product of the success probabilities of both partial payments:

$$P_f = \prod_{e \in E} P(X_e \geq f_e) = \frac{2}{3} * \frac{9}{16} = \frac{3}{8},$$

which is better than the success probability with selfish routing of $\frac{1}{3}$.

3.4.3 Price of Anarchy in the Example

Regarding the failure probabilities of the payments as the cost of the payments, we see that the selfish payment flow will incur a cost of $1 - \frac{1}{3} = \frac{2}{3}$ while the cost of the coordinated payment flow is $\frac{5}{8}$. The loss of welfare to the network from selfish routing on this subgraph is thus $\frac{1}{24}$.

Calculating the Price of Anarchy as the ratio between the failure probability for selfish routing and the optimal, coordinated flow, the Price of Anarchy ρ can be calculated as:

$$\rho = \frac{\frac{2}{3}}{\frac{5}{8}} = \frac{16}{15}$$

This demonstrates that the choice of the singular payment, selfishly routed, incurs a higher cost, than the coordinated routing.

It is a point of discussion, if improving the payment delivery mechanism for the selfish router's single payment could lead to a payment with less cost. Because any selfish participant would constantly try to reduce cost. One such possibility would be to try to replicate a similar approach by splitting payments. And indeed, this describes what is being done in the pickhardt payment approach. However, this only works, if all (partial) payments are known.

It is useful to consider the practical impact of the selfish payments. While we assume a uniform distribution of liquidity regarding the probability of having enough liquidity in the channels, one can see that routing along AB seems a dominant strategy for selfish routers. While when constructing the onion, the sending node can only guess the success probability a priori, it is highly likely that the actual liquidity has already been drained. This can be assumed, because having two paths from A to B, the cheaper one is likely to be chosen more often and thus the channel is more likely to be saturated.

4 Implementation of a Simulator of the Lightning Network

To analyze if a Price of Anarchy exists in the Lightning Network, we simulate the payment delivery strategies, non-cooperative as well as cooperative, in a model of the Lightning Network. The model is a directed multigraph with the vertices representing lightning nodes and the edges representing Poon-Dryja channels. The code used to implement the model can be found in the code repository [Als23].

When initializing our model of the Lightning Network we eliminate channels with no return channels as described below, and randomly assign liquidity balances to the channels. To allow for replication, we define a seed value, so that all experiments are based on the same randomly assigned liquidity balances.

For our analyzes we generate a set of 10,000 payments, triples consisting of a sending node, a receiving node, and a given amount. We decide to randomly assign payment amounts between 10,000 and 1,000,000 satoshis.

The topology of the Lightning Network is given by the gossip graph as of January 14th, 2023. The graph consists of 133,626 edges and 14,839 nodes. We remove 57,212 edges and 1,688 nodes because we ignore channels with a base fee to make solving for optimal multi-part payments a linear min-cost flow problem and not an NP-hard problem, as has been discussed in [PR21]. We then eliminate edges that have no return channel. This leads to a network of 37,074 edges or channels and 3,299 nodes.

4.1 Description of the Model

To imitate the Lightning Network in our model so that we can run our simulations, we model participants, functions and behavior of nodes in the network. We simulate a node that is capable of generating a path along which it wants to send its payment to the receiver. As described above, path generation follows different strategies. We execute a payment, which means that channels with flows deplete and the balances of the return channels are adjusted appropriately. We maintain the state of the Lightning Network, so that we can decide if payments are successful or not.

Additionally we create a framework that allows us to initialize the model with appropriate data, so it matches the real world Lightning Network as much as possible and we use methods that allow us to generate and execute the payments in the context of the simulation. From previous research about pickhardt payments, or probabilistic multi-part payments, a python library exists that models the Lightning Network and parts of the necessary functions [Pic23]. We use this library as a base and extend it appropriately.

4.1.1 Channel Graph

The Lightning Network is a set of nodes and channels with properties such as liquidity, capacity, fees and more. Information about the channel states is shared through the gossip protocol. A node implementation that follows the lightning protocol listens to the gossip in the network and establishes a list of peers, i.e. other nodes, and channels as well as information about other nodes. From any standard node implementation the knowledge about the channels in the network can be exported, for example with the `listchannels` call in `core lightning`.

A drawback from a simulation perspective is the missing information about the liquidity in the channels, which is in fact a privacy feature of the Lightning Network.

Calling `listchannels` on a node implementing core lightning, we get a json file that holds all publicly available information about the channels of the lightning network:

```
    {
       "source": "0325bb9b . . . 0bb12ffb",
       "destination": "032e6032 . . . 28ccb483",
       "short_channel_id": "759263x1025x0",
       "public": true,
       "satoshis": 11590756,
       "amount_msat": "11590756000msat",
       "message_flags": 1,
       "channel_flags": 0,
       "active": true,
       "last_update": 1673565300,
       "base_fee_millisatoshi": 0,
       "fee_per_millionth": 500,
       "delay": 60,
       "htlc_minimum_msat": "1000msat",
       "htlc_maximum_msat": "11590756000msat",
       "features": ""
    },
    {
       "source": "032e6032 . . . 28ccb483",
       "destination": "0325bb9b . . . 0bb12ffb",
       "short_channel_id": "759263x1025x0",
       "public": true,
       "satoshis": 11590756,
       "amount_msat": "11590756000msat",
       "message_flags": 1,
        "channel_flags": 1,
        "active": true,
        "last_update": 1673066753,
        "base_fee_millisatoshi": 0,
        "fee_per_millionth": 0,
        "delay": 34,
        "htlc_minimum_msat": "1msat",
        "htlc_maximum_msat": "11474849000msat",
        "features": ""
    },
```

Listing 4.1: Excerpt of the listchannels json file with the announced channels in the Lightning Network.

To implement a graph of the Lightning Network we refer to the implementation of the ChannelGraph Class in the pickhardt payment library.

The ChannelGraph is a directed graph with parallel edges. We generate the channel graph by creating a network graph form the networkx library [HSSC08] and adding the edges that are given by the `listchannel.json` file and their nodes. After initialization of the ChannelGraph, the ChannelGraph instance represents the known state of the Lightning Network as seen by a node.

We simulate effects on the graph after payments have been executed. To properly assign liquidity after the settlement of a payment, we allocate liquidity in the channel corresponding to the flow from sending to receiving node, and also in the return channel. For this to work, we eliminate all channels from the ChannelGraph that don't have a return channel announced, because otherwise the reallocation of liquidity will fail. A corresponding helper function is available in the simulation framework.

In our simulation we use the information from the gossip graph as of January 14th, 2023.

4.1.2 Oracle Lightning Network

While the ChannelGraph instance reflects all publicly available information about the Lightning Network channels, it does not provide information about the liquidity balances in the channels. To initialize a state of the graph that does provide liquidity balances and also carries the liquidity balances during our simulations, we use the OracleLightningNetwork Class.

The OracleLightningNetwork Class of the pickhardtpayment library inherits from ChannelGraph. The OracleLightningNetwork is a graph of OracleChannels.

At instantiation the OracleLightningNetwork instance adds the ChannelGraph's channels to the OracleLightingNetwork graph as OracleChannels. In this loop, the OracleChannels get a liquidity balance randomly assigned, with the values ranging from zero to the channel's announced capacity. The nodes don't have any knowledge about the balances in the oracle network other than about their own inbound and outbound channels. They can only learn about balances in other channels when they attempt to send a payment onion with payment information. For this a `sendonion` method exists on the OracleLightningNetwork instance. The `sendonion` call returns `true`, if the payment amount could be delivered along the given list of channels. If the onion fails, than the method returns `false` and the channel id of the channel that could not relay the payment is returned as well.

From a failing payment onion the node can learn that the liquidity in the respective channel is less then the amount sent, and the return channel carries at least an amount of liquidity equal to the capacity less the payment amount. If the payment onion succeeds, then it knows that originally the channel did carry at least this amount of liquidity. But when settled, this liquidity is available in the return channel. Additionally the node knows that the remaining liquidity is at most the capacity of the channel less the sent amount.

When originally assigning balances we choose a random number generator with uniformly distributed values between $x = 0$ and $x = 1$. The balance in the channel is then $x * \textit{channel capacity}$. If a liquidity balance has been assigned, the liquidity in the return channel is determined as well.

4.1.3 Uncertainty Network

In the Lighting Network the source decides on the channels used for payment delivery. This means that the nodes can utilize a strategy of their liking to generate the set of channels for the flow. To optimize the outcome, which will usually depend on the likelihood to settle successfully and/or with lowest possible fees, the nodes have to make assumptions on liquidity.

This is not a trivial task, however, because no operation exists to easily request this information. One can immediately see that liquidity is in a range between zero and the channel's capacity. When nodes send or relay payments, they receive a response from the method sending the onions, which they can compile. They then adjust their knowledge about the probed channels accordingly. Knowledge about the channels' minimum and maximum liquidity can be gained, retained and taken into consideration when generating the paths. The UncertaintyNetwork represents the belief network mentioned in chapter 2.3.3.

4.1.4 Payment

The Payment Class in the pickhardtpayment library contains all properties and necessary methods for a payment to be sent from sender to receiver. It provides methods to generate the flows for the given amount and registers them as one or, in case of a multi-part payment, several `Attempts`. An attempt is a collection of channels for a particular flow in the network. Each attempt can then be recorded as planned, failed, inflight or settled, depending on the result when calling `sendonion`. At first a flow is planned after being generated by an algorithm like Dijkstra or MinCostFlow. When the flow has been tested with `sendonion` and the result is that the amount can be forwarded, then the attempt is registered as inflight. If the amount for this attempt cannot be passed on, then it receives the failed flag. After all attempts have been tried with `sendonion` and the total flow of the payment can be delivered successfully, then settlement is initiated and the attempts that settled successfully receive the status of being settled.

The Payment Class given by the pickhardtpayment library contains a method to generate the flow for the payment with a MinCostFlow solver. This is used when generating paths for multi-part payments.

Because we also simulate non-multi-part payments, we add a method for finding the shortest path with the Dijkstra algorithm. In this method we pass a string to determine the weight for the Dijkstra search. This weight can be fees or probabilities and the corresponding cost will be used.

```python
def get_dijkstra_path(self, graph, weight: str):
    try:
        node_path = nx.dijkstra_path(graph, self.sender,
                                     self.receiver, weight=weight)
    except:
        logging.error("no path found")
        raise DijkstraSolverError("Cannot determine path with Dijkstra.")
    else:
        logging.debug(f"shortest path: {node_path}")
        path = self.convert_node_path_to_attempt_path(graph, node_path)
        self.attempts.append(Attempt(path, self._total_amount))
        return path
```

Listing 4.2: Method for generating the shortest path for a payment flow using Dijkstra in the Payment Class.

4.2 Simulation Framework

To conduct experiments in the simulation framework, we define helper methods. One helper method cleans the ChannelGraph as described above, so that only channel pairs exist. Another helper method generates payment triples by sampling two nodes from the graph and a random amount, so that all experiments run on the same set of payments.

A method for executing multi-part payments exists in the pickhardtpay library (`pickhardt_pay` method). The method can be called with a linear combination of success probability and fees as a cost function. This provides the functionality for testing payment delivery based on fees but also a probability based solution. This method also implements a payment loop so that several rounds of attempts can be tried. We restrict this loop however to only one payment try.

We complement the method for payment of multi-part payments with a method that replicates the payment delivery with the Dijkstra algorithm. We also implement a payment loop but restrict it to only one round. This is for consistency reasons when contributing the method to the pickhardtpay open source library.

```python
def dijkstra_pay(self, src, dest, amt=1, criteria="fee", base=DEFAULT_BASE_THRESHOLD,
     loglevel="debug") -> int:
    payment = Payment(self.uncertainty_network, self.oracle_network, src, dest, amt, 1, base)
    _round = 0
    success = False

    G = nx.MultiDiGraph()
    for edge in self.uncertainty_network.network.edges(data="channel", keys=True):
        if edge[3].capacity > amt:
            prob = -math.log(1 - amt / edge[3].capacity)
            G.add_edge(edge[0], edge[1], fee=edge[3].ppm, probability=prob, capacity=edge[3].
                 capacity,
                       short_id=edge[2])

    while payment.residual_amount > 0 and _round < 1 and not success:
        _round += 1
        logging.debug(f"round: {_round}")
        try:
            path = payment.get_dijkstra_path(G, criteria)
        except DijkstraSolverError as err:
            logging.warning(err)
            logging.warning("Payment failed. No path found.")
            return -1
        else:
            attempt = payment.attempts[len(payment.attempts) - 1]
            logging.debug("attempt: {}".format(attempt))
            success = True
            for channel in attempt.path:
                ch = self.oracle_network.get_channel(channel.src, channel.dest, channel.
                     short_channel_id)
                liqui = ch.actual_liquidity
                if attempt.amount >= liqui:
                    success = False
                    logging.debug(f"Failing channel: {ch}")
                    G.remove_edge(channel.src, channel.dest)

                logging.debug("- channel {}-{} with capacity {:,.0f}, liquidity {:,.0f} and
                     fees {:,.0f}"
                              .format(channel.src[0:4], channel.dest[0:4], channel.capacity,
                                   liqui, channel.ppm))
            payment.attempt_payments()

    if payment.residual_amount:
        return payment.residual_amount
    else:
        payment.execute()
    return 0
```

Listing 4.3: Method for execution of payments using Dijkstra for finding the payment flow.

The simulation framework provides methods for executing payments with a given payment delivery strategy. The simulations are similar in structure but depending on the payment delivery strategy differ by the payment method they call.

The function receives the set of payments as well as the ChannelGraph as parameters. Then the UncertaintyNetwork, the OracleLightningNetwork and a PaymentSession (an instance of the `SyncSimulatedPaymentSession Class` in the pickhardtpayment library) are instantiated, together with parameters for collecting the results of the simulation.

```
def dijkstra_probability(_payment_set, _graph):
    _uncertainty_network = UncertaintyNetwork(_graph)
    _oracle_lightning_network = OracleLightningNetwork(_graph)
    _sim_session = SyncSimulatedPaymentSession(_oracle_lightning_network, _uncertainty_network
        , prune_network=False)
    c = 0
    _all_payments = []
    sent_amount = 0
    failed_amount = 0
    total_amount = 0

    for payment in _payment_set:
        c += 1
        _sim_session.forget_information()
        ret = _sim_session.dijkstra_pay(payment["sender"], payment["receiver"],
                                        payment["amount"], "probability", loglevel=‘‘warning)
        total_amount += payment["amount"]
        payment["delivery_method"] = "dijkstra_probabilities"

        if ret == 0:
            payment['success'] = "success"
            payment["residual_amount"] = 0
            sent_amount += payment["amount"]
            logging.debug("Payment {} was successful.".format(c))
        elif ret == -1:
            payment['success'] = "no_path_found"
            payment["residual_amount"] = payment["amount"]
            failed_amount += payment["amount"]
            logging.debug("Payment {} failed.".format(c))
        elif ret > 0:
            payment['success'] = "delivery_failure"
            payment["residual_amount"] = ret
            failed_amount += payment["residual_amount"]
            logging.debug("Payment {} failed.".format(c))
        _all_payments.append(payment)

    # write all payments to file
    ndjson.dump(_all_payments, open("data/dijkstra_probability.ndjson", "w"))
```

Listing 4.4: Example for a method that executes the simulation for a payment delivery strategy, here Dijkstra, on the probability graph.

The method at first resets the state in the uncertainty network when the simulation does not model a general retention of information from successful or failing payments. Then it loops through all payments in the payment set and calls the corresponding pay method. If the method fails, because no path could be found for payment delivery, then it returns `-1`. In this case the result of `no_path_found` is registered with the payment list item and the residual amount outstanding is the total amount of the payment. Otherwise the payment method returns the residual amount of the payment that could not be delivered. If this value is 0, then the total amount could be delivered and the payment was executed successfully. If the return amount is positive, then the payment could not be delivered completely, which is the case if a single or partial payment failed, because of a lack of liquidity in the channels.

4.3 Payment Delivery Strategies

The sending node chooses a payment delivery method, for example to send a single payment or multi-part payment and to optimize for low fees or high payment delivery success probability. It then needs to solve the corresponding optimization problem to find the most suitable feasible flow for the payment.

4.3.1 Single Payment with Fees as Cost Function

The Lightning Network, similar to the bitcoin project, balances people's hedonistic and self-centered interests through mechanics of game theory. As such, participants tend to behave in a self-optimizing manner. The fees paid for delivering a payment is in this context an important criterion for participants, because it leads to low individual money cost, but more importantly fees can include a signal regarding the availability of liquidity, as has been discussed in chapter 1.2. In consequence the sender applies a cost function that is centered around fee expenses. The sender wants to send one single payment with as little fees paid as possible.

Because in our simulation we only perform one payment round, we emulate the fee based single payment cost function with a Dijkstra algorithm and look for the shortest path on the fee graph. We demonstrate the implementation in listing 4.3 in chapter 4.2.

4.3.2 Single Payment with Probability as Cost Function

While this view is still quite common, more and more participants and particularly node operators and Lightning Service Providers (LSPs) realize that fees paid (or fee income) cannot be the sole driver of payment delivery strategies, because the prediction quality is far from perfect. There is a rivalry in the Lightning Network, because participants compete for channel liquidity. And with more participants joining via LSPs, centrality in the Lightning Network increases, moving the graph further away from being a complete graph, leading to some channels being used more often, potentially creating bottlenecks. Thus, finding a flow that has a high probability of not running into liquidity problems gains in importance. This leads to considering the probability of a failed payment delivery as a cost in the optimization problem, as we discussed in chapter 2.3.4.

In our model we emulate the use of this cost function with a Dijkstra algorithm and look for the shortest path on the probability graph, with the probability being the negative logarithm of the probability for an amount to succeed on a particular channel, given the channel's capacity and the payment amount.

4.3.3 Multi-Part Payments

Because the Lightning Network protocol allows for multi-part payments, the cost functions are not restricted to considering one singular path of channels for delivery and for calculating costs on this path. Multi-part payments provide for the possibility to split payments and combine several payments of smaller amounts. This can be used to find a new optimum with regards to the same cost functions as previously used for a single payment.

For a cost function on fees, this allows for a combination of flows that make use of channels with low fees that have previously been unavailable when the amount was to be sent in one sum. This happens for example when the capacity of a channel is lower than the total amount to be paid. These channels have previously not been in the set of possible payment channels, but for a lower partial amount they could become part of a feasible flow.

The same is the case when using the failure rate as a cost function. Multi-part payments allow for a linear combination of more paths with a higher success probability each, as for example shown in chapter 3.4.2. Both functions are convex functions and thus can be used for finding an optimal flow in a MinCostFlow solver in an acceptable amount of time.

5 Experimental Setup

We conduct experiments to observe a Price of Anarchy for payment delivery in the Lightning Network. Network participants use different strategies. We observe the state of the network and the failure rates for payments when the respective strategies are applied. We assume that the individually optimal strategy, the selfish strategy, can be found within this set of possible payment delivery strategies. From this we can infer the reliability of the network in a competitive setting. The optimal strategy is defined as the available strategy with the highest benefit, the lowest failure rate for payments.

Following these experiments, we assume the existence of an omniscient participant who can decide on the flow of the payments with a preference to maximize the welfare in the network and minimize the failure rate of payments. These experiments represent the state of the network if participants were cooperating.

5.1 Methodology

For our analysis we generate a network graph that resembles the Lightning Network as observed on January 14th 2023, using the output of the `listchannels` method of the core lightning node implementation [Tea23a]. To achieve this, we use the pickhardt payments library [Pic23] (see chapter 4.1.1). Because no information is available about the liquidity balances in the payment channels between nodes, we create an initial Lightning Network model graph, our base graph, by assigning liquidity balances randomly (see chapter 4.1.2).

Based on this graph, we create a set of 10,000 payments, consisting of a sending node, a receiving node and a payment amount. The nodes are drawn randomly from the graph. We execute all 10,000 payments with each of the four selfish strategies implemented in our simulator as well as the cooperative strategy, multi-part payments based on success probabilities with learning, also known as pickhardt payments. For each executed payment we observe the result, one of three states: no path was found, payment failed, payment settled. From this we establish the failure rate for the given strategy on the graph. The failure rate is calculated as the number of failed payments divided by the number of payment attempts made with a given strategy. We run this experiment of the 10,000 payments on the five strategies 20 times, using different randomly assigned liquidity balances.

We monitor the payment attempts and what payment channels were used for the multi-part payment strategy with success probabilities as a cost function in the base graph. With this data we analyse the multiple use of payment channels and examine the possibilities for aggregation to reduce payment failure rates.

To avoid skewed results, we then follow up with further experiments to validate our findings regarding different payment sets and a different underlying graph topology. Using our base case Lightning Network model graph, we create five random sets of 10,000 payments and execute two payment delivery strategies to analyse for outliers in the failure rates of strategies because of an unfortunate choice of payment set. We manipulate centrality in the Lightning Network model graph by eliminating the 100, 200, and 300 most central nodes and execute on each of these graphs the 10,000 payments for each of the five strategies. We also construct five random graphs and execute the 10,000 payments for each of the five strategies to validate the results based on the Lightning Network model graph.

5.2 Creating Payment Triples

To analyze failure rates of payments in our Lightning Network model graph, besides a graph we need payments that we can execute. Because payments by network participants are hidden information in the Lightning Network, we need to generate payments ourself.

The Lightning Network is a payment network. The total number of non-cash payments in the euro area, comprising all types of payment services in 2021 is 114.2 billion payments [Ban22]. VISA is by far the most common credit card issuer with a market share of 52.8% of cards in circulation [For23]. Their payment volume for the 12 months ended June 30, 2022 is 255.4 billion payments [VIS22]. The amount of 10,000 payments is approximately the number of payments processed within three seconds in the euro area or by VISA in one second. We recognize that currently the overall payment volume in the Lightning Network is not near these values, however given the ambition to match this payment volume we decide to take such comparable payment sample. 10,000 payments thus are an appropriate number of payments to test in our Lightning Network model.

In each experiment we test the delivery of 10,000 payments from a randomly chosen sender to a randomly chosen receiver of the payment for a randomly chosen amount of satoshis (sats). One satoshi is a 100 millionth of a Bitcoin. The sending node and the receiving node are drawn from the Oracle Lightning Network (see chapter 4.1.2) and have to be two distinct nodes.

Regarding the payment amount, we choose a minimum amount of 10,000 satoshis. 10,000 satoshis is currently equivalent to approximately 2.30 USD or 2.10 EUR (March 7th, 2023). Even though smaller amounts can be sent, we consider this to be a reasonable lower bound for the payments, because otherwise the base fee for relaying payments in the Lightning Network will become more meaningful and a standard shopping event will rarely produce a lower amount to pay than this amount. We set the upper bound of the random payments at 1,000,000 satoshis, which is equivalent to approximately 230 USD or 210 EUR. We assume that the majority of general household payments lies within this range and for amounts higher than this people might consider paying on the bitcoin main chain to avoid settlement risks or not hold a considerable amount of liquidity in payment channels.

```
def create_payments(_graph, _number_of_payments, min_amount, max_amount):
    random.seed(1337)
    if (len(_graph.network.nodes())) < 3:
        logging.warning("graph has less than two nodes")
        exit(-1)
    _payments = []
    while len(_payments) < _number_of_payments:
        _random_nodes = random.sample(list(_graph.network.nodes), 2)
        src_exists = _random_nodes[0] in _graph.network.nodes()
        dest_exists = _random_nodes[1] in _graph.network.nodes()
        amount = random.randint(min_amount, max_amount)
        if src_exists and dest_exists:
            p = {"sender": _random_nodes[0], "receiver": _random_nodes[1],
              "amount": amount}
            _payments.append(p)
# write payments to file
    ndjson.dump(_payments, open(initial_payments_file_name, "w"))
```

Listing 5.1: Method that generates the 10,000 payment triples for the simulation.

The method produces 10,000 payments and writes the data to a file for future use.

5.3 Finding a Feasible Flow

For the Lightning Network the most important criterion is to be able to deliver a payment amount reliably, while not locking liquidity in the network unnecessarily. When a payment delivery method generates a path from sender to receiver it may fail to find a feasible flow, because the payment amount is larger than the capacity. When the algorithm fails to find a feasible flow because of such cut in the network graph it returns `no_path_found`. When no path can be found, the sending node will not try to send the payment. We analyze how often the payment could not be delivered, because the payment delivery method could not establish a feasible flow.

We use “D” to describe the Dijkstra algorithm for finding the flow, and “MCF” for the MinCostFlow path finding. The term “fees” describes the fee-based cost function, whereas “prob” describes the probability based cost function. “MCF(prob, learning)” describes the experiment where we retain knowledge about the success and failure of sent payments in the Uncertainty Network, when we apply learning to multi-part payments based on success probability as a cost function.

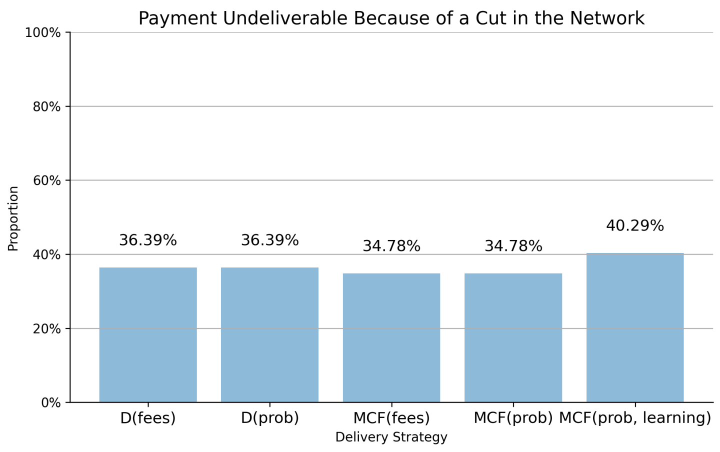

Figure 5.1: Proportion of undeliverable payments, N=10,000 per payment delivery strategy

Figure 5.1 shows that for slightly more than one third of the payments in the payment set, no path can be found for the given graph. This is not surprising, because payment nodes are sampled randomly, no social relationship for payments is taken into account, neither direct nor indirect, which usually is the case when conducting payments in real life and establishing payment channels. We consider this infeasibility of payments a structural problem. We hold this assumption because opening a channel usually follows some prior contact, be it through personal relationship or an economic motive.

When it can be determined that sending a payment is impossible, sending failing payments can be avoided. We don't consider this to be a lack of reliability, because there is no uncertainty about the payment being settled successfully or not. Consequently we disregard all cases in our analyzes, where it is possible to tell before sending a payment that the amount could not be delivered.

We recognize that the ratio of not finding a path between the two nodes in the payment set is highest in the experiment with MinCostFlow payment delivery method that retains knowledge about the liquidity balances in the network. Learning leads to sending a payment less often. This can only be the case, because the solver trying to find optimal flow refers to the information in the Uncertainty Network and receives the signal of missing liquidity, where initially this information would not have been available (and payments failed after trying to send the payment). Thus, learning provides more knowledge about cuts in the network.

In the Lightning Network it is possible for every participant to engage into opening a channel to another node, to mitigate the risk of cuts because of missing edges. With this knowledge of infeasible flows for payments, participants in the network can make a more meaningful decision about opening a channel with a channel partner. They can evaluate the reason for the cut, because having a cut might not (only) be due to a missing channel in general, but it can also stem from a lack of liquidity. This provides more opportunities for remediation. One can decide to renew a channel (close and open) with more liquidity or to engage into regular swaps that redistribute liquidity between two nodes to avoid drained channels.

5.4 Payment Delivery Strategies

We look at four different strategies for payment delivery that are available to participants given the information from the gossip graph and one payment delivery strategy that requires hidden information. This can be realized for example by participants sharing private information or establishing an omniscient router.

5.4.1 Single Payment Optimized for Fees

We replicate the standard payment strategy in the Lightning Network to find a payment path that incurs the least fees possible for delivering the payment amount. We use Dijkstra's algorithm to find the shortest path for a payment in one sum and use the fees for the payment amount on the path as the cost function. With this experiment we create a baseline for network reliability, given its current popularity with participants in the Lightning Network.

During execution of the 10,000 payments, we don't adjust the fees after each payment to signal the new liquidity balance. Node implementations only allow a certain number of updates in a specified interval. For the core lightning implementation this is up to four times a day. This rate limiting mechanism serves to avoid an extensive load on the gossip network. Thus, we assume that during the execution of the 10,000 payments no fee update would be propagated. Additionally, if a fee update is sent immediately after a change in liquidity that requires signaling, it might - in the worst case - take more than 10 minutes for a message to reach all nodes in the network [GRT22].

5.4.2 Single Payment Optimized for Success Probabilities

We replicate a probabilistic payment delivery strategy. For this we calculate the success probability of a payment with respect to a uniform liquidity distribution in the channel and treat the negative logarithm of this probability as the cost for the edge on the graph (see formula 2.2). Then we use the Dijkstra algorithm to find the shortest path on this success probability graph.

We expect to see better results than when using the Dijkstra algorithm with fees as a cost function, because after a few payments have been processed, the cheap channels should be depleted first and a fee based delivery method should always run into the same cheap channels, leading to failing payments, because of a lack of liquidity. This is in contrast to the success probability based Dijkstra algorithm for path finding, which prefers large channels, because they return a higher success probability for the payment along this edge. This signals a potentially higher liquidity balance and allows for more payments before liquidity is depleted by the payments processed.

However, over time the same problem will arise as with using a fee based algorithm. The cheapest channels are chosen and depleted, just that this time it is on the lowest probability, given by the payment amount and the channel's capacity, and not the lowest fee.

5.4.3 Multi-Part Payments Optimized for Fees

We model multi-part payments on the fee graph, optimizing the minimum cost flow problem. This properly represents multi-part payments, where the total amount can be split into several separate payment onions, which we call attempts. This demonstrates that a payment in the Lightning Network is a flow. We chose the fees for the flow on the edge as the cost, which resembles the experiment using Dijkstra's algorithm. The MinCostFlow solver now can implement the strategy to split up payments and use channels with a low fee but with a capacity lower than the payment amount, which have been excluded before.

We use the pickhardt payment library for this, in particular the `pickhardt_pay` method, where we set the parameter `mu=1000`. This parameter controls the balance between uncertainty cost and fees in the solver, with a high value giving preference to the fees.

Compared to the single payment method for fee based delivery, we expect a lower failure rate for the payments, because the underlying graph provides more feasible flow. More edges can potentially carry the flow.

5.4.4 Multi-Part Payments Optimized for Success Probabilities

Similarly, we model multi-part payments on the success probability graph. The optimization is more complex, because of the interdependence between the higher success probability for a split flow, but the joint probability of all flows reducing the overall probability. Splitting the payment in flow with a lower amount increases the success probability while at the same time the overall success probability for the complete payment to not fail is reduced with every flow with a success probability less than 100 percent. The expected value – the product of all probabilities for all channels – can be solved for as long as the cost function is convex. This is the case in our model.

We use the pickhardt payment library for this, in particular the `pickhardt_pay` method, where we set the parameter `mu=0`. This parameter controls the balance between uncertainty cost and fees in the solver, with a value of 0 using the uncertainty cost.

We expect better results than in the single payment method for the success probability based payment delivery, because the underlying graph provides more opportunities for an optimal flow, because more edges can potentially carry the flow. We also expect better results than with the fee-based multi-part payment delivery, because the flow will be split to use channels with a higher overall probability, which makes a reliable payment more likely. This means that channels will be preferred with a higher capacity and in consequence potentially higher liquidity, which leads to slower depletion of a channel, compared to a method that ignores channels' capacities.

5.4.5 Learning about Liquidity in the Network

How could payment delivery look like, if there was complete information about the liquidity balances in the channels? Or if such knowledge could be gained by cooperation? Can this be achieved for example through one omniscient router that learns about all payment onions sent, and with this knowledge establishes a belief about liquidity balances in the network? If this knowledge was applied to routing, how would this influence failure rates in the Lightning Network?

We simulate the knowledge that an observer of the network gains from monitoring all `send_onion` method calls over the course of our set of 10,000 payments. We achieve this by introducing and maintaining a belief network, the Uncertainty Network, which learns during executing the payment sample and correspondingly adjusts the payment flows. This belief network is implemented in the pickhardt payment library as the Uncertainty Network, see chapter 4.1.3.

We execute the payment delivery method of multi-part payments with the MinCostFlow strategy on the success probabilities as a cost function, better known as pickhardt payments [PR21]. From every payment delivered or failed we adjust the estimate of minimum and maximum liquidity in the two channels between the two nodes. When solving for the optimal feasible flow for a payment amount we calculate the conditional probability for an edge based on our belief of the minimum and maximum available amount of liquidity, in contrast to building the probability based on the total capacity given from the gossip graph.

Instead of resetting the Uncertainty Network before each method call for a payment delivery, we now maintain this Uncertainty Network for the session of all 10,000 payments and feed all results from payment tries into the learning process to improve our belief of the liquidity distribution.

Because we assume that probability based payment delivery is better than fee based delivery with regards to channel depletion in the network, we conduct the experiment with the probability cost function.

5.4.6 Aggregation of Payment Flow

In this experiment an omniscient observer acts as an intermediary and collects all successful payment flows in the network. This omniscient observer aggregates all payment amounts on the used edges. For this analysis, a flow describes a payment amount that is sent across one edge between to adjacent nodes in a payment onion.

We monitor the edges that the successful payments in our payment sample travel. Given the sufficiently large sample of 10,000 payments we expect payment flows to use some edges multiple times. We record the number of times that an edge is used and the corresponding payment amount, and calculate one aggregate flow amount on this particular edge.

In the next step we analyze, if for an edge between two adjacent nodes there is a flow in the opposite direction. If this is the case, then these two aggregate flow amounts are offset against each other, and one net flow on one of the two edges remains. We then compare the sum of all net flow amounts after this netting procedure to the previous sum of all aggregate flow amounts. For our analysis we take the most successful selfish strategy, multi-part payments with success probability as a cost.

5.5 Generalization of Results

To validate our findings we conduct further experiments. We test the failure rates on different liquidity balances, payment sets and topologies.

5.5.1 Randomization of Liquidity Balances

In the course of Lightning Network model graph generation, we have to make an assumption about the liquidity distribution in the channels. To generalize our findings we conduct our experiments on 20 different liquidity distributions on the Lightning Network model graph.

When generating the Oracle Lightning Network (see chapter 4.1.2), we assign liquidity balances to the channels. We randomly draw values between 0 and 1 from a uniform distribution that represent the split of the capacity in the two channels between adjacent nodes and the corresponding liquidity balances. Based on this liquidity distribution we execute the 10,000 payments in the set with each of the payment delivery strategies. We repeat this process for a total of 20 different random liquidity distributions.

5.5.2 Randomization of the Payment Set

We analyze the results for different sets of payments. We take a sample of five different payment sets of randomly selected 10,000 payments each and test them with the strategies D(fees) and MCF(prob) against our base case Lightning Model graph. These two strategies are structurally the most distinct, because one is using the Dijkstra algorithm, the other MinCostFlow, and one is using a fee-based payment delivery method, the other a probability based payment method.

We generate the payment sets following the process described in chapter 5.2. From an Oracle Lightning Network (see chapter 4.1.2) we sample two nodes, one being the sender, the other one being the receiver, and draw a payment amount between 10,000 satoshis and 1,000,000 satoshis from a uniform distribution.

5.5.3 Centrality

When sending payments from sender to receiver the sender tries to find a feasible flow to send the payment amount. This means that the channels need to signal a high enough capacity. If for capacity reasons no path can be established, then further questions regarding the possible liquidity along the channels need not be considered. Thus, the topology plays an important role, in particular the connectedness of the nodes and the available liquidity in the most connected nodes.

We look at how the failure rate changes for the cases where a path exists between sender and receiver after we eliminated the most central nodes. For all nodes in the Oracle Lightning Network graph we measure the betweenness centrality. From our base graph we then in three steps delete the 100 most central nodes, the 200 most central nodes and the 300 most central nodes. For each of these three graphs, in addition to our base graph, we execute 10,000 payments with each of the five payment delivery strategies and observe the failure rates

5.5.4 General Graphs

We construct a random graph to analyze, if our findings can be replicated and translated to potentially any existing graph, not just the Lightning Network model we derived from `listchannels`. We take an Erdős-Rényi random graph $G(n, M)$, an undirected graph with n nodes and m edges picked uniformly at random [ER60]. We add a parallel edge in the return direction if necessary. The result of choosing an Erdős-Rényi graph is a stochastically random graph and all graphs with the respective number of nodes and edges are equally likely. To avoid unwanted effects from comparing too different graphs, we choose the same number of nodes (3,299) and edges (18,537 channel pairs) as is in our Lightning Network model graph. We assign the capacities randomly by sampling channel capacities from the Lightning Network model graph, and we follow the same process with the fees.

As with our Lightning Network model, we randomly draw 10,000 payment triples (sender, receiver, amount) from the nodes in the network and execute each of these 10,000 payments with each payment delivery strategy on a random graph. We repeat this process for five different random graphs.

6 Results

To analyze if a Price of Anarchy exists, we run a set of experiments and compare the failure rates of payments for different payment delivery strategies. We manipulate parameters such as the liquidity balances in the model graph, the payment sets and the topology of the graph and re-test to validate our results and analyze, if our findings can be generalized.

6.1 Payments

We test for payment delivery of 10,000 payments with a particular payment delivery strategy on our model graph with 3,299 nodes and 37,074 edges.

The payment amount for the 10,000 payments is uniformly distributed. The mean payment amount is 501,068 sats, roughly equivalent to 115 USD or 105 EUR.

Table 6.1: The payment amount for the 10,000 payments is uniformly distributed.

	amount
count	10,000
mean	501,068
std	285,366.28
min	10,013.00
25%	254,692.75
50%	497,127.50
75%	744,401.50
max	999,962.00

Sending a payment leads to one of three results:

1. Based on existing knowledge, no path can be established between sender and receiver.
2. The total amount has been delivered successfully.
3. A path was found, but the delivery fails, because the assumed liquidity is not available in one or more of the channels.

6.2 Selfish Payment Delivery

Based on the Lightning Network graph as of January 14th we allocate liquidity balances randomly in the channels. Given this liquidity state we execute 10,000 payments with each of the payment delivery strategies. We calculate the failure rate of the payments by dividing the number of failed payments by the number of payments where a path was found from sender to receiver, assuming maximum possible liquidity. We repeat this process for in total 20 different liquidity states.

6.2.1 Failure Rates

In this section we analyze a set of payment delivery strategies, which we assume to include the optimal and selfish strategy for a participant. Focusing on the cases where it seems possible to find a feasible flow from sender to receiver, we measure the proportion of payments effectively succeeding or failing.

The results displayed in figure 6.1 show a clear distinction between the cost functions when it comes to payment delivery failure rates.

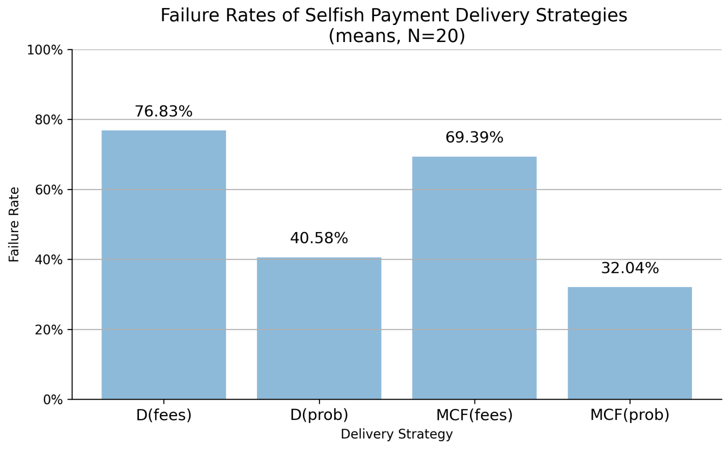

Figure 6.1: Failure rates for selfish payment delivery strategies; means for a sample of 20 graphs with random liquidity balances and 10,000 payments for each strategy on each graph.

6.2.2 Failure Rates over Time

If there is a liquidity drain in channels over time, then this would show up in our sample of 10,000 payments. And indeed, we can verify this by looking at the failure rates after processing blocks of 2,500 payments. Failure rates of fee based payment delivery strategies rise significantly over time, up to around 80% in the example setup (see figure 6.2).

Probability based payment delivery strategies have increasing failure rates as well, however slower and the rate of failing payments remains below 40%.

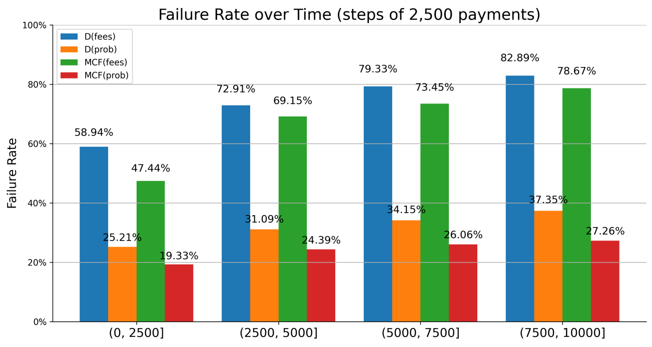

Figure 6.2: Failure rates for selfish payment delivery strategies over time, split in bins of 2,500 payments each.

Figure 6.3 displays the change in failure rate from one bin of 2,500 payments to the next. For example in the payment delivery strategy D(fees) the failure rate for 2,500 payments increased by 13.97%-points from the first set of 2,500 payments to the next set of 2,500 payments.

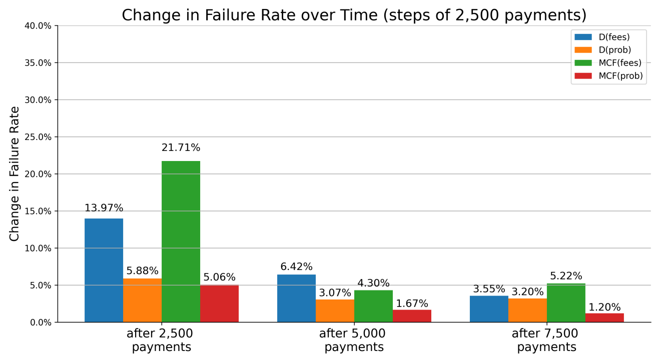

Figure 6.3: Change in failure rates from one bin of 2,500 payments to the following bin.

6.3 Beneficial Payment Delivery Strategies

We also test how failure rates could be kept low through beneficial payment delivery strategies. We assume an omniscient router exists, who knows all payments sent and their outcome. First we analyze the failure rates when the belief about payment balances is adjusted according to the outcome of payments attempted. Then we analyze if netting can be applied and if failure rates in the Lightning Network model graph can be improved.

6.3.1 Payment Delivery Strategy with Learning

Failure rates for payment delivery strategies show lowest values for multi-part payments based on success probabilities with learning.

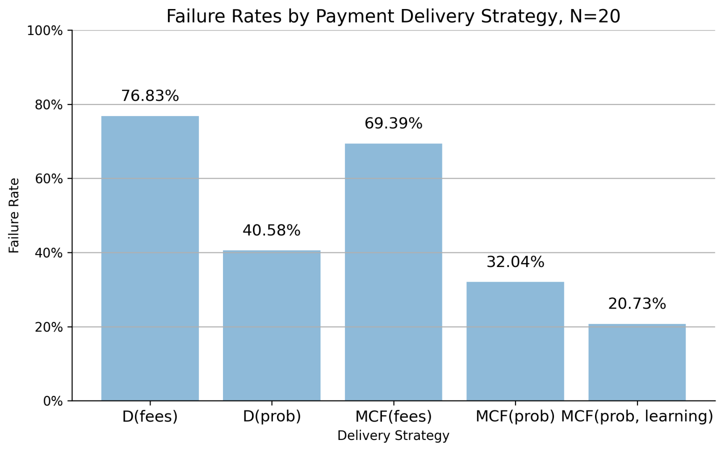

Figure 6.4: Failure rates for non-cooperative and cooperative payment delivery strategies.

Comparing the failure rate of the payment strategy with learning to the selfish strategies in figure 6.4, we recognize a significant drop in the failure rate of payments by 35% or 11 percentage points from 32.0% to 20.7% compared to the most beneficial selfish payment delivery strategy. Next, we analyze if there is a significant difference in the non-cooperative and selfish strategies compared to the strategy with learning.

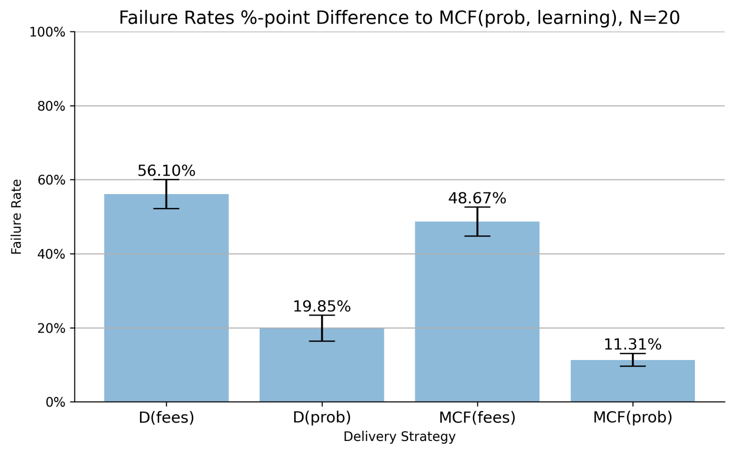

Figure 6.5: Difference in failure rates for non-cooperative payment delivery methods compared to pickhardt payments with learning, standard deviation represented by error bar.

For each difference in failure rates between the selfish payment delivery strategy and pickhardt payment with learning we calculate mean, min, max and standard deviation after 20 runs with different liquidity balances and 10,000 payments for each payment delivery strategy.

Table 6.2: Failure rates are significantly lower when using pickhardt payments with learning. 20 runs with random liquidity balances and 10,000 payments each.

Payment Delivery Method	N	M	min	max	SD
D(fees)	20	56.10%	45.61%	59.48%	3.89%
D(prob)	20	19.85%	15.46%	29.06%	3.50%
MCF(fees)	20	48.67%	38.55%	53.31%	3.90%
MCF(prob)	20	11.31%	9.44%	15.01%	1.74%

The selfish payment delivery strategies demonstrate significantly higher failure rates than the most beneficial strategy.

Table 6.3: Two-sided t-tests show with high confidence that selfish payment delivery strategies are less reliable than the cooperative strategy of pickhardt payments with learning.

Δ non-cooperative and cooperative strategy	t	df	p	M	SD
D(fees)	64.51	19	$p < .001$	56.10%	3.89%
D(prob)	25.39	19	$p < .001$	19.85%	3.50%
MCF(fees)	55.83	19	$p < .001$	48.67%	3.90%
MCF(prob)	29.09	19	$p < .001$	11.31%	1.74%

These two-sided t-tests reveal a significant difference in mean failure rates from pickhardt payments with learning, with a probability $p < .001$ for each selfish payment delivery method (N = 20, see table 6.3).

6.3.2 Failure Rates over Time for Payment Delivery Strategy with Learning

When starting our simulation, the omniscient router has no prior knowledge and learns over time, beginning with the first payment. Figure 6.6 displays the failure rates of the payment delivery strategies over time.

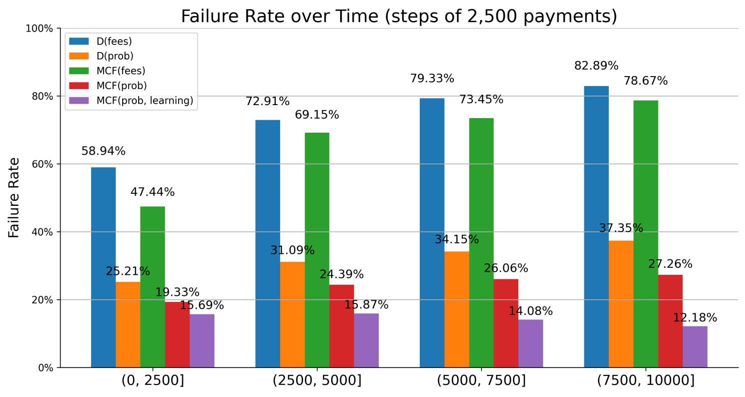

Figure 6.6: Failure rates for sets of 2,500 payments for non-cooperative and cooperative payment delivery strategies.

While the failure rates of payments delivered with selfish strategies increase over time, the failure rate decreases for the omniscient observer from 15.9% to 12.2%.

This also shows in the change in failure rates of payment delivery strategies from bin to bin in steps of 2,500 payments as depicted in figure 6.7. The increase in failure rate decreases over time for the D(fees) strategy, from an increase of 13.97%-points to an increase of 3.55%-points. For the strategy MCF(prob, learning) there is an initial slight increase from the first 2,500 payments to the next 2,500 payments followed by a decrease in failure rates of 1.79%-points and 1.91%-points for the remaining payments.

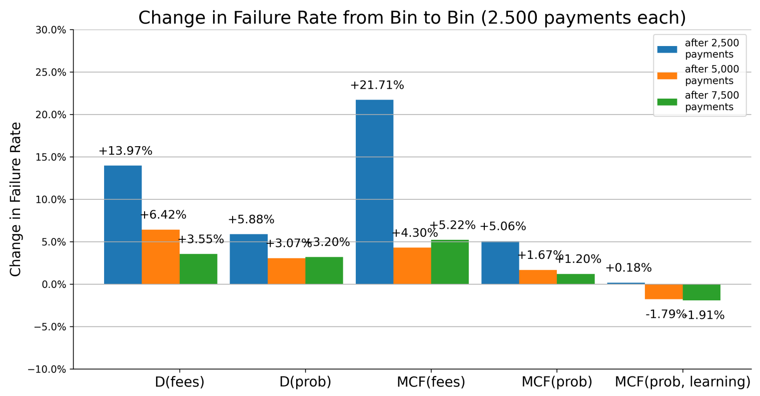

Figure 6.7: Change in failure rates from one set of 2,500 payments to the next in %-points for non-cooperative and cooperative payment delivery strategies

6.3.3 Payment Delivery Strategy with Netting

We simulate an omniscient observer that acts as an intermediary and collects all successful payment flows in the network from the most successful selfish strategy, multi-part payments with success probabilities as a cost function. This omniscient observer aggregates the payment amounts on the used edges as described in chapter 5.4.6 and builds a net flow amount on the edges as a remainder. We investigate, if it is possible to further increase the probability of successful payment delivery.

In the course of the 10,000 multi-part payments we tried 51,636 edges to send the payment from the senders to the receivers. To successfully settle the payment amounts, they were passed across 33,235 edges. The total payment amount in motion is equivalent to the payment amounts times the used edges of the respective payment. This equals a gross amount of 10,148,920,560 satoshis.

In the first step of collecting all multiple uses of an edge and sum it into one aggregate payment amount for this edge, we find that these payments have travelled along 7,065 distinct edges out of the total of 37,074 edges in the Lightning Network model graph. Progressing to the main idea of the experiment, netting flows in opposite channel pairs between to nodes, we generate one composite edge from both these edges in a channel pair.

With this, we reduce the number of flows and join 2,824 edges that originally had bi-directional flows. The remaining number of distinct edges used is now 4,241 for the successfully settled 4,939 payments.

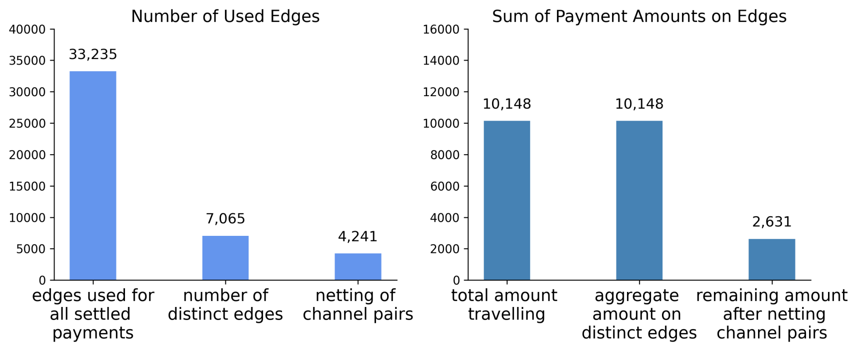

Figure 6.8: Number of used edges and total payment amount in motion for successful payments.

We net the amounts in the channel pair. Following this, we are left with only one of the two edges carrying the remainder of the two payment amounts. The sum of these remaining amounts on the edges is the net flow amount in motion. This net flow amount in motion decreases from 10,148,920,560 satoshis to 2,631,277,546 satoshis. Aggregating the payment amount on the edges and offsetting the payment flows on opposite edges of successful flows before settlement thus reduces the amount of money travelling in the network by 74.07%.

6.4 Price of Anarchy

We established that a Price of Anarchy exists under the assumption that welfare in the network can be measured as its reliability with which a payment can be successfully delivered. We also assume that participants reflect this goal when finding their best possible move in optimizing their a priori estimate of the success probability for the payment.

We found that among the analyzed strategies the effective payment reliability is in the range of the reliability achieved with strategy D(fees) in the worst case and MCF(probability) in the best case. We constituted that under the assumed preference function players shift their strategies towards the payment delivery method

MCF(probability), and as such this is the best strategy for a sender. However, other preference functions might include a consideration of fees to reflect the transaction cost of the payment itself. In this case, multi-part payments are still the best option amongst the strategies proposed for fee-based payment delivery, and a mixed strategy of MCF(probability) with MCF(fees) might be the payment method that matches the optimum of such preference function.

For our calculation of the Price of Anarchy we refer to the reliability of MCF(probability) as the best strategy for the sender. We also calculate the ratio with the MCF(fee) strategy to show an upper bound with the optimal fee-based payment delivery method. Thus, the reliability rates we use for our calculations are 32.04% for MCF(probability) and 69.39% for MCF(fee).

We demonstrated that strategies that increase the welfare for the network exist, but cannot be achieved by players without cooperation. We also showed that the strategy with netting does not give us a quantifiable result. For this reason we resort to the strategy MCF(prob, learning), pickhardt payment with learning, for our estimate of the reliability of payments in the Lightning Network in a cooperative strategy. The reliability rate in this case is 20.73%,

We calculate the Price of Anarchy ρ to quantify the inefficiency of the equilibrium of selfish strategies as we showed in chapter 3.2 and adopt the measured reliability rate as the outcome of the objective function. Following the result of our simulation for the ratio with the best selfish strategy we find

$$\rho = \frac{32.04\%}{20.73\%} = 1.5456$$

Taking into account that other preference functions might exist that show a preference for fee based payment delivery over probability based payment delivery, we also calculate the Price of Anarchy for MCF(fee) as

$$\rho_{MCF(fee)} = \frac{69.39\%}{20.73\%} = 3.34732$$

6.5 Generalization of Results

We validate our findings with regards to other random samples of payments and randomly distributed liquidity balances to allow for generalization of our results.

6.5.1 Randomization of Liquidity Balances

As we can see in table 6.4, in particular the differences between the payment delivery strategies have a rather low standard deviation. We find the results to be highly significant, the liquidity distribution does not have an impact on the relative ranking of the overall failure rate.

Table 6.4: Confidence intervals of payment delivery strategies and difference between strategies calculated on the sampled liquidity balances.

Payment Delivery Method	N	M	99% CI	SD
D(fees)	20	76.83%	[73.93%, 79.73%]	4.53%
D(prob)	20	40.58%	[33.48%, 47.63%]	11.10%
MCF(fees)	20	69.39%	[66.19%, 72.59%]	5.00%
MCF(prob)	20	32.04%	[25.73%, 38.35%]	9.86%
MCF(prob, learning)	20	20.73%	[15.40%, 26.05%]	8.32%
Δ D(fees)	20	56.10%	[53.61%, 58.59%]	3.89%
Δ D(prob)	20	19.85%	[17.62%, 22.09%]	3.50%
Δ MCF(fees)	20	48.67%	[46.17%, 51.16%]	3.90%
Δ MCF(prob)	20	11.31%	[10.20%, 12.41%]	1.74%

6.5.2 Randomization of the Payment Set

The base case has a mean value for the failure rate of the D(fees) strategy of 73.57%. The 99%-confidence interval for the failure rate of D(fees) for a variation in the payment set is [71.39%, 74.49%].

The base case has a mean value for the failure rate of the MCF(prob) strategy of 24.27%. The 99%-confidence interval for the failure rate of D(fees) for a variation in the payment set is [23.86%, 25.20%].

Base case describes the initial liquidity distribution of the Lightning Network model graph.

Table 6.5: Confidence intervals of example payment delivery strategies tested against different payment sets with the base case liquidity balance distribution.

Payment Delivery Method	N	M	99% CI	SD
D(fees)	5	76.83%	[71.39%, 74.49%]	4.53%
MCF(prob)	5	32.04%	[23.86%, 25.20%]	9.86%

In the context of the results for the difference in failure rates for the strategies compared to the most beneficial strategy for the Lightning Network, we consider the impact on the results from a change in the sample of the 10,000 payments to be negligible.

6.5.3 Centrality

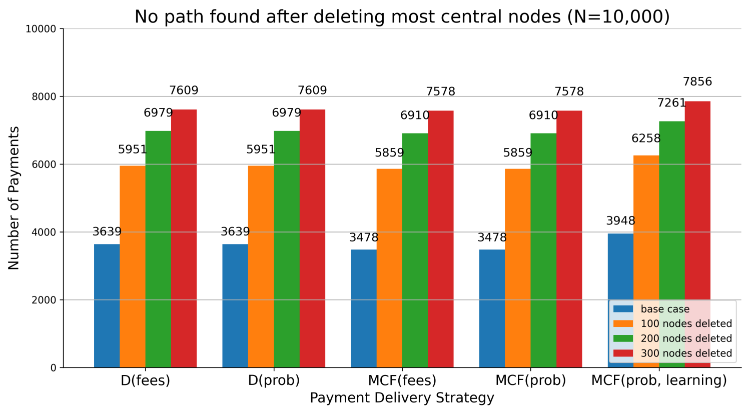

Figure 6.9: Number of payments where no path could be found for payment delivery strategies by number of most central nodes deleted

When the most central nodes are eliminated from the graph, we find that the number of cases where there is a cut in the network and no paths were found increases significantly, see figure 6.9. Eliminating the 200 most central nodes in our Lightning Network model doubles the number of impossible payments.

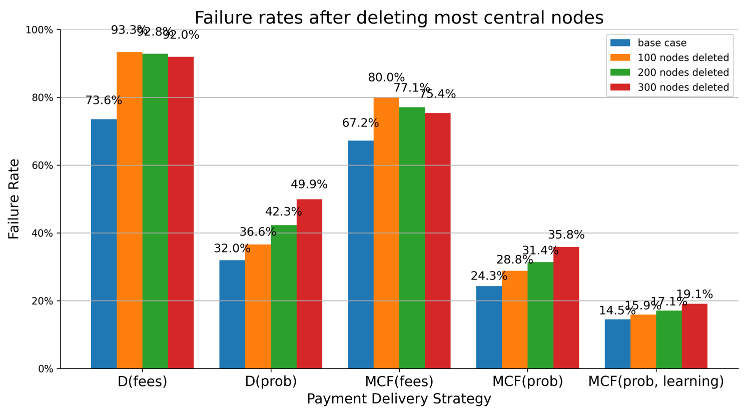

Figure 6.10: Failure rates for payment delivery strategies by number of most central nodes deleted

While generally the number of payments that can be sent through the Lightning Network drastically decreases, the failure rate of the payments for the fee-based payment delivery strategies are stable or decrease slightly (table 6.6).

Table 6.6: Failure rates for the payment delivery strategies after eliminating the 0, 100, 200 and 300 most central nodes from the Lightning Network model graph.

	most central nodes deleted			
Payment Delivery Method	0	100	200	300
D(fees)	73.57%	93.33%	92.82%	91.97%
D(prob)	31.98%	36.58%	42.30%	49.94%
MCF(fees)	67.22%	79.98%	77.06%	73.35%
MCF(prob)	24.27%	28.79%	31.39%	35.84%
MCF(prob, learning)	14.49%	15.87%	17.05%	19.08%

For the probability based payment delivery strategies we find that payment failure rates increase when the most central nodes are removed from the graph.

6.5.4 General Graphs

We find that the resulting failure rates for the different graphs are more pronounced than with the Lightning Network model, see figure 6.11. While the fee-based strategy

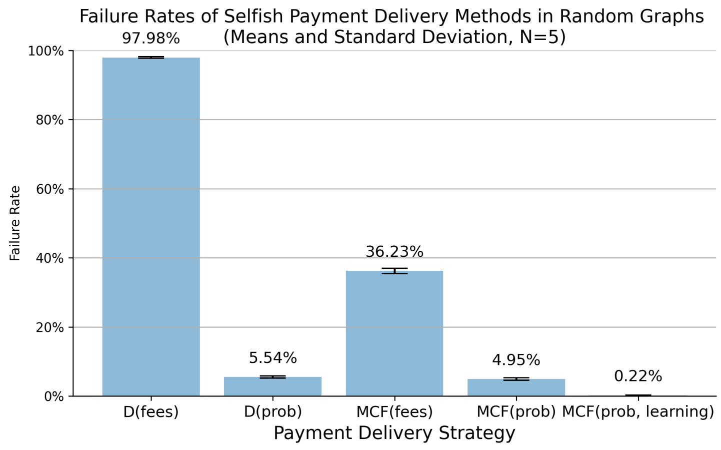

Figure 6.11: Failure rates for payment delivery methods in random Erdős-Rényi graphs, N=5

Table 6.7: The means for failure rates for the random graph are different to the Lightning Network model graph. Differences between the strategies are more pronounced.

Payment Delivery Method	N	M	min	max	SD
D(fees)	5	97.98%	97.79%	98.25 %	0.2052%
D(prob)	5	5.54%	5.20%	5.79%	0.2881%
MCF(fees)	5	36.23%	35.32%	37.18%	0.7555%
MCF(prob)	5	4.95%	4.49%	5.28%	0.3674%
MCF(prob, learning)	5	0.22%	0.13%	0.32%	0.0876%

with the single payment fails in 97.98% of the payment tries, the fee-based strategy for multi-part payments has less than half the failure rate compared to the Lightning Network model, 36.23%. However, this is still significantly worse than the probability based strategies. Regarding the single payment delivered with a probability based payment delivery strategy, the failure rate is only 5.54%.

Despite this already low failure rate, the multi-part payment strategy based on successful payment probability still performs better, with 4.95%. The cooperative strategy of multi-part payments with learning from previous payment tries clearly outperforms the failure rates with only 0.22% of payments failing (see table 6.7).

The data shows, the differences in the failure rates of the selfish payment strategies compared to the cooperative strategy, the multi-part payment delivery strategy with learning, are directionally the same as with our Lightning Network model graph, see figure 6.12.

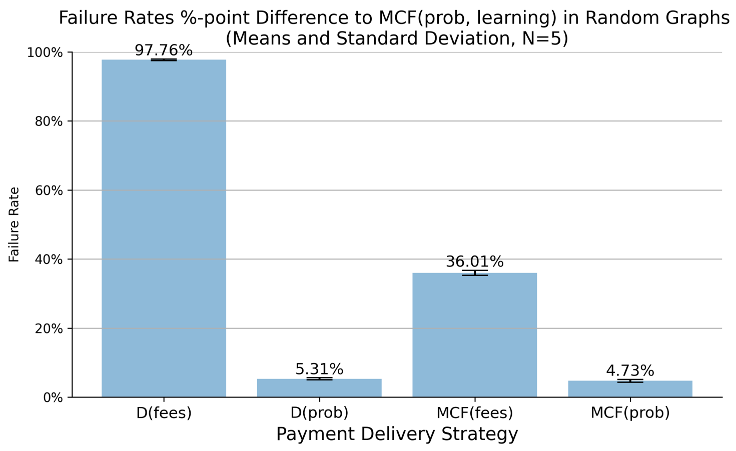

Figure 6.12: Difference in failure rates for non-cooperative payment delivery methods compared to pickhardt payments with learning.

The analyzes were conducted on five different random graphs, however the standard deviation is very low. For a graph with random properties, the payment delivery strategies perform with equal results as for the Lightning Network model.

Table 6.8: Differences in failure rates for non-cooperative payment delivery methods compared to pickhardt payments with learning are more pronounced on all five random graphs than on the Lightning Network model graph.

Payment Delivery Method	N	M	min	max	SD
D(fees)	5	97.76%	97.47%	98.01%	0.2275%
D(prob)	5	5.31%	4.88%	5.54%	0.2991%
MCF(fees)	5	36.01%	35.07%	36.90%	0.7112%
MCF(prob)	5	4.73%	4.30%	5.15%	0.3779%

7 Discussion and Future Work

In this work we define the reliability of the Lightning Network as a necessary requirement for its usefulness. We chose the failure rate of payments as a measure for unreliability and as the central driver of the participants' utility. Under this measure we calculate the Price of Anarchy.

We analyze payment delivery strategies such as reducing the amount of fees paid for a single payment or multi-part payment, and increasing the success probability for a single or multi-part payment. These strategies belong to a set of strategies that reflect the individual preference of network participants. We treat them as selfish strategies, because while optimizing individual preferences, they don't achieve the maximum possible welfare for the Lightning Network defined as minimal failure rate of payments sent. They represent competitive strategies, because the participants compete for liquidity in the payment channels.

We investigate the reliability of the Lightning Network when participants apply such competitive strategies, measured by the rate of failing payments. We also analyze if other strategies are more beneficial for the network and achieve a higher welfare by reducing the rate of failing payments. We look at how this reliability changes over time, when more beneficial strategies are used, strategies that require general knowledge about the graph or information that only the participants possess.

7.1 Selfish Payment Delivery Strategies

Our results for selfish payment strategies show that there is a significantly higher proportion of failing payments when following a payment delivery strategy that minimizes fees, than when maximizing successful payment delivery probability (see figure 6.1). While Dijkstra on fees as a cost function has a mean failure rate of 76.83% and MinCostFlow on fees as a cost function has a mean failure rate of 69.39%, the probability driven delivery methods are approximately 35%-points lower. Dijkstra based on the success probability has a mean failure rate of 40.58% and MinCostFlow based on the success probability has a mean failure rate of 32.04%.

This is in line with our expectation. One reason for the higher failure rate of fee-based delivery strategies is that payments completely drain the liquidity of channels with low fees over time. Because of the algorithm always choosing the same cheap channels, they at some point don't carry enough liquidity and thus payments fail more often. This is the case for success probability based strategies as well, however this is effect is less pronounced than with the fee-based strategies.

7.1.1 Failure Rates over Time for Selfish Payment Delivery Strategies

If there is a liquidity drain in channels over time, then this would show up in our sample of 10,000 payments. And indeed, we can verify this by looking at the failure rates after processing blocks of 2,500 payments. Failure rates of fee-based payment delivery strategies rise significantly over time, up to around 80% in the Lightning Network model graph (see figure 6.2). Probability based payment delivery strategies have increasing failure rates as well, however slower and the rate of failing payments remains below 40%.

This demonstrates a higher robustness against payment delivery failures. The data shows that choosing a payment delivery strategy that is based on the success probability of the payment in the optimization not only offers a higher reliability but also a more stable success probability.

7.1.2 Minimum-Possible Failure Rate of Selfish Payment Strategies

We know that the optimal strategy is within the set of the strategies described above and thus we can establish the minimum proportion of failing payments within this set of payment delivery strategies. To achieve the maximum possible welfare for the Lightning Network through selfish strategies, we can conclude that participants should converge to using multi-part payments and a success probability based payment delivery strategy.

We recognize that this is not yet the case with participants in the Lightning Network, and there might be several reasons for this. Participants might believe that an information-efficient market exists with all nodes following the strategy of signaling liquidity by adjustment of fees. Also it is possible that node maintainers don't follow the strategy of signaling liquidity balances through fees but aim to set their fees following other goals. Node implementations might not allow for such payment delivery method and/or node maintainers don't yet feel comfortable using such strategies. Also, the participants' utility function might not just be a function of the reliability of the payment but also of the fees paid. And while fees paid for reliable payments might be higher than for a payment with less success probability, participants might still feel like there is a maximum amount for a transaction fee they are willing to pay. Consequently a failing payment is not as bad for them as for other participants, because if a payment fails for this lower amount, then they can still switch to other payment strategies.

In our model the maximum welfare and the minimum failure rate of the four selfish strategies is at 32.04%.

7.2 Cooperative Payment Delivery Strategies

With more information, which for example can be gained through cooperation, a more beneficial payment delivery for the network can be achieved, leading to an increase in welfare. Above results show that without intervention through for example rebalancing or opening of new channels, the payment delivery methods for selfish payment delivery strategies lead to steadily increasing failure rates in the Lightning Network.

7.2.1 Payment Delivery Strategy with Learning

In chapter 5.4.5 we introduced learning, implemented as maintaining a belief network about the minimum and maximum liquidity in the payment channels. This contributes to more accurately estimating the success of payment delivery.

Comparing the failure rate of the payment delivery strategy with learning to the selfish strategies in figure 6.4, we recognize a significant drop in failing payments by 35% or 11 percentage points from 32.0% to 20.7% compared to the most beneficial selfish payment delivery strategy. We consider this to be a substantial contribution to the reliability of the network.

When conducting probability based payment delivery, the observer can learn about failing payments and incorporate this knowledge in its belief about liquidity balances in the Uncertainty Network. This leads to a higher success rate for the payments being tried to deliver, because the success probabilities now don't relate to the capacity of the channel, but to our — more precise — estimate of the range of possible liquidity.

Thus, learning about liquidity balances from successful payments and failing payments increases welfare in the Lightning Network. This is a strategy that is unavailable to the individual participant and thus cannot be achieved in non-cooperative or selfish routing.

We conclude that there is a significant difference in welfare for the network between a strategy that uses collected information or cooperation, and strategies by selfish participants (see table 6.3). Given this difference in welfare between the selfish strategy and the cooperative observer strategy, we can assume that a Price of Anarchy exists. The strategy of learning provides a failure rate for the denominator in our definition of the Price of Anarchy.

7.2.2 Learning over Time

When starting our simulation, the omniscient router has no prior knowledge and learns over time, beginning with the first payment. Because there is a benefit in retaining knowledge and building beliefs about liquidity balances in the channels, this results in constantly improving payment delivery and a decreasing failure rate for payments (see figure 6.7).

While the failure rates of payments delivered with selfish strategies increase over time, the failure rate decreases for the omniscient observer from 15.9% to 12.2%. This is intuitive, because the a priori estimate of a payment success improves. Consequently the payment method avoids depleted channels and, as has been shown in figure 5.1, more frequently avoids payment delivery generally if there is no feasible flow. The payment delivery strategy with learning is not only more stable over time, but the more payment results are incorporated in the belief network, the lower the failure rate.

7.2.3 Concluding Remarks on the Strategy with Learning

The results of our experiments show that the payment delivery strategy with learning provides a significant benefit for the Lightning Network. It has the lowest failure rates among all five simulated payment delivery strategies.

It is necessary to have more information for this payment delivery strategy than is available from the gossip graph in the Lightning Network. A sending node alone can collect all information it receives from its own payments, and with this can build its own belief network, to better estimate the liquidity balances in the network graph. However, this information will not cover the full graph and the estimate will never be as good as in the case of the omniscient router who knows about all payments and their status. In addition, because the sender only knows the own payment attempts, the effect from other participants in the network executing their payments and changing channel balances cannot be reflected in this belief. Knowledge that an individual sending node carries is thus inferior to the knowledge that a general observer of the network can gain, because this information incorporates a more complete state of the graph. For this reason as well, the sender cannot achieve a strategy that is equally beneficial on its own.

This leads to the conclusion that without cooperation of the participants in the network the optimal outcome, the highest welfare possible, cannot be achieved and a Price of Anarchy exists.

Another aspect that needs to be considered regarding the question of achieving an optimal payment execution through cooperation is wether the utility of the sender might not solely depend on the success rate of the payments executed, other more soft factors might play a role as well, such as privacy. If information hiding is beneficial to the individual, this can lead to the situation that the single player has an interest in receiving information so it can route successfully, but might hide information or give misleading information to maintain the own privacy.

Analyzing the consequences of this dilemma exceeds the scope of this work and remains for future research.

7.2.4 Payment Delivery Strategy with Netting

We find that netting reduces the flow in the Lightning Network model graph while maintaining the liquidity balances after executed payments (see chapter 6.3.3). The flow can be reduced substantially for the set of payments. The number of payments executed approximates the number of payments executed by a major credit card company within one to three seconds. In consequence, within an acceptable time-frame for participants in the network, this measure allows to reduce the load on the graph and reduces the probability of payments failing.

From a research perspective we are faced with the challenge to translate this reduction of flow into a change in failure rates in the Lightning Network. A change in failure rate cannot be quantified, but we can safely assume that reducing flow for payments by netting will have a positive impact. Firstly, this reduces the liquidity moved in the network and thus reduces the amount of liquidity consumed. Because liquidity is scarce in the network, this increases availability of liquidity. Secondly, if a lower amount is moved between two adjacent nodes, the probability to succeed is higher than for a higher amount being moved, thus reliability increases. Thirdly, any payment amount that is not sent is a payment amount that does not fail. As a consequence, the number of payments in the network that could possibly fail, is reduced.

Because we cannot calculate a lower failure rate for the optimal cooperative strategy multi-part payments with success probability as a cost function with netting, we can only determine this failure rate of 20.7% to be an upper bound. This demonstrates that the Price of Anarchy of 1.5456 is in fact a lower bound, because if such measures were applied, the Price of Anarchy increases.

For practical purposes we only considered flows that settled successfully. After netting and moving less money through channels and draining less liquidity in the used channels, it is possible that a previously failing payment could succeed. The question of failure or success translates into a question of timing. If the payment was to have taken place in a later position in the payment queue, it might have succeeded. One way to safely test this hypothesis is to translate the payment delivery experiment into a multi-source multi-sink MinCostFlow problem with the sending nodes and the receiving nodes of the 10,000 payments as sources and sinks. This exceeds the scope of this work, and we leave this question for future research. We do conclude, however, that netting has a positive contribution to the welfare of the network. It does decrease the failure rate of payments, even though it cannot be quantified.

7.2.5 Failure Rate of Optimal Flow

Learning about liquidity balances from successful payments leads to a better payment flow than selfish strategies. Better in this case refers to higher welfare in the Lightning Network, defined as a lower failure rate of payments. We also demonstrated that aggregating payments and netting flow can decrease failure rates and contribute to a more reliable payment delivery.

We find that there is a significant difference in welfare for the network, between a strategy with information through cooperation, and the degree of welfare that is established by participants following selfish strategies. Given this difference in welfare between the selfish strategy and the observer strategy, we can assume that a Price of Anarchy exists.

Because it is unfortunately not possible to determine a resulting failure rate after applying netting strategies, we have to refer to the failure rate for the strategy with learning as a conservative estimate for the failure rate of an optimal flow for the payments in our Lightning Network model.

7.2.6 Price of Anarchy

We demonstrate that there is a significant difference in welfare in the Lightning Network model graph between selfish payment delivery strategies and those of an omniscient router or cooperation. We established a Price of Anarchy exists and calculate the lower bound for the Price of Anarchy to be 1.5456. Coordination increases the benefit of the Lightning Network, compared to competitive payment delivery strategies.

This finding provides a legitimate reason for lightning service providers to exist, because they are beneficial for the Lightning Network. They are beneficial for the network because they can learn information about channels when executing payments or even use netting to reduce the overall flow. Trampoline routing is a possibility to transfer information about payment flow as well as the outcome of a payment attempt, and providing this information is a form of cooperation. This reduces the Price of Anarchy, because with building a belief network they learn and decrease failure rates.

We point out that when we translate our payment delivery methods to the implementation in the real world, we observe that participants don't only use one round for finding the successful payment, but once the payment failed they start further tries, eliminating the previously failing channel. Thus, we see some learning strategy currently implemented, which increases success rates when the definition of a payment is extended from one attempt to a set of several rounds. This lowers failure rates and increases the welfare of the network, possibly reducing the Price of Anarchy.

Extending the set of strategies for players from those with only one action to strategies where players can conduct a sequence of actions with signals about the liquidity state in between can provide further interesting results. We leave this analysis for future research.

7.3 Generalization of Results

We validate our findings with regards to other random samples of payments and randomly distributed liquidity balances to allow for generalization of our results. We find none of these factors has an impact on our results. As we can see in table 6.4, our finding regarding the overall failure rates is still highly significant for random liquidity distributions. This is also the case for a change in the sample of the 10,000 payments, see chapter 6.5.2.

7.3.1 Centrality

When sending payments from sender to receiver the sender tries to find a feasible flow to send the payment amount. This means that the channels need to signal a high enough capacity. If for capacity reasons no path can be established, then further questions regarding the possible liquidity along the channels need not be considered. Thus, the topology plays an important role. In particular the connectedness of the nodes and the available liquidity in the most connected nodes.

We find that those highly connected nodes carry the most responsibility for maintaining a reliable network (see figure 6.10).

While generally the number of payments that can be sent through the Lightning Network drastically decreases, the failure rate of the payments for the different payment delivery strategies still show the same profile, in that the fee based strategies have a higher failure rate than the payment delivery strategies using success probabilities for finding the path. The cooperative payment delivery strategy has the lowest failure rates, compared to all other strategies.

Thus, the cooperative strategy performs best, even with central nodes removed. The Price of Anarchy exists, irrespective of the centrality of the lightning network graph.

7.3.2 General Graphs

The differences in the failure rates of the selfish payment strategies compared to the cooperative strategy, the multi-part payment delivery strategy with learning, are directionally the same as with our Lightning Network model graph, see figure 6.12.

Our results demonstrate that payment delivery with multi-part payments based on a success probability as a cost function leads to the lowest failure rate for payment delivery for participants, even on a random graph. The results also show that a cooperative strategy outperforms selfish strategies, as expected, and can even lead to a negligible failure rate of payments, for our sample of random graphs.

We also notice that the resulting means for failure rates in our Lightning Network model are worse than the means for the random graphs, which suggests that payment channel creation by participants in the Lightning Network is not the most efficient under the measure of payment reliability. Given that in the Lightning Network channel creation is in the hands of participants and the topology is the result of a network creation game [FLM$^+$03], we do however not calculate a Price of Anarchy. We believe that one cannot separate the underlying graph from the participants utility functions that give rise to a specific graph. The selfish routing strategy, the resulting equilibrium and the topology of the graph are tightly connected [AHWW20].

7.4 Future Work

We simulate fee-based strategies for payment delivery. In our context we consider fees to be signals for liquidity in channels. Looking at the results and highest failure rates for fee-based strategies, we conclude that they are a rather unsuitable indicator for availability of liquidity in channels. This might change in the future, but we explain why such change would not come without its difficulties.

If fees were to be seen solely as transaction cost, then this cost measure can be used as an objective function. However, measuring the amount of fees paid does not work without incorporating the probability of failure of payments. Finding the tradeoff and an equilibrium of individually optimal strategies between low costs and low failure rates is highly dependent on the utility function of the participants. A generalization for mixed strategies that is suitable for a Price of Anarchy under such mixed strategies is an additional direction for further research.

From the perspective of game theory, the Lightning Network is an infinite game. We treated the set of 10,000 payments as a suitable subgame of the infinite game and translated the findings for our model to the Lightning Network in general. We see the necessity to extend this assumption and differentiate between participants with sporadic payments and participants with continuous, frequent and recurring payments, because the considerations for an optimal outcome might be different. One type of players sees the singular use case, optimizing for the short term maximum use, while the other type considers future payments and takes a long term view. To one type of players this is a finite game, to the other type of players it is an infinite game. It is an open question if in the Lightning Network this mix of strategies leads to a different equilibrium (if one at all) for such types of players prevalent in the network.

We describe how a cooperative strategy might look like, in that players share their information to achieve better learning about liquidity balances. We briefly discussed that the utility of the sender might not solely depend on the success rate of the payments executed, other more soft factors might play a role as well, such as privacy. However, participants might see the benefit of more knowledge about liquidity balances and decide to share information. But we then face a problem of moral hazard, because participants might share wrong information to hide their liquidity balance in exchange for potentially right information by other participants. While this is not possible with trampoline payments, this can be the case for situations like friends-of-friends information sharing. Analyzing what measures would incentivize participants to truthfully share their information and if there is an equilibrium where a good enough belief about liquidity balances can be established under moral hazard makes further research necessary.

When we analyzed the benefit of netting, for practical purposes we only considered flows that did settle successfully. The net balance of payment balances in the channels is the result of summing flows from a sequence of payments. It is possible that payments failed on these channels, because liquidity was moved to the other channel before. This leaves the question to what extent flow could have been found for failing payments, if there was a different sequence of payments. If the payment was to have taken place in an earlier or later position in the payment queue, it might have succeeded. One way to safely test this hypothesis would be to analyze the payment delivery for a given set of payments as a multi-source multi-sink MinCostFlow problem. The benefit of such netting is higher, the more payments can be considered. At the same time, the problem gains in complexity by adding more source-sink pairs. How does the tradeoff look like for the benefit of less failing payments versus longer processing time for the algorithm? Is this possible in a reasonable amount of time?

While current challenges exist, there is room for improvement in efficiency and reliability for the Lightning Network, and these open research questions can provide ample possibilities for understanding existing problems better and propose solutions, that increase the benefit of the Lightning Network and help lead the way for the Lightning Network to become a major decentralized payment delivery network.

8 Related Work

The Price of Anarchy is a topic at the intersection of game theory and the domain of networking. While there is extensive research on these two fields, not much work has been published that applies these findings to the Lightning Network.

While research on the Price of Anarchy is abundant for common network problems, the focus on the Price of Anarchy in the Lightning Network mainly centers around its topology. Prior research on the Price of Anarchy with regards to failure rates of payments is scarce and instead uses for example the change in liquidity distribution in the channels as a measure for the Price of Anarchy (like [Pic22]).

Game theory is a topic that gained traction in the second quarter of the 20th century and was at the time driven by John von Neumann and John Nash. Their definitions of games and equilibria are still being used today [Jr50]. Since then a lot of refinements and amendments evolved that take the Nash equilibrium as the general case and apply it to more specific cases, which we discussed in chapter 3.1.4. The Lightning Network can be regarded as an infinite game where participants take their first action based on an unknown state of the graph, but can build an expectation on liquidity balances. They learn about the fit of the probability function from the response after sending a payment and can better specify their model when considering the next best move. We particularly refer to [EP19] and [CVMS20] for equilibria under misspecified models and equilibria in nonatomic games.

While initially game theory was mainly applied in an economic context, computer science more and more discovered this field for its own topics or applying computer science methods to the questions posed. Its application to routing problems in networks initially referred to urban traffic routing problems (taken from the third quarter of the 20th century like [Bra68]) and then network routing. The most prominent work in this space is by Tim Roughgarden [Rou05] who applies the concept of the Price of Anarchy to competitive routing, or selfish routing.

What is referred to as the Price of Anarchy dates back to the work of Elias Koutsoupias and Christos Papadimitriou on worst-case equilibria where they analyze the cost of the lack of coordination as opposed to the lack of information or the lack of unbounded computational resources, as has been done in computer science before [KP99]. They propose the ratio between the worst possible Nash equilibrium and the social optimum as a measure of the effectiveness of a system.

The translation of the Price of Anarchy to the Lightning Network in particular poses the problem to determine the equilibria to describe the effectiveness. How is the equilibrium defined, what is the objective function that measures the welfare of the network but can also be applied as a cost function to the participants in the network?

Avarikioti et al. describe the effectiveness of the Lightning Network under the measure of transaction fees paid. They analyze the network from a network creation game perspective [AHWW20] and the topologies of the Lightning Network that emerge. This is influenced by participants deciding selfishly on opening a new channel to a node, which incurs fees for the transaction on the blockchain, versus locking liquidity in a lightning channel. In the network creation game, the incentive of the participants is to maximize transaction fees paid and received by choosing to whom they connect.

An optimal graph structure for the Lightning Network is examined in [AJWW18]. They analyze an optimal fee assignment to channels and find that a star is a near-optimal solution for the topology. However, they assume the capacity of every channel to be infinite. This is the central drawback of their research.

As we have pointed out before, payments in the Lightning Network are a flow, however the capacity of the possible flow is unknown and cannot easily be determined nor adjusted. When we compare the Lightning Network payment delivery to routing problems in traffic or data networks, there is no problem in passing an infinite amount of cars or information along a route. It is merely a question of time, but eventually one will succeed. It is a tradeoff between bandwidth and time. However, the Lightning Network is different in that channels can be depleted, because a payment means to move money from one side of the channel to the other. Which at one point doesn't work anymore, because one side ran out of money, and more money cannot be moved, even if one waits a long time. In consequence, loosening this constraint when analyzing the Lightning Network eliminates the principal feature that leads to unreliability in the fulfillment of its core functionality and restrains the adoption of Lightning Payment today.

Liquidity and the uncertainty of payments has been analyzed by René Pickhardt et al. [PTBN21]. Using probability theory they model the uncertainty of balances and thus successful payment in the Lightning Network. They establish the probability for a payment of a certain amount to successfully be delivered and derive an estimate of the number of payment attempts for an amount to be delivered completely.

The major step forward in optimizing payment strategies in the Lightning Network has been performed by Pickhardt, Richter [PR21]. Building on the previous work of establishing success probabilities, they use min cost flow to find the most likely multi-part payment. The algorithm they develop works by updating the probability distributions with the information gained from both successful and unsuccessful paths on prior rounds. With this algorithm even large amounts could be delivered successfully. Additionally, they propose how to reduce the originally NP-hard problem of finding the cheapest multi-part payments to a linear min-cost flow problem by linearizing the fees, by dropping the base fee.

Research on the Price of Anarchy in terms of depleted channels has been published by Pickhardt before [Pic22]. They define the Price of Anarchy as the change in the shape of the distribution of expected channel drain arising from selfish routing strategies in comparison to the shape of that distribution if payments were centrally planned in an optimal way. They find that given a skewed random walk on the payment channels, failure rates become stationary, but less than 100% even for heavily skewed random walk, for a large number of payments. They observe that the highest Price of Anarchy is realized by nodes using a strategy that combines fee-based and success probability based multi-part payments. They also state that the criteria applied might not necessarily be a Nash equilibrium in that there is a dominant selfish strategy amongst the set of analyzed strategies.

We share the opinion that drain explains failure rates of channels and the change in shape is an appropriate measure of the reliability of the Lightning Network. However, we choose a different measure for the Price of Anarchy, failure rates, because we regard this as the effective cost to participants. Failure rates is a metric that can be observed and optimized for, while the shape of the drain is a result of the optimization by sending nodes and as such not measurable or observable by individually optimizing participants.

9 Conclusion

The Lightning Network is inherently unreliable, because of unknown liquidity balances in the payment channels. Participants in the network compete for liquidity in the channels. A lack of liquidity leads to failing payments. A significant number of failing payments renders the Lightning Network unusable, because it is too unreliable to fulfill its purpose of a payment network. In consequence, network reliability is a central measure for the utility of the Lightning Network.

We build a model to simulate the Lightning Network and implement different payment delivery strategies. We derive the model from the gossip graph of the Lightning Network on January 14th, 2023. Because no liquidity balances are available, we randomly assign liquidity balances to the channels. To avoid outliers we draw a random sample of 20 different liquidity distributions, which we tested on. Based on this Lightning Network model graph we executed sets of 10,000 payments for each of our payment delivery strategies.

The payment delivery strategies that are used to construct a flow for the payment vary in payment delivery reliability. We show that fee-based strategies lead to the highest failure rates of payments, compared to probability based strategies. Failure rates of fee-based strategies are twice as high as failure rates of the tested probability based strategies. We also show that the failure rates of fee-based strategies increase over time, because the channels with low fees are depleted quicker. From these results we assume that the optimal individual strategy under the objective function of payment reliability is a multi-part payment that derives the payment flow based on the success probability of the flow.

We introduce cooperative strategies in the Lightning Network via an omniscient observer with knowledge about the outcome of the payments sent by nodes, and their corresponding flows. We establish a cooperative strategy by sharing knowledge about successful and failing payments with an omniscient router, who with this information establishes a belief about payment balances and decides on the flow for a payment. We also investigate if aggregation of flows is possible and if this can have a positive influence on the welfare of the Lightning Network model.

Our results from the simulation show that failure rates significantly decrease when payments are routed by the omniscient router who learnt about the minimum and maximum liquidity balances of channels. Compared to the non-cooperative strategy with the lowest failure rate, the failure rate decreases by 35%.

We established that a Price of Anarchy exists when using the payment network reliability as an objective function. The results of our simulation lead to a lower bound for the Price of Anarchy of $\rho = 1.5456$.

We also analyzed to what extent payment flows can be netted. For a payment network to be useful, it not only needs to be reliable, it also needs to be quick. Thus the amount of payments that could be taken into account for netting is limited. Using existing payment networks and their transaction volume, we find that 10,000 payments are equivalent to a timeframe of 1 to 3 seconds of transactions processed by a major credit card company. Thus, we assume that aggregation of 10,000 payments is a reasonable assumption.

Based on this assumption we find that in total we can reduce the liquidity load on the channels by 74%, if we collapse the successful payment flows between two adjacent nodes and aggregate the payments back and forth between these adjacent nodes. This demonstrates that it is possible to further increase payment reliability by netting. We cannot determine the exact reduction in failure rates, but we derive that a substantial improvement can be made, because of less volume being sent which reduces the probability of failure.

How realistic is the assumption to have participants share information with a central router or even have such counterparts relay payments? Based on one of the principal ideas of the Lightning Network of hiding information about payments, at first it seems to not be a likely outcome to have a central router. However, such nodes already exist, for example trampoline nodes. Trampoline nodes offer to take over the responsibility of finding a route from sender to recipient. Even though these nodes won't be aware of all payments in the network, any improvement on the belief of payment balances with a good enough estimate increases network reliability and reduces the Price of Anarchy. This does still come with a loss of privacy, if senders disclose sender, receiver, and payment amount.

List of Figures

List of Tables

List of Listings

Bibliography

[AHWW19] Zeta Avarikioti, Lioba Heimbach, Yuyi Wang, and Roger Wattenhofer. Ride the lightning: The game theory of payment channels, 2019.

[AHWW20] Zeta Avarikioti, Lioba Heimbach, Yuyi Wang, and Roger Wattenhofer. *Ride the Lightning: The Game Theory of Payment Channels*, pages 264–283. Springer International Publishing, Cham, 07 2020.

[AJWW18] Zeta Avarikioti, Gerrit Janssen, Yuyi Wang, and Roger Wattenhofer. Payment network design with fees. *CoRR*, abs/1810.07585, 2018.

[Als23] Sebastian Alscher. Simulation of the lightning network. `https://github.com/sebulino/ThesisSimulation`, February 2023. GitHub repository.

[AOP21] Andreas M. Antonopoulos, Olaoluwa Osuntokun, and René Pickhardt. *Mastering the Lightning Network*. O'Reilly Media, 2021.

[Ban22] European Central Bank. Payments statistics: 2021, Jul 2022.

[Bra68] Dietrich Braess. Über ein paradoxon aus der verkehrsplanung. *Unternehmensforschung*, 12(1):258–268, 1968.

[CVMS20] Simone Cerreia-Vioglio, Fabio Maccheroni, and David Schmeidler. Equilibria of nonatomic anonymous games, 2020.

[Dub86] Pradeep Dubey. Inefficiency of nash equilibria. *Mathematics of Operations Research*, 11(1):1–8, 1986.

[EP19] Ignacio Esponda and Demian Pouzo. Berk-nash equilibrium: A framework for modeling agents with misspecified models, 2019.

[ER60] Pál Erdös and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungary. Acad. Sci.*, 5:17–61, 1960.

[FLM$^+$03] Alex Fabrikant, Ankur Luthra, Elitza Maneva, Christos H. Papadimitriou, and Scott Shenker. On a network creation game. In *Proceedings of the Twenty-Second Annual Symposium on Principles of Distributed Computing*, PODC '03, page 347–351, New York, NY, USA, 2003. Association for Computing Machinery.

[For23] Forbes. 9 interesting credit card statistics, Jan 2023.

[GRT22] Niklas Gögge, Elias Rohrer, and Florian Tschorsch. On the routing convergence delay in the lightning network. *ArXiv*, abs/2205.12737, 2022.

[Har67] John Harsanyi. Games with incomplete information played by "bayesian" players, i-iii part i. the basic model. *Management Science*, 14(3):159–182, 1967.

[HSSC08] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[Jr50] John F Nash Jr. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

[KP99] Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th Annual Conference on Theoretical Aspects of Computer Science*, STACS'99, page 404–413, Berlin, Heidelberg, 1999. Springer-Verlag.

[Kuh19] Steven Kuhn. Prisoner's Dilemma. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2019 edition, 2019.

[Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[NJ96] John Nash Jr. Non-cooperative games. In *Essays on Game Theory*, pages 22–33. Edward Elgar Publishing, 1996.

[Pap01] Christos Papadimitriou. Algorithms, games, and the internet. *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, 2076, 05 2001.

[PD16] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network, 2016. Accessed: 2016-07-07.

[Pic22] René Pickhardt. Price of anarchy from selfish routing strategies on the lightning network, 2022. Accessed: 2022-07-14.

[Pic23] René Pickhardt. Pickhardt payments package. `https://github.com/renepickhardt/pickhardtpayments`, January 2023. GitHub repository.

[PR21] René Pickhardt and Stefan Richter. Optimally reliable & cheap payment flows on the lightning network, 2021.

[PTBN21] Rene Pickhardt, Sergei Tikhomirov, Alex Biryukov, and Mariusz Nowostawski. Security and privacy of lightning network payments with uncertain channel balances. *CoRR*, abs/2103.08576, 2021.

[Rou05] Tim Roughgarden. *Selfish routing and the price of anarchy.* MIT Press, 2005.

[Sch73] David Schmeidler. Equilibrium points of non-atomic games. *Journal of Statistical Physics*, 7:295–300, 04 1973.

[Tea23a] Core Lightning Dev Team. Core lightning (cln): A specification compliant lightning network implementation in c. `https://github.com/ElementsProject/lightning`, February 2023. GitHub repository.

[Tea23b] Lightning Dev Team. Lightning network in-progress specifications. `https://github.com/lightning/bolts/blob/master/04-onion-routing.md`, January 2023. GitHub repository.

[VIS22] VISA. Visa fact sheet, Sep 2022.

[Zmn22a] ZmnSCPxj. 'htlc_maximum_msat' as a valve for flow control on the lightning network. `https://lists.linuxfoundation.org/pipermail/lightning-dev/2022-September/003698.html`, September 2022. Lightning-dev mailing list.

[Zmn22b] ZmnSCPxj. 'htlc_maximum_msat' as a valve for flow control on the lightning network. `https://lists.linuxfoundation.org/pipermail/lightning-dev/2022-September/003697.html`, September 2022. Lightning-dev mailing list.