Boosting Liquidity in Payment Channel Networks with Online Admission Control

Anonymous author
Anonymous affiliation

Abstract

Payment channel networks (PCNs) are a promising technology to improve the scalability of cryptocurrencies. PCNs, however, face the challenge that the frequent usage of certain routes may deplete channels in one direction, and hence prevent further transactions. In order to reap the full potential of PCNs, recharging and rebalancing mechanisms are required to provision channels, as well as an admission control logic to decide which transactions to reject in case capacity is insufficient. This paper presents a formal model of this optimisation problem. In particular, we consider an online algorithms perspective, where transactions arrive over time in an unpredictable manner. Our main contributions are competitive online algorithms which come with provable guarantees over time. We empirically evaluate our algorithms on randomly generated transactions to compare the average performance of our algorithms to our theoretical bounds. We also show how this model and approach differs from related problems in classic communication networks.

2012 ACM Subject Classification Applied computing → Electronic commerce; Theory of computation → Mathematical optimization; Theory of computation → Online algorithms

Keywords and phrases Payment channel networks, Blockchain, Optimisation, Rebalancing, Online Algorithms, Layer 2

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Blockchain consensus protocols are notoriously inefficient: for instance, Bitcoin can only support 7 transactions per second on average which makes it unrealistic to use in everyday situations. Payment channel networks like Bitcoin's Lightning Network [15] and Ethereum's Raiden [1] have been proposed as scalability solutions to blockchains. Instead of sending transactions to the blockchain and waiting for the entire blockchain (which can comprise of millions of users) to achieve consensus, any two users that wish to transact with each other can simply open a payment channel between themselves. Opening a payment channel requires an initial funding transaction on the blockchain where both users lock some funds only to use in the channel. Once a payment channel is opened, the channel acts as a local, two-party ledger: payments between the users of channel simply involve decreasing the balance of the payer by the payment amount, and increasing the balance of the payee correspondingly. As these local transactions only involve exchanging signatures between the two users and do not involve the blockchain at all, they can be almost instantaneous. As long as there is sufficient balance, payments can occur indefinitely between two users, until the users decide to close the channel. This would involve going back to the blockchain and takes, in the worst case, a small constant number of transactions. Thus, with only a small constant number of on-chain transactions, any two users can potentially make arbitrarily many costless transactions between themselves.

Apart from joining a payment channel network to efficiently transact with other users, an additional financial incentive to joining the network is to profit from forwarding transactions. Any two users that are not directly connected can transact with each other in a multi-hop fashion as long as they are connected by a path of payment channels. To incentivise the intermediary nodes on the path to forward the payment, the network typically allows these nodes to charge a transaction fee. Thus, it is common for users to join the network specifically to play the role of an intermediary node that routes transactions, creating channels and fees optimally and selecting the most profitable transactions to maximise their profit from transaction fees [4, 8].

However, greedily accepting and routing incoming transactions could rapidly deplete a user's balance in their channels. In particular, if certain routes are primarily used in one direction, their channels can get depleted, making it impossible to forward further transactions. Accounting for this problem can be non-trivial since demand patterns are hard to predict and often confidential.

To resolve this issue, PCNs typically support two mechanisms:

- *On-chain recharging:* A user can close and reopen a depleted channel with more funds on-chain.
- *Off-chain rebalancing:* An alternative solution is to extend the lifetime of a depleted channel without involving the blockchain, by finding a cycle of payment channels in the network to shift funds from one channel to another.

Both cases, however, entail a cost. Intermediaries need to consider the tradeoff between admitting transactions and potential recharging and rebalancing costs. This decision making process is especially important to big routers which are the primary maintainers of payment channel networks like the Lightning Network.

In this work, we focus on the problem of admission control, recharging and rebalancing in a single payment channel from the perspective of an intermediary node that seeks to route as many transactions as possible with minimal costs. Specifically, we address the following research question:

Can we design efficient online algorithms for deciding when to accept/reject transactions, and when to recharge or rebalance in a single payment channel?

We seek to address this problem with as few restrictions on user actions in order to ensure that our work remains realistic. Thus, we assume a fixed PCN topology with some recharging and rebalancing costs, and a global fee function that is linear in the transaction size. We also assume users incur a rejection cost in the form of opportunity cost when they reject to route a transaction.

We are interested in robust solutions which do not depend on any knowledge or assumptions on the demand. Accordingly, we assume that transactions can arrive in an arbitrary order at a channel, and aim to design online algorithms which provide worst-case guarantees. We are in the realm of competitive analysis, and assume that an adversary with knowledge of our algorithms chooses the most pessimal online transaction sequence. Our objective is to optimise the *competitive ratio* [6]: we compare the performance of our online algorithms (to which the transaction sequence is revealed over time) with the optimal offline algorithm that has access to the entire transaction sequence in advance.

1.1 Our contributions

We initiate the study of a fundamental resource allocation problem in payment channel networks, from an online algorithms perspective. Our main result is a competitive online algorithm to admit transaction streams arriving at both sides of a payment channel, and also to recharge and rebalance the channel, in order to maximise the throughput over the channel while accounting for costs. In particular, our algorithm achieves a competitive ratio of $7 + 2\lceil \log C \rceil$ where $C + 1$ is the length of the rebalancing cycle used to replenish the funds on the channel off-chain. We also provide lower bounds on the amount of funds needed in a channel in order to ensure our algorithm is c-competitive for $c < \frac{\log C}{\log \log C}$.

In order to prove our main theorem, we decompose the problem into two simpler sub problems that may also be of independent interest:

1. *Sub problem 1:* The first and most restrictive sub problem considers a transaction stream coming only from one direction across a payment channel, and users do not have the option to reject incoming transactions. We present a 2-competitive algorithm for this problem, which is optimal in the sense that no deterministic online algorithm can achieve a lower competitive ratio.
2. *Sub problem 2:* As a relaxation, our second sub problem allows users to reject transactions although all transactions are still restricted to come from one direction along a payment channel. We show that our algorithm achieves a competitive ratio of $2 + \frac{\sqrt{5}-1}{2}$ for this sub problem. We stress that our lower bound of 2 we achieve in sub problem 1 also holds in this sub problem, hence our competitive ratio of $2 + \frac{\sqrt{5}-1}{2}$ is close to optimal.

All intermediate and main results are summarised in Table 1. The algorithms and analysis designed to address these sub problems are eventually used as building blocks for our main algorithm and main theorem.

We complement our theoretical worst-case analysis by performing an empirical evaluation of the performance of our algorithm on randomly generated transaction sequences. We observe that our algorithms perform much better on average compared to our theoretical worst-case bound.

Sub problem	Competitive ratio
Unidirectional stream without rejection	2
Unidirectional stream with rejection	$2 + \frac{\sqrt{5}-1}{2}$
Bidirectional stream	$\mathbf{7 + 2\lceil \log C \rceil}$

Table 1 Summary of the theoretical results in our paper. The first column presents each sub problem we analyse in our paper and the second column shows the competitive ratio achieved by our algorithms for each sub problem

1.2 Related work

Maintaining balanced payment channels. As channel balances are typically private, classic transaction routing protocols on payment channel networks like Flare [16], SilentWhispers [13] and SpeedyMurmurs [17] focus mainly on throughput and ignore the issue of balance depletion. Recently, several works shift the focus on maintaining balanced payment channels for as long as possible while ensuring liveness of the network. Revive [10] initiated the study rebalancing strategies, Spider[19] uses multi-path routing to ensure high transaction throughput while maintaining balanced payment channels, the Merchant[20] utilises fee strategies to incentivise the balanced use of payment channels, and [11] uses estimated payment demands along channels to plan the amount of funds to inject into a channel during channel creation, to just give a few examples. Our work focuses on minimising costs incurred in the process of handling transactions across a channel and thus we also indirectly seek to maintain balanced payment channels. Moreover, in contrast to previous works which typically assume some form of offline knowledge of the transaction flow in the network, we provide an algorithm which comes with provable worst-case guarantees.

Off-chain rebalancing. Off-chain rebalancing has been studied as a cheaper alternative to refunding a channel by closing and reopening it on the blockchain. In the Lightning Network, there are already several off-chain rebalancing plugins for c-lightning[1] and lnd[2]. An automated approach to performing off-chain rebalancing using the imbalance measure as a heuristic has been proposed in [14]. Our work similarly studies when to rebalance payment channels, however we make the decision in tandem with other decisions like accepting or rejecting transactions. Recently, [5] and [10] propose a global approach to off-chain rebalancing where demand for rebalancing cycles is aggregated across the entire network and translated to an LP which is subsequently solved to obtain an optimal rebalancing solution. These approaches are orthogonal and complementary to ours as our focus concerns decision making in a single payment channel and not the entire network.

Online algorithms for payment channel networks. Online algorithms for payment channel networks have also been studied in [3] and [9]. Avarikioti et al. [3] establish impossibility results against certain classes of adversaries, however they only consider a limited problem setting where their algorithms can only accept or reject transactions (with constant rejection cost). Fazli et al. [9] considers the problem of optimally scheduling on-chain recharging given a sequence of transactions. In contrast to previous work, our work considers a more general problem setting where our algorithms can not only accept or reject transactions, but also recharge and rebalance channels off chain. We also extend the cost of rejection to take into account the size of the transaction.

Relationship to classic communication networks. Admission control problems such as online call admission [2, 12] are fundamental and have also received much attention in the context of communication networks. However, in classic communication networks the available capacity of a link in one direction is independent of the flows travelling in the other direction, and moreover, link capacities are only consumed by the currently allocated flows. In contrast, the capacities of links in payment-channel networks are permanently reduced by transactions flowing in one direction, but can be topped up by flows travelling in the other direction. The resulting rebalancing opportunity renders the underlying algorithmic problem significantly different.

2 Model

Payment channels. We model the payment channel network as an undirected graph $G = (V, E)$. A payment channel between users ℓ (left) and r (right) in the network is an edge $(\ell, r) \in E$. We denote the balance of user ℓ (resp. r) in the channel (ℓ, r) by $b(\ell)$ (resp. $b(r)$). The *capacity* of the channel is the total amount of funds locked in the channel. That is, for a channel (ℓ, r), the capacity of (ℓ, r) is $b(\ell) + b(r)$. A left-to-right transaction of amount x decreases ℓ's balance by x and increases r's balance by x and vice versa for a right-to-left transaction of x.

Recharging and rebalancing payment channels. When a user in a channel does not have sufficient funds to accept a transaction, the user can either reject the transaction, recharge the channel, or rebalance the channel. Recharging the channel happens on-chain and corresponds

[1] https://github.com/lightningd/plugins/tree/master/rebalance
[2] https://github.com/bitromortac/lndmanage

to closing the payment channel on the blockchain and opening a new channel with more funds. In contrast, rebalancing the channel happens entirely off-chain (refer to Figure 1 for an example). Here, users find a cycle of payment channels to shift funds from one of their other channels to refund the depleted channel.

Transactions. We consider a transaction sequence $X_t = (x_1, ..., x_t)$, $x_i \in \mathbb{R}^+$, that arrives at a payment channel online. Each transaction x_i has both a value and a direction along a payment channel. The value of a transaction is simply the amount that is being transferred. The direction of a transaction across a payment channel (ℓ, r) determines who is the sender and who is the receiver. When we have a sequence of transactions that go in both directions along a payment channel, we use $\overrightarrow{x}$ to denote a transaction that goes from left-to-right and $\overleftarrow{x}$ to denote a transaction that goes from right-to-left. We say a user, wlog ℓ, *accepts* a transaction of size x coming from the left to right direction along the channel (ℓ, r) if ℓ agrees to forward x to r. Similarly, we say a user ℓ *rejects* a transaction x coming from the left to right direction along the channel (ℓ, r) when ℓ does not forward the transaction to r. When it is clear which channel and direction we are referring to, we simply say ℓ accepts or rejects x.

Costs. We consider three types of costs in our problem setting:

1. **Rejecting transactions:** For a user ℓ, the revenue in terms of transaction fees from forwarding a payment of size x is $Rx + f_2$, where $R, f_2 \in \mathbb{R}^+$. Consequently, the cost of rejecting a transaction of size x is simply the opportunity cost of gaining revenue from accepting the transaction, i.e. $Rx + f_2$.
2. **On-chain recharging:** For any user ℓ, the cost of recharging a channel on-chain is $F + f_1$, where F is some function of the amount of funds ℓ puts into the new channel (this captures the opportunity cost of locking in the funds in the channel) and $f_1 \in \mathbb{R}^+$ is an auxiliary cost independent of F which captures the on-chain recharging transaction fee.
3. **Off-chain rebalancing:** For any user ℓ, the cost of off-chain rebalancing for an amount x is $C \cdot (Rx + f_2)$, where C is the length of the cycle along which funds are sent -1. In the example of off-chain rebalancing in Figure 1, the length of the rebalancing cycle is 3 and thus $C = 2$.

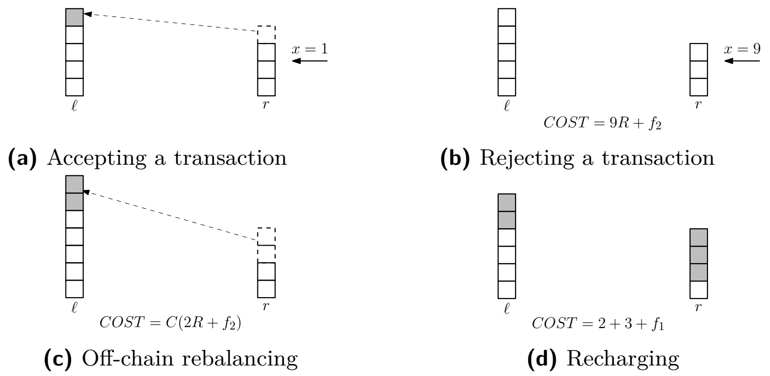
Let us denote by OFF the optimal offline algorithm and ON an online deterministic algorithm. We denote by $\textsc{Cost}_{\text{ON}}(X_t)$ (resp. $\textsc{Cost}_{\text{OFF}}(X_t)$) the total cost of ON (resp. OFF) given the transaction sequence X_t.

Figure 1 Example of off-chain rebalancing with users ℓ, h, and r. The graph on the right depicts the channel balances after off-chain rebalancing.

Competitive ratio. We say an online algorithm ON is *c-competitive* if for every transaction sequence X_t generated by the adversary,

$$\textsc{Cost}_{\text{ON}}(X_t) \leq c \cdot \textsc{Cost}_{\text{OFF}}(X_t)$$

Main problem. Our main problem is to design a competitive deterministic online algorithm that determines when to accept/reject transactions and when to recharge or rebalance the channel given a bidirectional stream of transactions across a payment channel. More precisely, we consider a stream of transactions that can arrive from both right to left or left to right in



(a) Accepting a transaction **(b)** Rejecting a transaction

(c) Off-chain rebalancing **(d)** Recharging

Figure 2 Example of actions users ℓ and r can take in the general bidirectional stream setting. Each square represents 1 coin.

a given payment channel (ℓ, r). ℓ (resp. r) can choose to accept or reject transactions coming in the left-to-right (resp. right-to-left) direction in the stream. Either user would incur a cost of $Rx + f_2$ for rejecting a transaction of size x. Both users can also recharge the channel on-chain at any point, incurring a cost of $F_\ell + F_r + f_1$ where F_ℓ and F_r are functions of the funds put into the channel by ℓ and r respectively. Since transactions are streaming in both directions in this model, both users would incur costs in this setting. Thus, we seek to design an algorithm that minimises the cost of the *entire* channel. Refer to Figure 2 for examples of the actions that a user can take in our main bidirectional transaction stream setting.

To this end, we give a formal definition of two sub problems of decreasing restrictiveness on user actions. We present these sub problems as the algorithms and analysis used to solve these sub problems are used in developing the algorithm and analysis for our main problem.

Unidirectional stream without rejection. In this model, transactions stream only in one direction along a given payment channel. Here, we assume users cannot reject incoming transactions. Formally, given a channel (ℓ, r) and a transaction stream from wlog left to right, user ℓ only accepts a transaction x if $b(\ell) > x$. Otherwise, ℓ has to recharge the channel on-chain with more funds, incurring a cost of $F + f_1$ where F is some function of the amount of funds ℓ adds to the channel. As we only consider transactions streaming in one direction, only one user would incur costs in this setting (the user that has to decide whether to accept or reject transactions).

Unidirectional stream with rejection. In this model, we still restrict the transaction stream from wlog left to right in a payment channel (ℓ, r). However, in addition to accepting transactions and recharging, ℓ can now also reject transactions, incurring a rejection cost of $Rx + f_2$ for a transaction of size x.

3 Algorithmic Building Blocks

Before we describe and analyse the performance of our algorithms in the various problem settings, we first introduce two algorithmic building blocks that we use extensively in our work. The first building block is an algorithm FUNDS. It takes a sequence of transactions as an input and returns the amount of funds that an optimal algorithm uses on this sequence. The purpose of the algorithm is to track the funds OFF has in their channel assuming that the sequence of transactions ends at this point. For the first two sub problems we show how to compute FUNDS. For the main problem, we propose a dynamic programming approach in

Algorithm 1 (γ, δ)-recharging

Initialise: $F_{tracker}, X \leftarrow 0, \emptyset$
for *transaction x in order of arrival* **do**
  concatenate x to X
  $F'_{tracker} \leftarrow \textsc{Funds}(X)$
  if $F'_{tracker} > F_{tracker}$ **then**
    $F_{tracker} \leftarrow F'_{tracker} + \delta$
    recharge to $\gamma F_{tracker}$

Appendix E. The second building block is a general recharging online algorithm that calls FUNDS as a subroutine and uses the output to decide when and how much to recharge the channel. The intuition behind the recharging online algorithm is to recharge whenever the amount of funds in OFF's channel "catches up" to the amount of funds ON has in their channel.

Building block 1: tracking funds of OFF. For a given transaction sequence $X_t = (x_1, \ldots, x_t)$, let us denote $A(X_i)$ to be the amount of funds OFF would use in the channel if OFF gets the sequence $X_i = (x_1, \ldots, x_i)$ (i.e. the length i prefix of X_t) as input. By appending subsequent transactions $x_{i+1}, \ldots, x_t$ from X_t to X_i, we can view $A(X_i)$ as a partial solution to the online optimisation problem that gets updated with any new transaction. In the unidirectional transaction stream (with or without rejection) setting, $A(X_i)$ refers to the funds a user locks into a payment channel. In the bidirectional transaction stream setting, $A(X_i)$ refers to the total balance of both users in the channel. We assume that given an input sequence X_t, $\textsc{Funds}(X_t)$ performs the necessary computations and returns $A(X_t)$. For our main problem, computing $\textsc{Funds}(X_t)$ is generally NP-hard, but we can approximate it to a constant factor, see [18] for more details.

Building block 2: using tracking for recharging. In Algorithm 1, we describe an online (γ, δ)-recharging algorithm ON that uses FUNDS as a subroutine to decide when and how much to recharge the channel. ON is run by one user (wlog ℓ) in a payment channel (ℓ, r). ON calls FUNDS after each transaction to check if the new transaction sequence results in a significant increase in the amount of funds OFF has in their channel. Whenever ON notices that OFF's funds have increased above a threshold (Algorithm 1), ON recharges the channel with an amount of $\gamma(A(X_i) + \delta)$ where $A(X_i)$ is the amount of funds OFF has in their channel.

Let us denote $A_t := \max_{i \leq t} A(X_i)$. Now we state and prove (in Appendix C.1 and Appendix C.2) two important properties of the (γ, δ)-recharging algorithm.

▶ **Lemma 1.** *Algorithm 1 with parameters (γ, δ) ensures that ON always has at least γ times the amount of funds OFF has and ensures that ON incurs a cost of at most $\gamma(A_t + \delta) + f_1 \cdot \lceil \frac{A_t}{\delta} \rceil$.*

Next, we show a simple lower bound in terms of A_t for the cost of OFF given a sequence of transactions X_t.

▶ **Lemma 2.** *If $A_t > 0$, then $\textsc{Cost}_{OFF}(X_t)$ is at least $A_t + f_1$.*

4 Unidirectional transaction stream without rejection

In this section we consider the first sub problem where, given a payment channel (ℓ, r), transactions stream along the channel in only one direction (wlog left to right). Moreover, ℓ

has to accept an incoming transaction of size x and forward it to r if ℓ's balance $b(\ell) \geq x$. Otherwise, ℓ needs to recharge the channel on-chain (and accept the transaction after).

The optimal offline algorithm OFF follows a simple strategy: since it knows the entire stream of transactions in advance, it makes a single recharging action at the beginning of the transaction sequence X_t of size $\sum_{i=1}^{t} x_i$. The cost incurred by OFF is thus $f_1 + \sum_{i=1}^{t} x_i$.

Now, we present a 2-competitive online algorithm ON for this sub problem (see Algorithm 5 in Appendix A). ON uses (γ, δ)-recharging with parameters $\gamma = 1$ and $\delta = f_1$. The algorithm accepts all transactions and the recharging ensures that ON always has enough funds.

▶ **Theorem 3.** *The algorithm described above is* 2*-competitive in the unidirectional transaction stream without rejection.*

In addition, we note that ON is optimal in this setting. The next theorem (with proof in Appendix C.4) proves that no deterministic algorithm can achieve a strictly smaller competitive ratio compared to ON. In particular, our proof shows that ON cannot lock too much funds into the channel, otherwise ON's cost is too high, but if ON locks too little funds, it needs to recharge often.

▶ **Theorem 4.** *There is no deterministic algorithm that is c-competitive for $c < 2$ in the unidirectional transaction stream without rejection sub problem.*

5 Unidirectional transaction stream with rejection

In this section we consider the second sub problem where transactions are still streaming along a given payment channel (ℓ, r) in one direction (wlog left to right). This time though, a user can choose to reject incoming transactions. We describe an algorithm (detailed in Appendix A as Algorithm 6) with competitive ratio $2 + \frac{\sqrt{5}-1}{2}$. We note that the competitive ratio for this setting is larger than the competitive ratio we achieve in the previous setting as OFF has a wider range of decisions.

Let us call a transaction of size x *big* if $x > Rx + f_2$ and *small* otherwise. We first observe that OFF in this setting always rejects big transactions.

▶ **Lemma 5.** *OFF rejects all big transactions in the unidirectional transaction stream with rejection.*

Thus, the strategy of OFF in this setting is to simply reject all big transactions. Moreover, if there are sufficiently many small transactions in the sequence to offset the cost of recharging, OFF makes a single recharging action at the beginning of the sequence of size $\sum_{x \in X_t, x \text{ is small}} x$ for a cost of $f_1 + \sum_{x \in X_t, x \text{ is small}} x$.

The online algorithm performs $(1, \frac{\sqrt{5}-1}{2} f_1)$-recharging and it accepts a transaction x if it has enough funds and x is small. The following theorem (with proof in Appendix C.6) states that ON is $(2 + \frac{\sqrt{5}-1}{2})$-competitive in this problem setting.

▶ **Theorem 6.** *The algorithm described above is $(2 + \frac{\sqrt{5}-1}{2})$-competitive in the unidirectional transaction stream with rejection sub problem.*

Before analysing the optimality of ON, we first observe, as a simple corollary of Theorem 4, that the lower bound of 2 also holds for this sub problem.

▶ **Corollary 7.** *There is no deterministic algorithm that is c-competitive for $c < 2$ in the unidirectional transaction stream with rejection sub problem.*

We conjecture that no other deterministic algorithm can perform better that ON in this setting. Moreover, we sketch an approach to prove the conjecture in Appendix B.

▶ **Conjecture 8.** *There is no deterministic algorithm that is c-competitive for* $c < 2 + \frac{\sqrt{5}-1}{2}$ *in the unidirectional transaction stream with rejection setting.*

6 Bidirectional transaction stream

In this section, we consider the most general problem setting, where for a given payment channel (ℓ, r), transactions stream along the channel (ℓ, r) in both directions. A user ℓ (resp. r) can accept or reject incoming transactions that stream from left to right (resp. right to left). Either user would incur a cost of $Rx + f_2$ for rejecting a transaction of size x. ℓ does not need to take any action when encountering transactions that stream from right to left as they simply increase the balance of ℓ in the channel (ℓ, r). Both users can also decide at any point to recharge their channel on-chain, or rebalance their channel off-chain.

Our main online algorithm ON for the bidirectional transaction stream setting is detailed in Algorithm 4. For simplicity, we assume that $R = 0$ in the rejection cost. This means that the cost of rejecting a single transaction of size x is simply f_2, and rebalancing an amount of x off-chain now only incurs a cost of Cf_2. Our algorithm is run by both users on a payment channel and is composed of three smaller algorithms: the first is a recharging algorithm to determine when and how much to recharge the channel on-chain. The second algorithm (Algorithm 2) decides whether to accept or reject new transactions and when to perform off-chain rebalancing. The last algorithm (Algorithm 3) describes how to store the funds received from the other user of the channel.

$(4 + 2\lceil\log C\rceil, f_1)$*-recharging.* ON runs an on-chain recharging algorithm similar to Algorithm 1 (see Algorithm 4 and Algorithm 4 in Algorithm 4) but with parameters $\gamma = 4 + 2\lceil\log C\rceil$ and $\delta = f_1$. Since we are in the bidirectional transaction stream setting, FUNDS returns the amount of funds OFF has inside the entire channel (i.e. $b(\ell) + b(r)$) given a transaction sequence.

Let us look at the period between the on-chain recharging instances of ON. From Algorithm 4 in Algorithm 4, we know that ON ensures that it has more than $4 + 2\lceil\log C\rceil$ times more funds than OFF locked in the channel. These funds are distributed in the following way: ON initialises $\lceil\log C\rceil + 2$ "buckets" on each end of the channel. We denote set of left-side buckets as B^ℓ and it consists of $B^\ell_s, B^\ell_1, \ldots, B^\ell_{\lceil\log C\rceil}, B^\ell_o$. Likewise, the set of right-side buckets is B^r and it consists of $B^r_s, B^r_1, \ldots, B^r_{\lceil\log C\rceil}, B^r_o$.

After recharging, users decide how to distribute funds in the channel, so the buckets B^ℓ_s and B^r_s are filled with $2F_{tracker}$ funds. Buckets B^ℓ_o and B^r_o are empty (0 funds). Other buckets contain $F_{tracker}$ funds.

Looking ahead, the funds in the i-th bucket on both sides are used to accept transactions x with a size in the interval $\left[\frac{F_{tracker}}{2^i}, \frac{F_{tracker}}{2^{i-1}}\right)$. The funds in B_s are used to accept transactions with a size less than $\frac{F_{tracker}}{C}$. Finally, B_o stores excess funds coming from payments from the other side when all other buckets are full.

Transaction handling. When a transaction arrives at the channel, based on the direction of the transaction, either ℓ or r executes Algorithm 2 to decide whether to accept the transaction. Wlog let us assume ℓ encounters transaction $\vec{x}$. If $\frac{F_{tracker}}{2^i} < x \leq \frac{F_{tracker}}{2^{i-1}}$ for some $i \in [\lceil\log C\rceil]$ and B^ℓ_i has sufficient funds, the funds from B^ℓ_i are used to accept the transaction. If B^ℓ_i lacks sufficient funds for accepting x, ℓ rejects x.

Now, we consider the case where $x \leq \frac{F_{tracker}}{C}$. If B^ℓ_s has sufficient funds, ℓ uses the funds from B^ℓ_s to accept x. If B^ℓ_s has insufficient funds to accept x, ℓ performs off-chain rebalancing with an amount such that after deducting x from B_s^ℓ, there would still be $2F_{tracker}$ funds left in B_s^ℓ. ℓ subsequently accepts x. The required funds for off-chain rebalancing are transferred from B_o^r and B_s^r (see Algorithm 2 and Algorithm 2 in Algorithm 2). Whenever $B_o^\ell > 0$ and some bucket in B^ℓ gets under its original capacity, funds are reallocated from B_o^ℓ to fill the bucket. Figure 3 in Appendix D depicts an example of how funds are used from different buckets to accept transactions.

Algorithm 2 Decision on transaction

$\textsc{Decide}(F_{tracker}, x, B^{sdr}, B^{rcv})$
- $Status \leftarrow$ `Accept`
- **if** $\frac{F_{tracker}}{2^i} < x \leq \frac{F_{tracker}}{2^{i-1}}$ *and* $x \leq B_i^{sdr}$ **then**
  - Accept x
  - $X \leftarrow \min(F_{tracker}, B_i^{sdr} - x + B_o^{sdr})$
  - $B_o^{sdr} \leftarrow \max(0, B_i^{sdr} - x + B_o^{sdr} - F_{tracker})$
  - $B_i^{sdr} \leftarrow X$
- **else if** $x_i \leq \frac{F_{tracker}}{C}$ *and* $x \leq B_s^{sdr}$ **then**
  - Accept x
  - $X \leftarrow \min(2F_{tracker}, B_s^{sdr} - x + B_o^{sdr})$
  - $B_o^{sdr} \leftarrow \max(0, B_s^{sdr} - x + B_o^{sdr} - 2F_{tracker})$
  - $B_s^{sdr} \leftarrow X$
- **else if** $x_i \leq \frac{F_{tracker}}{C}$ *and* $x > B_s^{sdr}$ **then**
  - Do off-chain rebalancing to fill B_s and pay f_2C.
  - $B_o^{rcv} \leftarrow B_o^{rcv} - (2F_{tracker} - B_s^{sdr})$.
  - $B_s^{rcv} \leftarrow B_s^{rcv} - x$.
  - Accept x
  - $B_s^{sdr} \leftarrow 2F_{tracker}$.
- **else**
  - Reject x
  - $Status \leftarrow$ `Reject`
- **return** $(B^{sdr}, B^{rcv}, Status)$

Handling funds coming from the other side. When a transaction x is accepted by wlog ℓ, ON calls Algorithm 3 to distribute the transferred funds among r's buckets in the following way: r first uses x to fill B_s^r up to its capacity of $2F_{tracker}$ (see Algorithm 3 in Algorithm 3). If there are still funds left, r refills the B_i^r buckets in descending order from $i = \lceil \log C \rceil$ to $i = 1$. Intuitively, the reason why buckets are refilled in descending order is due to our simplified cost model for this problem where we assume the cost of rejection for any transaction is f_2. Thus, rejecting three small transactions size x costs thrice as much as rejecting a larger transaction of size $3x$. Finally, if there are still some funds left, they are added to B_o^r.

Our main theorem (with proof in Appendix C.7) shows that our main algorithm is $7 + 2\lceil \log C \rceil$ competitive.

▶ **Theorem 9.** *Algorithm 4 is* $7 + 2\lceil \log C \rceil$ *competitive.*

Finally, we also analyse in the next lemma (with proof in Appendix C.8) how much funds ON needs to lock in the channel to have a chance to be c-competitive. We make the construction for A, the amount of funds that OFF locked in the channel. Observe that

Algorithm 3 Handling funds coming from the other side

HandleFunds($F_{tracker}, x, B$)
  $X \leftarrow \min(2F_{tracker}, B_s + x)$
  $x \leftarrow \max(x + B_s - 2F_{tracker}, 0)$
  $B_s \leftarrow X$
  for $i \in [\lceil \log C \rceil]$ *in decreasing order* **do**
    if $x > 0$ **then**
      $X \leftarrow \min(F_{tracker}, B_i + x)$
      $x \leftarrow \max(x + B_i - F_{tracker}, 0)$
      $B_i \leftarrow X$
  $B_o \leftarrow B_o + x$ **return** (B)

Algorithm 4 Main algorithm

Initialise: left side buckets B^ℓ
Initialise: right side buckets B^r
Initialise: tracker $F_{tracker}, X \leftarrow 0, \emptyset$
for *transaction* x *in order of arrival* **do**
  concatenate x to X
  $F'_{tracker} \leftarrow$ Funds(X)
  if $F'_{tracker} > F_{tracker}$ **then**
    $F_{tracker} \leftarrow F'_{tracker} + f_1$
    recharge to $2(2 + \lceil \log C \rceil)F_{tracker}$
  $sdr, rcv \leftarrow \ell, r$
  if x *is from right to left* **then**
    $sdr, rcv \leftarrow r, \ell$
  $B^{sdr}, B^{rcv}, Status \leftarrow$ Decide($F_{tracker}, x, B^{sdr}, B^{rcv}$)
  if $Status ==$ `Accept` **then**
    $B^{rcv} \leftarrow$ HandleFunds($F_{tracker}, x, B^{rcv}$)

OFF would rather reject transactions that have average size $> \frac{A}{C}$ than perform off-chain rebalancing to accept them.

▶ **Lemma 10.** *For any A, if ON's cost for rejection is at most c times OFF's cost for rejection (for $c < \frac{\log C}{\log \log C}$), any deterministic ON needs to lock at least $\sigma = A \cdot \left(\frac{\frac{1}{c+1} \log C}{\log c+1} + 1 \right)$ funds in the channel.*

▶ **Theorem 11.** *There is no deterministic c-competitive algorithm for $c \in o(\sqrt{\log C})$.*

Proof. From Lemma 10 for any A, ON needs $A \cdot (\frac{\frac{1}{c+1} \log C}{\log c+1} + 1)$ funds to have its rejection cost c-competitive. But ON also needs to lock some funds in the channel. The total cost is then $c + A(\frac{\frac{1}{c+1} \log C}{\log c+1} + 1)$, which is bigger than $\mathcal{O}(\sqrt{\log C})$. ◀

7 Empirical Evaluation

Methodology. We consider the performance of Algorithm 4 on randomly generated transaction sequences. We compare it with the optimal offline algorithm OFF. Since computing the optimal solution is NP-hard [18], we use dynamic programming to compute the cost (see the algorithm in Appendix E).

Average performance of ON. We sample 50 random transaction sequences of length 50 each. In each sequence, transaction sizes are sampled independently from a folded Gaussian with mean 0 and standard deviation 3, and then we assign its direction (left-to-right or right-to-left) uniformly at random. Finally, we quantise the size of the transaction to the closest integer. We run both OFF and ON on the generated sequences and compute five important metrics.

We present our results in Table 2. As we can see from the cost of ON vs OFF in Table 2, the competitive ratio is generally lower than the $7 + 2\lceil \log C \rceil$ bound as suggested by our conservative worst-case analysis in Theorem 9. In addition, we notice that when we use some heuristics to make further minor modifications to ON, we achieve even better performance. We compare the average-case performance of ON and OFF with these modified algorithms in Appendix F, and also on sequences sampled from different distributions. Finally, we also perform an empirical case study of the Lightning Network which we present in Appendix G.

Param		OFF					ON				
C	f_2	Cost	$A(X)$	Accept rate	Off-chain re-balancing	Rechar-gings	Cost	$A(X)$	Accept rate	Off-chain re-balancing	Rechar-gings
2	0.5	**15.02**	6.4	0.78	0.8	1	**63.3**	44.26	0.50	0.9	2.18
8	0.5	**15.21**	6.38	0.77	0	1	**87.79**	69.06	0.50	0	2.04
2	2	**23.6**	14.26	0.95	5.36	1	**127.02**	100.2	0.91	0.38	5.86
8	2	**24.5**	13.9	0.92	0	1	**184.32**	156.6	0.9	0	5.84

Table 2 Comparison between OFF and ON for $f_1 = 3$ and $R = 0$. $A(X)$ is the total amount of funds in the channel. “Accept rate" shows the fraction of transactions that were accepted. “Off-chain rebalancing" shows how much funds on was moved along the channel using off-chain rebalancing. “Rechargings" shows the number of rechargings performed. Note that OFF recharges only once.

8 Conclusion

This paper presents competitive strategies to maintain minimise cost while maximising liquidity and transaction throughput in a payment channel. Our algorithms come with formal worst-case guarantees, and also perform well in realistic scenarios in simulations.

We believe that our work opens several interesting avenues for future research. On the theoretical front, it would be interesting to close the gap in the achievable competitive ratio, and to explore the implications of our approach on other classic online admission control problems. Furthermore, while in our work we have focused on deterministic algorithms, it would be interesting to study the power of randomised approaches in this context, or to consider different adversarial models.

References

1 Raiden network. `https://raiden.network/`, 2017.

2 James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM (JACM)*, 44(3):486–504, 1997.

3 Georgia Avarikioti, Kenan Besic, Yuyi Wang, and Roger Wattenhofer. Online payment network design. *CoRR*, abs/1908.00432, 2019. URL: `http://arxiv.org/abs/1908.00432`, `arXiv:1908.00432`.

4 Zeta Avarikioti, Lioba Heimbach, Yuyi Wang, and Roger Wattenhofer. Ride the lightning: The game theory of payment channels. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, volume 12059 of *Lecture Notes in Computer Science*, pages 264–283. Springer, 2020. doi:10.1007/978-3-030-51280-4_15.

5 Zeta Avarikioti, Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, Samarth Tiwari, and Michelle Yeo. HIDE & SEEK: privacy-preserving rebalancing on payment channel networks. *CoRR*, abs/2110.08848, 2021. URL: https://arxiv.org/abs/2110.08848, arXiv:2110.08848.

6 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. cambridge university press, 2005.

7 Christian Decker. Lightning network research; topology, datasets. https://github.com/lnresearch/topology. Accessed: 2022-04-01. doi:10.5281/zenodo.4088530.

8 Oguzhan Ersoy, Stefanie Roos, and Zekeriya Erkin. How to profit from payments channels, 2019. arXiv:1911.08803.

9 MohammadAmin Fazli, Seyed Moeen Nehzati, and MohammadAmin Salarkia. Building stable off-chain payment networks. *CoRR*, abs/2107.03367, 2021. URL: https://arxiv.org/abs/2107.03367, arXiv:2107.03367.

10 Rami Khalil and Arthur Gervais. Revive: Rebalancing off-blockchain payment networks. *IACR Cryptol. ePrint Arch.*, page 823, 2017. URL: http://eprint.iacr.org/2017/823.

11 Peng Li, Toshiaki Miyazaki, and Wanlei Zhou. Secure balance planning of off-blockchain payment channel networks. In *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*, pages 1728–1737. IEEE, 2020. doi:10.1109/INFOCOM41043.2020.9155375.

12 Tamás Lukovszki and Stefan Schmid. Online admission control and embedding of service chains. In *International Colloquium on Structural Information and Communication Complexity*, pages 104–118. Springer, 2015.

13 Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Silentwhispers: Enforcing security and privacy in decentralized credit networks. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017.

14 Rene Pickhardt and Mariusz Nowostawski. Imbalance measure and proactive channel rebalancing algorithm for the lightning network. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2020, Toronto, ON, Canada, May 2-6, 2020*, pages 1–5. IEEE, 2020. doi:10.1109/ICBC48266.2020.9169456.

15 Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf, 2015.

16 Pavel Prihodko, S. N. Zhigulin, Mykola Sahno, A B Ostrovskiy, and Olaoluwa Osuntokun. Flare : An approach to routing in lightning network white paper. 2016.

17 Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

18 Stefan Schmid, Jakub Svoboda, and Michelle Yeo. Weighted packet selection for rechargeable links: Complexity and approximation, 2022. URL: https://arxiv.org/abs/2204.13459, doi:10.48550/ARXIV.2204.13459.

19 Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathleen Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Giulia C. Fanti, and Mohammad Alizadeh. High throughput cryptocurrency routing in payment channel networks. In Ranjita Bhagwan and George Porter, editors, *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, pages 777–796. USENIX Association, 2020.

Algorithm 5 Unidirectional transaction stream without rejection

Initialise: tracker $F_{tracker}, X \leftarrow 0, \emptyset$
Initialise: balance $b = 0$
for *transaction* x *in order of arrival* **do**
  concatenate x to X
  $F'_{tracker} \leftarrow$ FUNDS(X)
  if $F'_{tracker} > F_{tracker}$ **then**
    $F_{tracker} \leftarrow F'_{tracker} + f_1$
    recharge to $F_{tracker}$
  Accept x

Algorithm 6 Unidirectional transaction stream with rejection

Initialise: tracker $F_{tracker}, X \leftarrow 0, \emptyset$
Initialise: balance $b = 0$
for *transaction* x *in order of arrival* **do**
  concatenate x to X
  $F'_{tracker} \leftarrow$ FUNDS(X)
  if $F'_{tracker} > F_{tracker}$ **then**
    $F_{tracker} \leftarrow F'_{tracker} + \frac{\sqrt{5}-1}{2} f_1$
    recharge to $F_{tracker}$
  if $b \geq x$ *and* x *is small* **then**
    Accept x
  else
    Reject x

20 Yuup van Engelshoven and Stefanie Roos. The merchant: Avoiding payment channel depletion through incentives. *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 59–68, 2021.

A Algorithms

B Optimality of deterministic algorithms in a unidirectional stream with rejection

Sketch proof of Conjecture 8.

Sketch. The best ON algorithm needs to recharge the channel when OFF does. If it recharges the channel later, it incurs cost that OFF is not incurring, so the competitive ratio worsens. If it recharges sooner, there exists a sequence that either forces OFF to waste funds or incur a big cost.

After OFF recharges, it can reconsider and accept previously rejected transactions, but ON needs to reject them. Now, the situation is similar as in the case without rejection. ON needs to recharge, but it already paid for some rejections whereas OFF pays only for recharging and accepting very small transactions.

ON disadvantaged in this way cannot achieve a better competitive ratio than $2+\frac{\sqrt{5}-1}{2}$ ◀

C Omitted proofs

C.1 Proof of Lemma 1

Proof. The first part of the claim follows from the fact that the moment $A(X_i) > F_{tracker}$ for some i, $F_{tracker}$ gets updated to $A(X_i) + \delta > A(X_i)$ and ON recharges the channel to $\gamma F_{tracker} > \gamma A(X_i)$.

For the second part of the claim, we note that the cost incurred by ON is simply the total amount of funds added to the channel with an additional cost of f_1 each time ON recharges the channel on-chain. The amount of funds locked in the channel for ON is always at most $A_t + \delta$ and the times when ON recharges the channel occurs whenever OFF increases its funds by an amount of at least δ. Thus, the number of rechargings for ON that can occur is at most $\lceil \frac{A_t}{\delta} \rceil$ with a cost of f_1 for each recharging instance. The total cost incurred by ON is therefore $\gamma(A_t + \delta) + f_1 \cdot \lceil \frac{A_t}{\delta} \rceil$. ◀

C.2 Proof of Lemma 2

Proof. We first note that the sequence of costs for OFF is monotonically increasing, i.e. $\textsc{Cost}_{\mathrm{OFF}}(X_t) \leq \textsc{Cost}_{\mathrm{OFF}}(X_{t+1})$. This comes from the fact that any action of OFF at step i of the sequence can only increase its cost (i.e. either rejecting x_{i+1} or recharging the channel and then accepting x_{i+1}), or it does not change the cost at all (i.e. by accepting x_{i+1} without recharging).

Since $A_t > 0$, we know that OFF recharged on-chain at some point to an amount A_t for a recharging cost of $A_t + f_1$. Since the sequence of costs for OFF is monotonically increasing, $\textsc{Cost}_{\mathrm{OFF}}(X_t) \geq A_t + f_1$. ◀

C.3 Proof of Theorem 3

Proof. From Lemma 1, setting $\gamma = 1$ and $\delta = f_1$ gives ON a cost of at most $A_t + f_1 + f_1 \cdot \lceil \frac{A_t}{f_1} \rceil$. Since $f_1 \cdot \lceil \frac{A_t}{f_1} \rceil \leq f_1 \cdot (\frac{A_t}{f_1} + 1) = A_t + f_1$, the cost of ON is at most $2(A_t + f_1)$. From Lemma 2, we know that the cost of OFF is at least $A_t + f_1$. Thus, ON is 2-competitive. ◀

C.4 Proof of Theorem 4

Proof. We prove the theorem by contradiction. For the sake of contradiction, suppose that there exists a c-competitive algorithm ON for $c = 2 - \varepsilon$ for some $\varepsilon > 0$. Consider the following sequence of transactions: $\frac{\varepsilon}{3}, \frac{\varepsilon}{3}, \frac{\varepsilon}{3}, \ldots$. We note that when the sequence of transactions is of length k, the cost of OFF is $f_1 + k \cdot \frac{\varepsilon}{3}$ as the optimal solution is to recharge the channel at the start of the sequence to the total sum of the transactions in the sequence.

For ON to remain $(2 - \varepsilon)$-competitive after processing the first transaction, ON locked at most $f_1 - \varepsilon f_1 + \frac{\varepsilon}{3}$ in the channel ($\textsc{Cost}_{\mathrm{ON}}(X_1) = 2f_1 - \varepsilon f_1 + 2\frac{\varepsilon}{3}$).

We generalize the above idea and show that either ON has always smaller amount than $f_1 - \frac{\varepsilon}{3}$ in the channel or at some point it has at least $f_1 - \frac{\varepsilon}{3}$. In both cases, we derive a contradiction to the $(2 - \varepsilon)$ competitive ratio of ON.

First, suppose that ON always recharges to at most $f_1 - \varepsilon'$ for some $\varepsilon' > 0$. Then after t transactions, the number of rechargings is at least $\lceil \frac{t\frac{\varepsilon}{3}}{f_1 - \varepsilon'} \rceil$. So $\textsc{Cost}_{\mathrm{ON}}(X_t) \geq t \cdot \frac{\varepsilon}{3} + f_1 \lceil \frac{t\frac{\varepsilon}{3}}{f_1 - \varepsilon'} \rceil$. Setting $t = \frac{3k(f_1 - \varepsilon')}{\varepsilon}$ for some k gives $\textsc{Cost}_{\mathrm{OFF}}(X_t) = f_1 + k(f_1 - \varepsilon')$ and $\textsc{Cost}_{\mathrm{ON}}(X_t) = k(f_1 - \varepsilon') + kf_1$, but since $\lim_{k \to \infty} \frac{2kf_1 - k\varepsilon'}{(k+1)f_1 - k\varepsilon'} = \frac{2f_1 - \varepsilon'}{f_1 - \varepsilon'}$ for any ε', then the competitive ratio is at least 2.

Now, suppose that t is the first time that after processing a transaction, ON has at least $f_1 - \frac{\varepsilon}{3}$ locked in the channel. At time t, $\textsc{Cost}_{\text{OFF}}(X_t) = f_1 + t\frac{\varepsilon}{3}$. Cost of ON is $f_1 + t\frac{\varepsilon}{3}$ for funds locked in the channel plus any additional recharging cost. But since it is the first time ON recharged by more than f_1, the cost for recharging is $f_1\lceil \frac{t\frac{\varepsilon}{3}}{f_1 - \varepsilon'} \rceil \geq f_1 + \frac{t\varepsilon}{3}$ for some other positive ε'. So again, $\textsc{Cost}_{\text{ON}}(X_t) \geq t\frac{\varepsilon}{3} + f_1 + t\frac{\varepsilon}{3} + f_1 = 2(f_1 + t\frac{\varepsilon}{3})$ which is twice of $\textsc{Cost}_{\text{OFF}}(X_t)$.

In both cases the cost of ON is at least twice that of OFF which contradicts the assumption that ON is $(2-\varepsilon)$-competitive. ◀

C.5 Proof of Lemma 5

Proof. Accepting a transaction x incurs a cost of x for increasing funds. Rejecting a transaction x incurs a cost of $Rx + f_2$. So any big transaction should be rejected. ◀

C.6 Proof of Theorem 6

Proof. From Lemma 5, OFF rejects big transactions. Thus, ON should also reject these transactions.

While $A_t = 0$, both OFF and ON reject all transactions in the sequence and both incur the same cost. The moment $A_t > 0$, we know that OFF recharged with an amount at least A_t to accept all small transactions in the sequence. Thus $\textsc{Cost}_{\text{OFF}}(X_t) \geq A_t + f_1$.

At this time ON would have rejected the small transactions in the sequence for at most a cost of $A_t + f_1$ together with some additional recharging cost. From Lemma 1, we know that the recharging cost for ON is at most

$$A_t + \frac{\sqrt{5}-1}{2} f_1 + \left\lceil \frac{A_t}{\frac{\sqrt{5}-1}{2} f_1} \right\rceil f_1 \leq A_t + \frac{\sqrt{5}-1}{2} f_1 + \frac{A_t}{\frac{\sqrt{5}-1}{2}} + f_1.$$

Summing up both costs, we get

$$\begin{aligned} \textsc{Cost}_{\text{ON}}(X_t) \leq & A_t + f_1 + A_t + \frac{\sqrt{5}-1}{2} f_1 + \frac{A_t}{\frac{\sqrt{5}-1}{2}} + f_1 \\ = & \left(2 + \frac{\sqrt{5}-1}{2}\right) \textsc{Cost}_{\text{OFF}}(X_t) \end{aligned}$$

◀

C.7 Proof of our main Theorem

Here, we detail the full proof of Theorem 9

Proof. We know that for any i, $\textsc{Cost}_{\text{OFF}}(X_i) \geq \textsc{Cost}_{\text{OFF}}(X_{i-1})$. From Lemma 1 and Lemma 2, we know that cost of ON for recharging (in Algorithm 4) is at most $(5 + 2\lceil \log C \rceil)\textsc{Cost}_{\text{OFF}}(X_t)$. Let t_1 and t_2 (with $t_2 > t_1$) be the two consecutive times ON recharges, then we show that the cost of ON for rebalancing and rejection is smaller than $2(\textsc{Cost}_{\text{OFF}}(X_{t_2}) - \textsc{Cost}_{\text{OFF}}(X_{t_1}))$. Then $\textsc{Cost}_{\text{ON}}(X_t) \leq (7 + 2\lceil \log C \rceil)\textsc{Cost}_{\text{OFF}}(X_t)$.

For every strategy of OFF and any two consecutive recharging times t_1 and t_2, we show that the rebalancing and rejection cost of ON between times t_1 and t_2 is at most twice that of OFF as defined by the strategy. Having the strategy of OFF, we split the time between rechargings even further, into epochs; we will show that the competitive ratio of 2 holds for every epoch.

The left epoch starts with the first transaction that makes some bucket in B^ℓ non-full (smaller than the original amount); the left epoch ends either before ON recharges, or $B_o^\ell > 0$. In a left epoch, every transaction from the right side is accepted; non-fullness of some buckets on one side means $B_o > 0$ on the other side. The right epoch is defined similarly, but since the epochs are disjoint, we can prove the statement for a left epoch only.

For transactions below $\frac{F_{tracker}}{C}$, we argue that the cost of ON is at most the cost of OFF. ON accepts everything, so ON pays only for rebalancing. OFF either rebalances too, in which case the cost is the same as ON; or it rejected some transactions. Since ON starts with $2F_{tracker}$ funds in B_s and refills the bucket with the highest priority, this means OFF rejected some transactions summing to at least $F_{tracker}$. There are at least C of them, so OFF's cost is also above Cf_2. If there is a counterexample containing a small transaction that OFF rejects, then we can modify it to a counterexample where the transaction is increased to $\frac{F_{tracker}}{C}$. So we can show the ratio in the case that no small transactions are coming.

Now that we have the strategy for OFF (decisions before rebalancing), we define some variables that track the competitive ratio. We will look at incoming transactions, and prove that the competitive ratio is always below 2. We say that a transaction x belongs to a bucket B_i if $\frac{F_{tracker}}{2^i} < x \leq \frac{F_{tracker}}{2^{i-1}}$. A transaction is red if it is rejected by ON and accepted by OFF and it is blue if it is accepted by ON and rejected by OFF. Let ρ_i (β_i) be the number of red (blue) transactions in the bucket B_i. We can disregard transactions for which ON and OFF make the same decision. If the transaction is rejected by both, it improves the ratio. If the transaction is accepted by both, it can be simulated by decreasing $F_{tracker}$.

We prove by induction that $\sum_{k\leq j} \rho_k \leq 2\sum_{k\leq j} \beta_k$ for all j. We show that if the equality holds and OFF has enough funds to accept incoming transactions, ON accepts too.

Let us examine $j = \lceil \log C \rceil$. We know that the ratio between any two transaction sizes in $B_{\lceil \log C \rceil}$ is less than 2, so any two red transactions are bigger than one blue. Moreover, all funds that arrived from the right side were put into $B_{\lceil \log C \rceil}$ (if it is not full). So if $\rho_{\lceil \log C \rceil} = 2\beta_{\lceil \log C \rceil}$, ON has at least the amount of funds in $B_{\lceil \log C \rceil}$ OFF has. To continue the induction for buckets with smaller indices, we reassign some red or blue transactions to different buckets. If $\rho_{\lceil \log C \rceil} < 2\beta_{\lceil \log C \rceil}$, we move at most one red and some blue transactions (in decreasing order of size) to $B_{\lceil \log C \rceil -1}$, stopping just before $\rho_{\lceil \log C \rceil} \geq 2\beta_{\lceil \log C \rceil}$.

For general j, we know that, due to the reassignment, in every bucket smaller than j, ON rejected exactly twice the number of transactions OFF did. Moreover, OFF needs to use at least the same amount of funds to accept red transactions compared to funds needed by ON to accept blue ones. Now, in the bucket B_j holds $\rho_j = 2\beta_j$. Again, we pair every two red transactions to one blue, such that the sum of red is bigger than blue. Before the reassignment, the ratio between any two transactions is at most 2. The reassignment (if occurred) moved at most one red and at least one blue that is smaller than any original transaction in the bucket, so we can pair the moved red to moved blue. In transactions in buckets in j and bigger, OFF used more funds than ON. Any funds that arrived from the right side were put into some bucket in j or below, so if OFF has enough funds to accept, ON has too.

The same argument holds for a right-epoch, and we note that epochs are disjoint and cover the entire transaction sequence between times t_1 and t_2. Since we chose the consecutive recharging times t_1 and t_2 arbitrarily, the rebalancing and rejection cost of ON between any two consecutive rechargings is at most twice that of OFF within the same period. Therefore, Algorithm 4 is $7 + 2\lceil \log C \rceil$ competitive. ◀

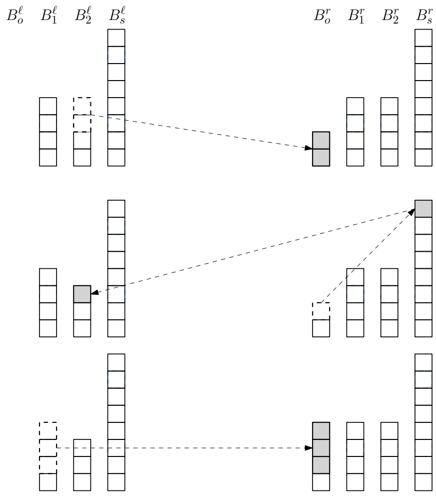


Figure 3 An example of how funds are transferred across a payment channel and how buckets are refilled. Both ℓ and r start with full buckets. The first transaction is in the left-to-right direction and is transferred using funds from B_2^ℓ to B_o^r. The second transaction is in the right-to-left direction and is small, thus funds from B_s^r are used. B_s^r is immediately refilled using funds from B_o^r. The third transaction is in the left-to-right direction and uses funds from B_1^ℓ.

C.8 Proof of Lemma 10

Proof. We describe an epoch: OFF starts with A funds left, then some transactions are sent from left to right and finally one transaction of size A is sent right to left. If the funds in the channel of ON is smaller than σ, then the cost of ON is more than c times that of OFF.

One epoch consists of at most $\frac{\log C}{\log c+1} + 1$ phases. In phase i (starting from $i = 0$), there are $(c+1)^i$ transactions of size $\frac{A}{(c+1)^i}$. OFF always accepts all transactions in the latest phase. If at the end of any phase, the cost of ON is more than c times of OFF, then a transaction of size A is sent back and another epoch starts. Observe that for $c < \frac{\log C}{\log \log C}$, after rebalancing the epoch ends too (because the cost is c times bigger). We can assume this cannot happen, ON does not perform off-chain rebalancing.

We compute how much funds ON needs to stay within the competitive ratio until the last phase (where transactions of sizes $\frac{A}{C}$ are sent). After phase i, OFF accepted $(c+1)^i$ transactions and rejected $\frac{(c+1)^i-1}{c}$, so OFF can reject up to $(c+1)^i - 1$ transactions among $\frac{(c+1)^{i+1}-1}{c}$. So ON has to accept at least $\frac{(c+1)^{i+1}-1-c(c+1)^i+c}{c} = (c+1)\frac{(c+1)^{i-1}-1}{c}$ transactions.

The size of transactions is decreasing, so optimally, ON accepts transactions when they are needed. So in phase $i+1$ it needs to accept $(c+1)\frac{(c+1)^i-1}{c} - (c+1)\frac{(c+1)^{i-1}-1}{c} = (c+1)^i$ transactions. Of course, it needs to accept the transaction in the phase 0.

The cost of transactions accepted by ON in phase $i > 0$ is $(c+1)^{i-1}\frac{A}{(c+1)^i} = \frac{A}{c+1}$. To maintain the competitive ratio ON needs to accept transactions worth $A + A\frac{\log C}{\log c+1}$ in total. Independently of the cost of OFF and ON before, there can be one epoch after another where the ratio is worse than c, so at the end, the ratio would be above c. ◀

D Helpful diagrams

E Computing the cost of OFF using dynamic programming

In this section, we describe a dynamic programming algorithm ON that solves the main problem. We assume that the size of transactions is integer (moreover the sum of transactions should be small).

Let $\textsc{Cost}_i(F_\ell, F_r)$ be the minimum cost for rejecting and off-chain rebalancing in processing sequence X_i that ends with $b(\ell) = F_\ell$ and $b(r) = F_r$ (For values of F_ℓ and F_r smaller that 0, we define it to be ∞). $\textsc{Cost}_i(F_\ell, F_r)$ can be derived from $\textsc{Cost}_{i-1}$ given the decision on the i'th transaction.

Let us assume wlog that i'th transaction is from ℓ to r. OFF has three choices when encountering x_i. The first option is to reject x_i, the the cost is $A_1 = \textsc{Cost}_{i-1}(F_\ell, F_r) + Rx_i + f_2$. The second option is to accept x_i which gives cost $A_2 = \textsc{Cost}_{i-1}(F_\ell + x_i, F_r - x_i)$. The last option is to off-chain rebalance before x_i and then accept x_i. Note that any off-chain rebalancing before rejecting or accepting (while having enough funds) can be postponed. This gives cost $A_3 = \min_{a \le F_\ell + x_i} \textsc{Cost}_{i-1}(a, F_r + F_\ell - a) + C \cdot R(F_\ell + x_i - a) + Cf_2$. OFF chooses the best option, that means $\textsc{Cost}_i(F_\ell, F_r) = \min\{A_1, A_2, A_3\}$

We handle right to left transaction in the same way.

Given the previous, ON computes $\textsc{Cost}_t$ for all valid pairs (F_ℓ, F_r) and the final cost is

$$\textsc{Cost}_{\text{OFF}}(X_i) = \min_{F_\ell, F_r} \textsc{Cost}_t(F_\ell, F_r) + F_\ell + F_r + f_1$$

To bound the time complexity of ON, we observe some bounds for $S = F_\ell^* + F_r^*$, where F_ℓ^* and F_r^* are the values of F_ℓ and F_r achieving the minimal cost. Observe that $S \le \sum_{i \le t} x_i$, it is not worth to have more money than the sum of the trasactions. We can strenghten the inequality and instead of $\sum_{i \le t} x_i$, we can compute the minimal amount needed to accept every transaction. The other option is to reject everything, so we know that $f_1 + S \le \sum_{i \le t} Rx_i + f_2$.

Now we can prove the theorem about the described algorithm ON.

▶ **Theorem 12.** *ON computes the optimal cost in time $\mathcal{O}(tS^3)$, where S is the bound on the maximal funds in the channel and t the number of transactions.*

Proof. In the dynamic programming, we take into account all possible decisions ON can make, by this, correctness follows.

The algorithm ON tries all possible amounts between 1 and S and starting distributions. There are S^2 of them. While computing one value, it needs to look at at most S precomputed values. And it needs to do it at most t times. ◀

Using dynamic programming for calculating OFF has two advantages. First, we can easily recover the decisions of OFF. Secondly, dynamic programming provides us with optimum solution for all subsequences of X_t. This is useful for implementing Algorithm 4.

F Heuristics to improve the average-case performance of our online algorithm

We notice in our experiments that ON seems to overcharge the channel. This is most noticeable when we observe the effect of C on the performance of ON. From Table 2, increasing C in a range of medium (not too small) values does not change OFF's cost noticeably. In contrast, both the average cost and total amount of locked funds of ON grows with C. This is due to the fact that ON uses $(4 + 2\lceil \log C \rceil, f_1)$-recharging to ensure that it always has significantly more funds than OFF, even though a big fraction of these funds remain unspent. ON is also limited by the fact that it does not borrow funds from other buckets when a bucket is depleted. For instance, ON always charges B_s to $2F_{tracker}$ and only uses these funds to accept transactions that are smaller than $\frac{F_{tracker}}{C}$. Thus, as C increases, the number of transactions that fall into the B_s bucket decreases and the funds in B_s remain unspent.

These observations motivate us to design a less pessimistic version of ON that we expect will perform better than ON. We introduce ON-I which is a slightly altered version of ON: ON-I follows the $(\lceil \log C \rceil, f_1)$-recharging algorithm and does not divide the funds into separate buckets. Instead, ON-I accepts all the transactions smaller than $F_{tracker}$ as long as it has funds. Otherwise, if the transaction is small ($< \frac{F_{tracker}}{C}$) it off-chain rebalances to fill the bucket and accepts the transaction. Similar to ON, ON-I rejects a transaction if it is larger than $\frac{F_{tracker}}{C}$.

Our empirical results in Table 3 confirms that the average cost of ON-I is significantly smaller than ON. The acceptance rate of ON-I is slightly smaller than ON, which is expected as ON-I does not have separate funds for each range of transactions (as defined by the buckets), and as a result might miss some transactions. We observe that ON-I performs more off-chain rebalancings compared to ON on average because ON-I does not reserve separate funds for small transactions. However, one issue with both ON and ON-I is that they recharge the channel too often (as soon as $F_{tracker} < A(X_t)$). As can be seen from Table 2 ($f_2 = 2$), both algorithms perform more than 5 rechargings on average for transaction sequences of length 50. This increases the cost of both algorithms significantly as each recharging instance incurs a cost of at least f_1.

We thus design another version of ON-I to address the aforementioned problem. ON-II works exactly as ON-I except that it does not recharge the channel as frequently as ON-I does. ON-II only recharges the channel if $\alpha \cdot F_{tracker} < A(X)$, where $\alpha > 1$ is some constant that controls the how often the algorithm recharges the channel and can be fine-tuned empirically based on f_1 and f_2. If we set $\alpha = 1$, ON-II becomes equivalent to ON-I and has higher acceptance rate. This is favorable when f_2 is large and f_1 is small. Conversely, by increasing α, ON-II recharges the channel less frequently but the acceptance rate falls. This is favorable when f_1 is large and f_2 is small. In our experiments, we observe that for the case $f_2 = 2, C = 2$, when all the other parameters are as Table 2, $\alpha = 2$ yields the lowest average cost. Thus, in our evaluation of ON-II, we use $\alpha = 2$ and from Table 2 we note that this choice of α halves the number of rechargings compared to ON-I, which consequently leads to lower average cost. Additionally, we note that the total amount of funds in the channel of ON-II is close to OFF.

F.0.0.1 Varying the distribution of the generated sequences.

We also evaluate how the performance of our algorithms is affected by the variance of the transaction size. We sample 50 sequences each of length 50 with each transaction in the sequence independently sampled from the folded normal distribution with mean 0 and standard deviation σ, for a range of σ values across $[3, 20]$. We then observe the cost of ON and OFF. As can be seen from in Figure 4a, the cost of both algorithms rises as σ increases. This is due to the fact that increasing the variance of the sampled transactions reduces the probability of getting a similarly sized transaction coming from the other side, thus increasing the speed at which the balance on one side gets depleted. We note, however, that Figure 4a shows that even for large values of σ, ON's average cost remains a lot smaller than the worst case upper bound of $7 + 2\lceil \log C \rceil$.

Another factor we look at is how the asymmetry of the transaction flow along a channel can affect the performance of our algorithms. To do so, we generate 50 sequences each of length 50, and sample the size of each transaction from a folded normal distribution with mean 0 and standard deviation 3. We then sample the direction of the transactions according to a Bernoulli distribution with parameter p, where p represents the probability of sampling a left-to-right transaction. We see from Figure 4b that the cost of all algorithms decrease as p increases from 0 to 0.5. As p increases from 0.5 to 1, the cost function increases again. This conforms to our intuition that extremely asymmetric sequences are harder to handle as the lack of sufficiently many transactions from one side just increases the speed at which the balance on the other side gets depleted.

Param		OFF					ON				
C	f_2	Cost	$A(X)$	Accept rate	Off-chain re-balancing	Rechar-gings	Cost	$A(X)$	Accept rate	Off-chain re-balancing	Rechar-gings
2	0.5	**15.02**	6.4	0.78	0.8	1	**63.3**	44.26	0.50	0.9	2.18
8	0.5	**15.21**	6.38	0.77	0	1	**87.79**	69.06	0.50	0	2.04
2	2	**23.6**	14.26	0.95	5.36	1	**127.02**	100.2	0.91	0.38	5.86
8	2	**24.5**	13.9	0.92	0	1	**184.32**	156.6	0.9	0	5.84

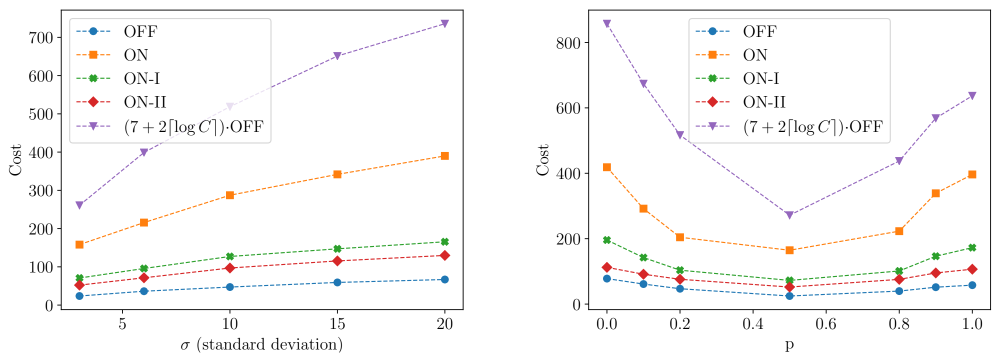
Param		ON-I					ON-II				
C	f_2	Cost	$A(X)$	Accept rate	Off-chain re-balancing	Rechar-gings	Cost	$A(X)$	Accept rate	Off-chain re-balancing	Rechar-gings
2	0.5	**30.74**	6.86	0.44	11.18	2.18	**26.52**	5.56	0.41	10.22	1.2
8	0.5	**39.98**	19.86	0.48	0	2.04	**32.94**	15.9	0.46	0	1.18
2	2	**66.16**	14.42	0.84	25.48	5.86	**60.38**	11.3	0.78	27.16	2.2
8	2	**81.35**	42.72	0.89	1.5	5.84	**58.38**	33.3	0.83	1.56	2.16

Table 3 Comparison between the performance of OFF, ON, ON-I and ON-II on randomly generated transaction streams. The result is averaged over 50 sequences each of length 50. The size of each transaction is independently sampled from the folded normal distribution with mean 0 and standard deviation 3, then quantised to the closest integer. We set $f_1 = 3$ and $R = 0$. $A(X)$ is the total amount of funds in the channel from recharging the channel. "Accept rate" shows the average fraction of transactions that were accepted. "Off-chain rebalancing" shows how much funds on average was moved along the channel using off-chain rebalancing. "Rechargings" shows the average number of rechargings performed. Note that since OFF knows the entire sequence in advance, it only recharges the channel once at the beginning of each sequence.

G Case study: Lightning Network

We also ran a case study of the Lightning network. We first run our experiments with realistic parameters taken from Lightning Network data. In the Lightning Network, f_1 is the on-chain transaction fee (roughly 1000 satoshi) which is a lot larger than f_2, the base fee one receives when forwarding a payment (around 1 satoshi).

From analysis of the Lightning Network (We use snapshot from September 2021) [7], we know that the average cycle length is 4.15 (after excluding roughly 10% of vertices that are not part of any cycle). That means the value C in Theorem 9 is just slightly above 4. Details are in Table 4.



(a) σ is the standard deviation of the normal distribution.

(b) p is the probability of sampling a left-to-right transaction.

Figure 4 Average cost of our algorithms over 50 randomly generated transaction streams each of length 50. In both figures the size of each transaction is sampled from the folded normal distribution with mean 0. The standard deviation of the normal distribution is fixed to 3 in the right figure, however in the left figure the standard deviation is varying on the x-Axis. We use the parameters $f_1 = 3, f_2 = 2, R = 0, C = 4, \alpha = 2$.

Cycle length	≤ 4	5	6	7	N.A.
Frequency	$49,424(77.44\%)$	$7,758(12.16\%)$	$469(0.73\%)$	$12(0.02\%)$	$6,157(9.65\%)$

Table 4 Frequency of the length of shortest cycle between all users in the Lightning Network. The last column shows the frequency of channels that are not part of any cycle (N.A. not applicable) The average cycle length is 4.15