Theoretical Computer Science 989 (2024) 114353

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Weighted packet selection for rechargeable links in cryptocurrency networks: Complexity and approximation ☆

Stefan Schmid [a], Jakub Svoboda [b,*], Michelle Yeo [b]

[a] Technische Universität Berlin, Germany
[b] Institute of Science and Technology, Austria

ARTICLE INFO

Keywords:
Network algorithms
Approximation algorithms
Complexity
Cryptocurrencies
Payment channel networks

ABSTRACT

We consider a natural problem dealing with weighted packet selection across a rechargeable link, which e.g., finds applications in cryptocurrency networks. The capacity of a link (u, v) is determined by how many nodes u and v allocate for this link. Specifically, the input is a finite ordered sequence of packets that arrive in both directions along a link. Given (u, v) and a packet of weight x going from u to v, node u can either accept or reject the packet. If u accepts the packet, the capacity on link (u, v) decreases by x. Correspondingly, v's capacity on (u, v) increases by x. If a node rejects the packet, this will entail a cost affinely linear in the weight of the packet. A link is "rechargeable" in the sense that the total capacity of the link has to remain constant, but the allocation of capacity at the ends of the link can depend arbitrarily on the nodes' decisions. The goal is to minimise the sum of the capacity injected into the link and the cost of rejecting packets. We show that the problem is NP-hard, but can be approximated efficiently with a ratio of $(1+\varepsilon)\cdot(1+\sqrt{3})$ for some arbitrary $\varepsilon > 0$.

1. Introduction

This paper considers a novel and natural throughput optimization problem where the goal is to maximise the number of packets routed through a network. The problem variant comes with a twist: link capacities are "rechargeable", which is primarily motivated by payment-channel networks routing cryptocurrencies.

We confine ourselves to a single capacitated network link and consider a finite ordered sequence of packet arrivals in both directions along the link. This can be modelled by a graph that consists of a single edge between two vertices u and v, where b_u and b_v represent the capacity u and v inject into the edge respectively. Each packet in the sequence has a weight (or value) and a direction (either going from u to v, or from v to u). When u forwards a packet going in the direction u to v, u's capacity b_u decreases by the packet weight and v's capacity b_v correspondingly increases by the packet weight (see Fig. 1 for an example). Node u can also reject to forward a packet, incurring a cost linear in the weight of the packet. The links we consider are rechargeable in the sense that the total capacity $b_u + b_v$ of the link can be arbitrarily distributed on both ends, but the total capacity of the link cannot be altered throughout the lifetime of the link. Given a packet sequence, our goal is to minimise the sum of the cost of rejecting packets and the amount of capacity allocated to a link.

☆ This article belongs to Section A: Algorithms, automata, complexity and games, Edited by Paul Spirakis.

* Corresponding author.
E-mail addresses: stefan.schmid@tu-berlin.de (S. Schmid), jsvoboda@ist.ac.at (J. Svoboda), myeo@ist.ac.at (M. Yeo).

https://doi.org/10.1016/j.tcs.2023.114353
Received 12 July 2023; Accepted 15 December 2023

Available online 29 December 2023

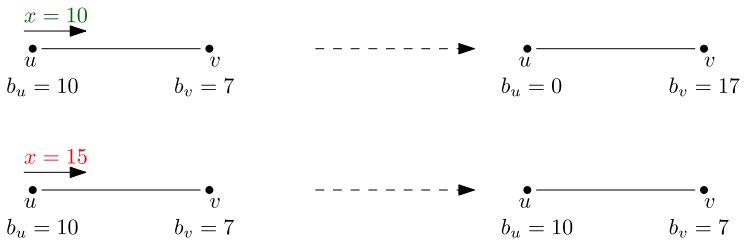

Fig. 1. The diagram on the top shows the outcome of u successfully processing a packet x of weight 10 along the link (u, v). The subsequent capacities of u and v are 0 and 17 respectively. The diagram on the bottom shows the outcome where, even though the total capacity of the (u, v) link is 17, u's capacity of 10 on (u, v) is insufficient to forward a packet x of weight 15. As such, the subsequent capacities of u and v on the link (u, v) remain the same.

The primary motivating example of our model is payment channel networks [9,10] supporting cryptocurrencies [1,14]. These networks are used to route payments of some amount (i.e. weighted packets in our model) in a multi-hop fashion between any two users of the network. In this way, users can directly transact with other users off-chain, and in so doing avoid the hefty transaction fees as well as long delays they would incur when transacting on the blockchain. Any two users in a payment channel network can create a channel (i.e. rechargeable link in our model) between themselves and deposit some funds only to be used in this channel (i.e. the initial capacity injected at each endpoint in our model). We note that users can always retrieve their funds in the payment channel at any time, but this would involve closing the channel and taking out the funds. For users that transact frequently and hence use payment channel networks, frequently closing channels and withdrawing their funds would defeat the purpose of them using payment channels as they would now need go back to transacting on the blockchain which is costly. Thus, the amount of funds injected into the payment channel can be seen as a "cost" for keeping the payment channel open to avoid using the blockchain. The total amount of funds deposited in the channel is its total capacity and remains invariant for the lifetime of the channel. Each payment moving across the channel simply updates the current balances (i.e. capacity at each end point of the link) of the two users in the channel, while maintaining that the total amount of funds in the channel remains the same.

Routing payments in payment channel networks comes with a profit: intermediate nodes on a payment route typically charge a fee for forwarding payments that is linear in the payment amount. Hence, if users reject to forward a payment, they would lose out on profiting from this fee and thereby incur the fee amount as opportunity cost. However, a depleted channel (i.e. a link with capacity 0 at one end) due to indiscriminate forwarding of payments can also impact transaction throughput. In particular, a depleted channel cannot forward any further payments unless the channel is closed and reopened with larger capacity, which also incurs corresponding cost. Hence the choice of how much capacity to inject into a channel and which transactions to forward and which to reject is crucial to maintain the lifetime of a payment channel [5,12,4]. Channels in payment channel networks are also rechargeable for security reasons, see [14] for more details.

Here we stress a crucial difference between our problem and problems on optimising flows and throughput in typical capacitated communication networks [7,15]. In traditional communication networks, the capacity is usually independent in the two directions of the link [11]. In our case, however, the amount of packets u sends to v in a link (u, v) directly affects v's capability to send packets, as each packet u sends to v increases v's capacity on (u, v).

We start with a description of rechargeable links, then explain the actions nodes can take and corresponding costs. Finally, we state our main results.

Rechargeable links One unique aspect of our problem is that the links we consider are rechargeable. Rechargeable links are links that satisfy the following properties:

1. Given a link (u, v) with total capacity M, the capacity can be arbitrarily split between both ends based on the number and weight of packets processed by u and v. That is, b_u and b_v can be arbitrary as long as $b_u + b_v = M$ and $b_u, b_v \geq 0$. See Fig. 1 for an example of how b_u and b_v can vary in the course of processing packets.
2. The total capacity of a link is invariant throughout the lifetime of the link. That is, it is impossible for nodes to add to or remove any part of the capacity in the link. In particular, if a node is incident to more than one link in the network, the node cannot transfer part of their capacity in one link to "top up" the capacity in the other one.

Node actions and costs First, we note that creating a link incurs an initial cost of the amount the node allocates in the link. That is, if node u allocates b_u in link (u, v), the cost of creating the link (u, v) for u would be b_u. Consider a link (u, v) in the network and a packet going from u to v along the edge. Node u can choose to do the following to the packet:

– **Accept packet.** Node u can accept to forward the packet if their capacity in (u, v) is at least the weight of the packet. The result of doing so decreases their capacity by the packet weight and increases the capacity of v by the packet weight. Note that apart from gradually depleting a node's capacity, accepting the packet does not incur any cost.
– **Reject packet.** Node u can also reject the packet. This could happen if u's capacity is insufficient, or if accepting the packet would incur a larger cost in the future. For a packet of weight x, the cost of rejecting the packet is $f \cdot x + m$ where $f, m \in \mathbb{R}^+$.

We note that node u does not need to take any action for packets going in the opposite direction (i.e. from v to u) as these packets do not affect u's cost. See Section 2 for more detail regarding packets.

Our contributions We introduce the natural weighted packet selection problem and show that it is NP-hard by a reduction from subset sum. Our main contribution is an efficient constant-factor approximation algorithm. We further initiate the discussion of how our approach can be generalised from a single link to a more complex network.

Organisation Section 2 introduces the requisite notations and definitions we use in our paper, and also a formal statement of the weighted packet selection for a link problem. Section 3 provides the necessary algorithmic building blocks we use to construct our main algorithm. In Section 4, we present our main approximation algorithm and prove that it achieves an approximation ratio of $(1+\varepsilon)(1+\sqrt{3})$ for weighted packet selection in Theorem 1. We show that weighted packet selection for a link is NP-hard in Section 5. Finally, we discuss some possible generalisations of our algorithm from a single link to a larger network in Section 6. We conclude our work by discussing future directions in Section 7.

2. Notation and definitions

Packet sequence Let (u, v) be a link. We denote an ordered sequence of packets by $X_t = (x_1, \ldots, x_t)$. Each packet $x_i \in X_t$ has a weight and a direction. We simply use $x_i \in \mathbb{R}^+$ to denote the weight of the packet x_i. We say a packet x_i goes in the left to right direction (resp. right to left) if it goes from u to v (resp. from v to u). Let $X_{\rightarrow}$ denote the subsequence of X_t that consists of packets going from left to right and $X_{\leftarrow}$ the subsequence of X_t that consists of packets going from right to left. For an integer $t \geq 1$, we use $[t]$ to denote $\{1, \ldots, t\}$.

Problem definition We now formally define weighted packet selection for a link. The input to our problem is a rechargeable link (u, v) and a sequence of packets X_t arriving on that link. We adopt the optimisation problem perspective over the *entire link*, instead of individual nodes. That is, we suppose nodes u and v collaborate and act as a coalition regardless of how they decide to initially split the capacity on both ends. The problem therefore is to compute the initial capacity and distribution (how it should be split on both ends) on the link as well as to decide on whether to accept or reject each packet in X_t such that the overall solution minimises the sum of the rejection cost as well as the cost of the capacity locked in the link.

Optimal algorithm and costs Let $x_{\min}$ be the weight of the packet with the smallest weight in X_t and $M_{\max}$ be the total capacity in the link needed to accept all packets. $M_{\max}$ for X_t is easy to compute in time $\mathcal{O}(t)$ and is upper bounded by the sum of the weight of all packets in X_t. Similarly, given any sequence of *decisions*, we can compute the minimal cost of the capacity locked in the link and optimal initial distribution of capacity by greedy simulation. Let OPT be the cost of the optimal algorithm and OPT_M be the cost of the optimal algorithm using a capacity of M in the link. Additionally, we use OPT^R to denote the cost of the optimal algorithm for rejecting packets and OPT^C to denote the corresponding capacity cost (i.e., amount of capacity injected in the link). Similarly, we use OPT^R_M to denote the cost for rejecting packets of the optimal algorithm using a capacity of M in the link (note that $OPT^C_M = M$, $OPT = OPT^C + OPT^R$, and $OPT^R_M \leq OPT^R$).

3. Preliminary insights and algorithmic building blocks

We start our investigation of the weighted packet selection problem by describing a procedure to approximate the optimal capacity in a link using binary search and use this approximation to derive a lower bound on the cost of the optimal algorithm. We then describe a linear program that fractionally accepts packets (i.e. part of a packet can be accepted) given a fixed link capacity M and show that the solution of the linear program given M is a lower bound on the cost of the optimal algorithm given M. These results are used as building blocks for our main algorithm and theorem in Section 4. Nevertheless, we also present a simpler example of how to use the solution of the linear program that also comes with some guarantees in Section 3.3 which may be of independent interest.

3.1. Approximating the optimal capacity

We present a lemma that allows us to fix the capacity of the link to some value $M \in \mathbb{R}^+$ for a small trade-off in the approximation ratio. Recall that $x_{\min}$ is the weight of the smallest packet in X_t and $M_{\max}$ is the capacity needed to accept all packets. Observe that if the optimal capacity is not 0, it has to lie in the interval $[x_{\min}, M_{\max}]$. We thus fix some $\varepsilon > 0$ and perform a search for M over all $k \in \mathbb{N}$ such that $x_{\min}(1+\varepsilon)^k \leq M_{\max}$. Let us denote by LB_M any lower bound on OPT^R_M, the optimal rejection cost using at most capacity M.

Lemma 1. *For any $\varepsilon > 0$, let $\mathcal{M} = \{x_{\min}(1+\varepsilon)^k \mid k \in \mathbb{N}$ and $x_{\min}(1+\varepsilon)^k \leq M_{\max}\} \cup \{0\}$. Then, the following inequality holds:*

$$\min_{M \in \mathcal{M}} \left(LB_M + \frac{M}{1+\varepsilon} \right) \leq OPT$$

Proof. We first analyse the case where the optimal algorithm rejects all packets. In this case, we know that since $\mathcal{M}$ contains 0, $LB_0 \leq OPT_0^R = OPT_0 = OPT$, so the inequality holds.

Now, suppose that the optimal algorithm accepts at least one packet. This means $OPT^C \geq x_{\min}$. So there exists a $k \in \mathbb{N}$ such that $x_{\min}(1+\varepsilon)^{k-1} \leq OPT^C \leq x_{\min}(1+\varepsilon)^k$. Set $M = x_{\min}(1+\varepsilon)^k$. We need to show that $LB_M + \frac{M}{1+\varepsilon} \leq OPT = OPT^R + OPT^C$. From the way we choose M, we know that $\frac{M}{1+\varepsilon} \leq OPT^C$.

Now we just need to show $LB_M \leq OPT^R$. Observe that the optimal rejection cost for any link with larger capacity is always at most the rejection cost for any link with smaller capacity, as in the worst case the algorithm in the former setting accepts the same set of packets that the algorithm in the latter setting accepts. Thus, for any $M' \geq M$, $OPT_{M'}^R \leq OPT_M^R$. And since we chose M as an upper bound on OPT^C, it means $OPT^R = OPT_{OPT^C}^R \geq OPT_M^R \geq LB_M$. □

Looking ahead, we describe an algorithm that is a $(1+\sqrt{3})$-approximation of LB_M in Section 4. Thus, together with Lemma 1, we can use this algorithm to approximate weighted packet selection with a ratio of $(1+\varepsilon)(1+\sqrt{3})$ by running the algorithm at most $\frac{1}{\varepsilon}\log\frac{M_{\max}}{x_{\min}}$ times. We note that choosing a smaller value of ε yields a better approximation, but increases the running time.

3.2. Linear program formulation

Here, we describe a linear program that computes a lower bound for OPT_M^R. We first observe that due to the capacity constraints, the optimal algorithm with capacity M cannot accept packets with weight larger than M. Hence, for the rest of the analysis, we assume that all packets in X_t have weight less than M.

In the linear program, we allow accepting a fractional amount of a packet. That is, we create a variable $0 \leq y_i \leq x_i$ for every packet $x_i \in X_t$ that represents the extent to which the packet is accepted. For instance, $y_i = \frac{x_i}{2}$ means that half of x_i is accepted. We introduce variables $S_{L,i}$ and $S_{R,i}$ denoting the capacity on the left and right ends of the link after processing first i packets from X_t. We reiterate that due to the rechargeable property of the link, $S_{L,i} + S_{R,i} = M$, and $0 \leq S_{L,i}, S_{R,i} \leq M$.

We can now formulate the linear program in eq. (1):

$$\begin{aligned}
\text{minimise} \quad & \sum_i f(x_i - y_i) + m\frac{x_i - y_i}{x_i} && (1)\\
\text{subject to} \quad & \forall i : y_i, S_{L,i}, S_{R,i} \geq 0 \\
& \forall i : y_i \leq x_i \\
& \forall i : S_{L,i} + S_{R,i} = M \\
& \forall x_i \in X_{\rightarrow} : S_{L,i} = S_{L,i-1} - y_i \\
& \forall x_i \in X_{\rightarrow} : S_{R,i} = S_{R,i-1} + y_i \\
& \forall x_i \in X_{\leftarrow} : S_{L,i} = S_{L,i-1} + y_i \\
& \forall x_i \in X_{\leftarrow} : S_{R,i} = S_{R,i-1} - y_i
\end{aligned}$$

Let LP_M be the solution of the linear program with capacity parameter M. The following lemma states that LP_M is a lower bound of the optimal cost of the weighted packet selection for a link problem with capacity M.

Lemma 2. *$LP_M \leq OPT_M$ for all M.*

Proof. OPT_M is an admissible solution to the linear program. If some other (fractional) solution is found, we know that it is at most OPT_M. □

The linear program can be solved in time $\mathcal{O}(n^\omega)$ where n is the number of variables in the linear program and ω the matrix multiplication exponent [8] (currently ω is around 2.37).

3.3. Example algorithm

We present an example of how to use the solution of the linear program in Section 3.2 for some fixed capacity M to create an algorithm that uses twice as much capacity as the linear program but guarantees that all packets that are fully accepted by the linear program (i.e. $x_i = y_i$) will also be fully accepted by the algorithm.

Algorithm 1 describes the decision making process only for packets coming from left to right on the link, i.e. $X_{\rightarrow}$. As the decision process for $X_{\leftarrow}$ is symmetric, we omit it to avoid repetition. The algorithm takes as input the solution to the linear program and the packet sequence X_t. Recall that $S_{L,i}$ and $S_{R,i}$ for $i \in [t]$ are the capacity distributions from the linear program solution on the left and right end of the link respectively after processing the ith packet. The algorithm uses the initial distribution $S_{L,0}$ and $S_{R,0}$, and additionally splits the extra M capacity into 2 "reserve buckets" R_L and R_R of size $\frac{M}{2}$ each on both ends. Thus, the initial capacity

Algorithm 1 Algorithm accepting all fully accepted packets.

Input: packet sequence X_t, capacity M, solution of LP: $S_{L,i}, S_{R,i}, y_i$.
Output: decisions to accept or reject
1: initialise $R_L = \frac{M}{2}$, $R_R = \frac{M}{2}$
2: **for** $i \in [t]$ **do**
3: **if** $x_i \in X_{\rightarrow}$ **then**
4: **if** $R_L \geq x_i - y_i$ **then**
5: **Accept**
6: $R_L = R_L - (x_i - y_i)$
7: $R_R = R_R + (x_i - y_i)$
8: $S_{L,i} = S_{L,i} - y_i$
9: $S_{R,i} = S_{R,i} + y_i$
10: **else**
11: **Reject**
12: $R_L = R_L + y_i$
13: $R_R = R_R - y_i$
14: $S_{L,i} = S_{L,i} - y_i$
15: $S_{R,i} = S_{R,i} + y_i$

of the left node would be $S_{L,0} + R_L$ and the initial capacity of the right node would be $S_{R,0} + R_R$. Intuitively, one can think of the additional capacity in R_L and R_R as a reserve source of capacity that is used to help Algorithm 1 *fully* accept packets that are *fractionally* accepted in the linear program solution. We stress that Algorithm 1 always maintains the invariant that $S_{L,i} + S_{R,i} = M$ and $R_L + R_R = M$ for all i.

When processing each packet, say packet i which is wlog in $X_{\rightarrow}$, the algorithm first checks if there is sufficient excess capacity in R_L to accept the remaining fraction of packet i (Line 4 in Algorithm 1). If so, the packet is accepted using $(x_i - y_i)$ capacity from R_L and y_i capacity from $S_{L,i}$. The capacity of $S_{L,i}$ decreases by y_i and the capacity of $S_{R,i}$ increases by y_i, and the capacity in R_L decreases by $(x_i - y_i)$ while the capacity in R_R increases by the same amount. If there is insufficient capacity in $R_{L,i}$, i.e. $R_{L,i} < x_i - y_i$, the algorithm takes y_i from $R_{R,i}$ and adds it to $S_{R,i+1}$, and takes y_i from $S_{L,i}$ and adds it to $R_{L,i+1}$ (see Lines 12 to 15 in Algorithm 1). Note that the updates to $S_{L,i}$ and $S_{R,i}$ at each step are exactly as the solution to the linear program (Lines 8 and 9 and Lines 14 and 15).

Lemma 3. *Given the solution of the linear program, a sequence of packets X_t, and link capacity M, Algorithm 1 incurs a link capacity cost of $2M$ and accepts all packets fully accepted in the linear program.*

Proof. Wlog, let the ith packet in the sequence belong to $X_{\rightarrow}$. After processing the ith packet x_i, we denote R_L (resp. R_R) at that step as $R_{L,i}$ (resp. $R_{R,i}$). We show that the link, at time i, has capacity at least $S_{L,i}$ on the left and at least $S_{R,i}$ on the right.

When $R_{L,i}$ is large enough to accept packet x_i, we use y_i capacity from $S_{L,i}$ and $x_i - y_i$ capacity from $R_{L,i}$. The capacity y_i from the accepted packet goes to $S_{R,i+1}$ and the rest $(x_i - y_i)$ of the capacity goes to $R_{R,i+1}$.

If the packet is forced to be rejected, we know that $R_{L,i} < x_i - y_i$. Since $R_{R,i} = M - R_{L,i}$, we know that $R_{R,i} > M - x_i + y_i$, and because all packets have weight smaller than M, $R_{R,i} > y_i$ follows. This means we can take y_i from $R_{R,i}$ and add it to $S_{R,i+1}$ and remove y_i from $S_{L,i}$ (because the capacity disappeared from there) and add it to $R_{L,i+1}$.

If the packet is fully accepted, then $x_i - y_i = 0$. This means that the condition $R_{L,i} \geq x_i - y_i$ is satisfied and the algorithm accepts it. □

We conclude this example with two remarks.

Remark 1. Lemma 3 holds for any initial distribution of R_L and R_R so long as $R_L + R_R = M$.

Remark 2. Algorithm 1 is greedy and accepts all packets as long as $R_L \geq x_i - y_i$. This could be suboptimal as it might not have enough capacity in R_L to accept important packets later in the sequence. However, to maintain the condition $R_L, R_R \geq 0$ in line 4 of Algorithm 1, we can substitute the conditional check $R_L \geq x_i - y_i$ with $R_R < y_i$ at any point. Then, the proof of Lemma 3 still holds. We note that one could use this as a heuristic to develop a better approximation as it allows more fine-grained control over the greediness of the algorithm.

4. A constant approximation algorithm

Based on the insights in the previous section, we now present a $(1 + \sqrt{3})$-approximation algorithm for the weighted packet selection for a link problem with fixed capacity M. We present the formal description of the algorithm in Algorithm 2 for packets $x_i \in X_{\rightarrow}$ and omit the procedure for $x_i \in X_{\leftarrow}$ as the decision process is symmetric.

In a nutshell, Algorithm 2 consists of three main ideas: first, it uses M capacity to follow the decisions made by the linear program solution as much as possible, using an additional $\sqrt{3}M$ as reserve capacity to fully accept some packets that were fractionally accepted by the linear program. Second, Algorithm 2 tries to maintain a balance in the distribution of capacity in both ends of the link. Intuitively, any packet that is accepted by the linear program can be accepted when the algorithm is in this balanced state.

Lastly, whenever the capacity is unbalanced: one side (wlog left side) has too little capacity, the algorithm prioritises accepting packets that come from right to left as well as rejecting packets that go from left to right. This brings the capacity at both sides to the balanced state, and our analysis shows that the approximation ratio is maintained below $1+\sqrt{3}$.

Input and initial capacity distribution Algorithm 2 takes as input X_t and the solution of linear program given a fixed capacity M. Recall that $S_{L,i}$ and $S_{R,i}$ for $i \in [t]$ are the capacity distributions from the linear program solution on the left and right end of the link respectively after processing the ith packet. The algorithm uses the initial distribution $S_{L,0}$ and $S_{R,0}$, and additionally creates 2 "reserve capacity buckets" R_L and R_R of size $\frac{\sqrt{3}}{2}M$ each on both ends. Thus, the initial capacity of the left node would be $S_{L,0}+R_L$ and the initial capacity of the right node would be $S_{R,0}+R_R$. Intuitively, one can think of the additional capacity in R_L and R_R as a reserve source of capacity that is used to help Algorithm 2 *fully* accept packets that are *fractionally* accepted in the linear program solution.

Algorithm 2 accepts packets in the following way: for a packet of size x_i wlog in $X_{\rightarrow}$, assuming there is sufficient capacity in R_L, the packet is accepted using $(x_i - y_i)$ capacity from R_L and y_i capacity from $S_{L,i}$. The capacity of $S_{L,i}$ decreases by y_i and the capacity of $S_{R,i}$ increases by y_i, and the capacity in R_L decreases by $(x_i - y_i)$ while the capacity in R_R increases by the same amount. If the algorithm rejects x_i, the algorithm takes y_i from $R_{R,i}$ and adds it to $S_{R,i+1}$, and takes y_i from $S_{L,i}$ and adds it to $R_{L,i+1}$. We stress that in doing so, the algorithm always ensures that the updates to $S_{L,i}$ and $S_{R,i}$ at each step are exactly the same as the solution to the linear program. We also note that Algorithm 2 always maintains the invariant that $S_{L,i}+S_{R,i}=M$ and $R_L+R_R=\sqrt{3}M$ for all i.

We distinguish between three phases of Algorithm 2. We say the algorithm is in the *balanced phase* if both $R_L \geq \frac{\sqrt{3}-1}{2}M$ and $R_R \geq \frac{\sqrt{3}-1}{2}M$. If $R_L < \frac{\sqrt{3}-1}{2}M$, we say the algorithm is in the *left phase*, and if $R_R < \frac{\sqrt{3}-1}{2}M$, we say the algorithm is in the *right phase*. We also distinguish between 2 types of packets: little-accepted and almost-accepted packets. We say a packet is *little-accepted* if $\frac{y_i}{x_i} < \frac{\sqrt{3}}{1+\sqrt{3}}$, and *almost-accepted* if $\frac{y_i}{x_i} \geq \frac{\sqrt{3}}{1+\sqrt{3}}$.

Balanced phase In the balanced phase, Algorithm 2 accepts all packets that are almost-accepted in the linear program solution. It also accepts little-accepted packets that allow it to remain in the balanced phase. That is, for a little-accepted packet x_i wlog in $X_{\rightarrow}$, it first checks if the left reserve R_L is sufficient to forward the packet, and that doing so keeps the algorithm in the balanced phase (Line 4). If R_L does not have sufficient capacity, the algorithm rejects x_i (Line 8).

We first show in the following lemma that rejecting any little-accepted packet is safe in the sense that doing so will not push the approximation ratio of the algorithm above $1+\sqrt{3}$.

Lemma 4. *All little-accepted packets can be rejected while keeping the approximation ratio below* $1+\sqrt{3}$.

Proof. Recall that rejecting a packet x_i incurs a cost of fx_i+m. From Eq. (1), the cost of a little-accepted packet x_i for the linear program is $f\cdot(x_i-y_i)+m\frac{x_i-y_i}{x_i} \geq \frac{fx_i}{1+\sqrt{3}}+\frac{m}{1+\sqrt{3}}=\frac{1}{1+\sqrt{3}}(fx_i+m)$. From Lemma 2, we know that the solution of the linear program for a fixed capacity M is a lower bound on the optimal solution with capacity M, hence rejecting little-accepted packets will not increase the approximation ratio above $1+\sqrt{3}$. □

In the next lemma, we show that processing little-accepted packets does not affect whether the algorithm stays in the balanced phase or not. This simplifies the decision making process of Algorithm 2 as it can just focus on the decision problem for almost-accepted packets.

Lemma 5. *Algorithm 2 never leaves the balanced phase after processing a little-accepted packet.*

Proof. Each little-accepted packet moves at most $\frac{1}{1+\sqrt{3}}M$ from the left side to the right side of a link, and at most $\frac{\sqrt{3}}{1+\sqrt{3}}M$ from the right side to the left side of a link.

Because $R_L+R_R=\sqrt{3}M$ and any packet has a weight at most M, if $R_L-\frac{1}{1+\sqrt{3}}M < \frac{\sqrt{3}-1}{2}M$, then $R_R-\frac{\sqrt{3}}{1+\sqrt{3}}M \geq \frac{\sqrt{3}-1}{2}M$.

That means that rejecting a little-accepted packet from $X_{\rightarrow}$ does not create a situation where $R_R < \frac{\sqrt{3}-1}{2}M$. □

Left phase Since Algorithm 2 accepts all almost-accepted packets in the balanced phase, it would sometimes have to enter the left or right phase. Here we describe the procedure for what happens in the left phase (the right phase is analogous).

Suppose Algorithm 2 enters the left phase after processing packet x_{i-1}. The objective of Algorithm 2 in this phase is to accept all almost-accepted packets among the unprocessed packets (i.e. packets $x_i,\ldots,x_t$). If this is not possible, the algorithm rejects some of them such that both of the following conditions hold: first, the approximation ratio remains $1+\sqrt{3}$, and second, the algorithm returns to a balanced phase.

Algorithm 2 $(1+\sqrt{3})$-approximation algorithm.

Input: packet sequence X_t, capacity M, solution of LP_M:$S_{L,i}, S_{R,i}, y_i$.
Output: decisions to accept or reject

1: initialise $R_L = \frac{\sqrt{3}}{2}M$, $R_R = \frac{\sqrt{3}}{2}M$
2: **for** $i \in [t]$ **do**
3: **if** $x_i \in X_{\rightarrow}$ **then**
4: **if** $R_L - (x_i - y_i) \geq \frac{\sqrt{3}-1}{2}M$ **then**
5: **Accept**
6: $R_L = R_L - (x_i - y_i)$
7: $R_R = R_R + (x_i - y_i)$
8: **else if** x_i is little-accepted **then**
9: **Reject**
10: $R_L = R_L + y_i$
11: $R_R = R_R - y_i$
12: **else**
13: $\phi_A, \phi_R, U, R'_L, j \leftarrow$ DIVIDE(R_L, LP_M, X_t, i)
14: $U_R \leftarrow \{\}$
15: **if** $R'_L < 0$ **then**
16: $U_R, R'_L \leftarrow$ REJECTBIG(X_t, U, R'_L)
17: **Accept** all $x_i \in \phi_A \cup (U \setminus U_R)$
18: **Reject** all $x_i \in \phi_R \cup U_R$.
19: $R_L = R'_L$
20: $R_R = \sqrt{3}M - R_L$
21: $i = j$

To do so, Algorithm 2 calls a subroutine DIVIDE (Line 13 in Algorithm 2) to sort all unprocessed packets into three sets: ϕ_A, ϕ_R, U. Set ϕ_A contains all packets from $X_{\leftarrow}$. These will be accepted as they will increase the left capacity reserve R_L and help to bring Algorithm 2 back into the balanced phase. Set ϕ_R contains little-accepted packets from $X_{\rightarrow}$. These will be rejected and from Lemma 4 we know that doing so does not increase the approximation ratio. Set U contains almost-accepted packets from $X_{\rightarrow}$. Some of these packets will be accepted and some rejected in a way that maintains the approximation ratio.

DIVIDE (described in Algorithm 3) takes as input the packet sequence X_t, the solution of the linear program as well as the current capacity in the left reserve R_L. DIVIDE creates the sets ϕ_A, ϕ_R, U incrementally by processing each unprocessed packet and accepting packets from $\phi_A \cup U$ and rejecting packets from ϕ_R until one of the following stopping conditions occurs:

1. $R_L < 0$ which would mean the left capacity reserves are depleted
2. $R_L > \frac{\sqrt{3}-1}{2}M$
3. all packets are processed

If the first stopping condition is reached (Line 15 in Algorithm 2), the procedure REJECTBIG is called. REJECTBIG (described in Algorithm 4) takes as input the set U and outputs another set $U_R \subset U$. This set U_R is created by greedily selecting the biggest sized packets in U (Line 3 in Algorithm 4) and adding them to U_R. These packets will be rejected and the left capacity reserves will be accordingly updated after each rejected packet (Line 6 in Algorithm 4). The procedure REJECTBIG terminates when the left capacity reserves $R_L \geq \frac{\sqrt{3}-1}{2}$.

Now we show that if DIVIDE terminates on either the second and third stopping condition, Algorithm 2 will either be in the balanced phase (second stopping condition) or all packets will be processed and Algorithm 2 terminates (third stopping condition).

Lemma 6. *If* DIVIDE *returns* $R'_L \geq 0$ *and* j*, all almost-accepted packets between* i *and* j *are accepted by Algorithm 2 and either all packets are processed or* $R_{R,j} \geq \frac{\sqrt{3}-1}{2}M$ *and* $R_{L,j} \geq \frac{\sqrt{3}-1}{2}M$.

Proof. There are two reasons why DIVIDE returned $R'_L \geq 0$: either $R'_L \geq \frac{\sqrt{3}-1}{2}$ or $j = t$.

In both cases, we note that DIVIDE simulated accepting all packets from ϕ_A and U and rejecting all packets from ϕ_R, and at no time R'_L went below 0. That means that Algorithm 2 just repeats decisions of DIVIDE.

Finally, since $R_L + R_R = \sqrt{3}$ and all packets are smaller than M, this means that Algorithm 2 after emerging from left-phase cannot plunge to a right-phase right away. □

As the penultimate step in our analysis, we show in the next lemma that REJECTBIG does not bring the approximation ratio of Algorithm 2 over $1+\sqrt{3}$. We do this by computing the rejection cost incurred by Algorithm 2 on packets from U_R and showing that it is always lower than $(1+\sqrt{3})$ times the cost of the linear program on U. As the solution of the linear program is a lower bound on the cost of the optimal algorithm, this shows that Algorithm 2 maintains the approximation ratio even when rejecting packets from U_R.

Algorithm 3 Function DIVIDE to create sets ϕ_A, ϕ_R, and U.

Input: packet sequence X_t, solution of LP : $S_{L,i}, S_{R,i}, y_i$, value R_L, capacity M,
Output: sets ϕ_A, ϕ_R, U, resulting R_L

$R_L = R_L - (x_i - y_i)$
$\phi_A, \phi_R, U \leftarrow \{\}, \{\}, \{x_i\}$
$j = i$
while $R_L \geq 0$ and $R_L < \frac{\sqrt{3}-1}{2}$ and $j < t$ **do**
  $j = j + 1$
  if $x_j \in X_{\rightarrow}$ and x_j is almost-accepted **then**
    $R_L = R_L - (x_j - y_j)$
    $U \leftarrow U \cup x_j$
  else if $x_j \in X_{\rightarrow}$ and x_j is little-accepted **then**
    $R_L = R_L + y_j$
    $\phi_R \leftarrow \phi_R \cup x_j$
  else
    $R_L = R_L + (x_j - y_j)$
    $\phi_A \leftarrow \phi_A \cup x_j$
return $\phi_A, \phi_R, U, R_L, j$

Algorithm 4 Function REJECTBIG to prune out packets from U.

Input: packet sequence X_t, set U, value R'_L
Output: set U_R, value R'_L

$U_R \leftarrow \{\}$
while $R'_L < \frac{\sqrt{3}-1}{2}$ **do**
  $x_k \leftarrow$ biggest packet from U
  $U \leftarrow U \setminus x_k$
  $U_R \leftarrow U_R \cup \{x_k\}$
  $R'_L = R'_L + x_k$
return U_R, R'_L

Lemma 7. *In Algorithm 2, for sets U and U_R the following inequality holds:*

$$(1+\sqrt{3}) \sum_{x_i \in U} f \cdot (x_i - y_i) + m\frac{x_i - y_i}{x_i} \geq \sum_{x_i \in U_R} f x_i + m$$

Proof. If $R'_L \geq 0$, we know that all almost-accepted transactions are accepted from Lemma 6. For $R'_L < 0$ we prove that $(1+\sqrt{3}) \sum_{x_i \in U} (x_i - y_i) \geq \sum_{x_i \in U_R} x_i$, then we argue that the whole theorem holds.

Let $D = R_{L,i-1} - R'_L$ where R'_L is the value returned by DIVIDE in Algorithm 2. We know from the fact that $R_{L,i-1}$ is the amount of reserves on the left *before* the left phase and $R'_L < 0$, thus $D \geq \frac{\sqrt{3}-1}{2} M$.

By following the changes of R'_L in DIVIDE, we get

$$\sum_{x_i \in U} x_i - y_i = D + \sum_{x_i \in \phi_R} y_i + \sum_{x_i \in \phi_A} x_i - y_i$$

Therefore, we know that $\sum_{x_i \in U} x_i - y_i \geq D$.

Algorithm REJECTBIG removes transactions from U until $\sum_{x_i \in U_R} x_i \geq D$. If the condition is satisfied, we know that REJECTBIG returns U_R, because $R'_L \geq \frac{\sqrt{3}-1}{2}$.

If $|U_R| = 1$, we know that $\sum_{x_i \in U_R} x_i \leq M$, because every $x_i \leq M$. So in that case $\sum_{x_i \in U_R} x_i \leq M \leq (1+\sqrt{3})\frac{\sqrt{3}-1}{2} M \leq (1+\sqrt{3})D$.

If $|U_R| > 1$, we know that rejecting just one transaction is not enough. This means the biggest transaction has size at most D, so $\sum_{x_i \in U_R} x_i \leq 2D \leq (1+\sqrt{3})D$.

Now, we know that Algorithm 2 rejects less size than the linear program multiplied by $(1+\sqrt{3})$. This implies that $(1+\sqrt{3}) \sum_{x_i \in U} f \cdot (x_i - y_i) \geq \sum_{x_i \in U_R} f x_i$, and leaves us to prove $(1+\sqrt{3}) \sum_{x_i \in U} m\frac{x_i - y_i}{x_i} \geq \sum_{x_i \in U_R} m$.

But we know that the transactions are moved to U_R from the biggest size to smallest size. This means that for every $x_k \in U_R$ and $x_l \in U$, $x_k > x_l$, and so $\frac{x_l - y_l}{x_l} \geq \frac{x_l - y_l}{x_k}$ holds. Let x^* be the size of the smallest transaction in U_R. It suffices to show that $(1+\sqrt{3}) \sum_{x_i \in U} \frac{x_i - y_i}{x^*} \geq \sum_{x_i \in U_R} \frac{x_i}{x^*} \geq \sum_{x_i \in U_R} x_i$. However, we know that $\sum_{x_i \in U_R} x_i \leq 2D \leq (1+\sqrt{3})D \leq (1+\sqrt{3}) \sum_{x_i \in U} x_i - y_i$ since $\sum_{x_i \in U} x_i - y_i \geq D$. Thus, we conclude that $(1+\sqrt{3}) \sum_{x_i \in U} m\frac{x_i - y_i}{x_i} \geq \sum_{x_i \in U_R} m$. □

We now have all the necessary ingredients to state and prove our main theorem, which is that weighted packet selection can be approximated with an approximation ratio of $(1+\varepsilon)(1+\sqrt{3})$.

Theorem 1. *The weighted packet selection for a link problem can be approximated with a ratio* $(1+\varepsilon)(1+\sqrt{3})$ *in time* $\mathcal{O}(n^{\omega} \cdot \frac{1}{\varepsilon} \cdot \log \frac{M_{\max}}{x_{\min}})$, *where* ω *is the exponent of* n *in matrix multiplication.*

Proof. We perform a search for the capacity of the link according to Lemma 1 and for every capacity M searched we solve the linear program (as stated in Equation (1)) and run Algorithm 2. The solution is the output of Algorithm 2 with the smallest cost.

We know that $x_{\min}(1+\varepsilon)^{\frac{1}{\varepsilon}\cdot\log\frac{M_{\max}}{x_{\min}}} \geq M_{\max}$. That means we need to solve the linear program and run Algorithm 2 at most $\frac{1}{\varepsilon} \cdot \log \frac{M_{\max}}{x_{\min}}$ times.

From Lemma 2 we know that the solution of the linear program with parameter M is a lower bound for OPT^R_M.

From Lemma 3, we know that Algorithm 2 accepts all fully-accepted packets. The algorithm can reject any little-accepted packets by Lemma 4. We also know from Lemma 5 that in the balanced phase Algorithm 2 accepts all almost-accepted packets and never leaves the phase after processing little-accepted packets. Finally, Lemma 7 shows that even in a left (or right) phase the approximation ratio of Algorithm 2 on almost-accepted packets is $1+\sqrt{3}$. This means Algorithm 2 is $(1+\sqrt{3})$-approximation algorithm for the solution of the linear program. Moreover, the algorithm uses $(1+\sqrt{3})$ times more capacity that the linear program.

Using Lemma 1, we find that the selected solution is a $(1+\varepsilon)(1+\sqrt{3})$-approximation of the weighted packet selection for a link problem. □

5. Hardness

In this section, we show that weighted packet selection for a link is generally NP-hard.

Theorem 2. *Weighted packet selection for a link is NP-hard.*

Proof. We show a reduction from the subset sum problem, which is known to be NP-hard [2]. In the subset sum problem, we are given a multiset of integers $\mathcal{I} := \{i_1, i_2, \ldots, i_n\}$ and a target integer S. The goal is to find a subset of $\mathcal{I}$ with a sum of S.

Consider the following question in the weighted packet selection for a link problem: "is the cost below a given value?" We show this question is NP-hard.

We set the constants to $m = 0$ and $f = \frac{3}{4}$. We create a packet sequence consisting of $i_1, i_2, \ldots, i_n$ where the jth packet in the sequence has weight i_j which is the jth element in $\mathcal{I}$. These packets all go from left to right. Then we add a packet of weight S going from right to left.

Suppose that there exists $\mathcal{I}' \subseteq \mathcal{I}$, such that $\sum_{j\in\mathcal{I}'} i_j = S$. Then we show that the cost is at most $\frac{1}{4}S + \frac{3}{4}\sum_{j\in\mathcal{I}} i_j$.

The solution reaching that cost is as follows: nodes start with capacity S on the right and accept all packets from $\mathcal{I}'$ and then accept the last packet of weight S. The cost is then $S + \frac{3}{4}\sum_{j\in\mathcal{I}\setminus\mathcal{I}'} i_j$. Since $\sum_{j\in\mathcal{I}'} i_j = S$, the bound holds.

Now, suppose that there is no subset of $\mathcal{I}$ summing to S. Let $\mathcal{A} \subseteq \mathcal{I}$ be any set with sum A. We follow the same procedure as described above by starting with A capacity on the right and accept all packets from $\mathcal{A}$. The cost for the packets going from left to right is $A + \frac{3}{4}\sum_{j\in\mathcal{I}\setminus\mathcal{A}} i_j = \frac{1}{4}A + \frac{3}{4}\sum_{j\in\mathcal{I}} i_j$. Depending on whether $A < S$ or $A > S$, the last packet of size S can be either accepted or rejected. Thus, we need to add $\min(\max(S-A,0), \frac{3}{4}S)$ to the overall cost, which represents either the additional capacity cost of "topping up" the initial capacity of A on the right side by $S - A$ such that we can accept the last packet, or the additional cost of rejecting the last packet S, whichever is smaller. Since $A \neq S$, we know that

$$\frac{1}{4}A + \frac{3}{4}\sum_{j\in\mathcal{I}} i_j + \min(\max(S-A,0), \frac{3}{4}S) > \frac{1}{4}S + \frac{3}{4}\sum_{j\in\mathcal{I}} i_j$$

Rearranging, we get $\min(\max(S-A,0), \frac{3}{4}S) > \frac{1}{4}(S-A)$, which means that weighted packet selection for a link is NP-hard. □

6. Extensions

We highlight two natural and interesting directions to generalise our approach from a link to a network.

6.1. Cyclic redistribution of capacity to reduce cost

Suppose node u on link (u,v) is incident to ≥ 2 links (let us call one of the incident links (u,w)). From our definition of rechargeable links (see Section 1), we know it is not possible for u to increase the capacity on the (u,v) link by transferring excess capacity from (u,w). However, if (u,v) and (u,w) are part of a larger cycle in the network, u can send excess capacity from link to link in a cyclic fashion starting from the (u,w) link and ending at (u,v) while maintaining the invariant that the total capacity on each link as well as the sum of all the capacities of a node on their incident links remains the same. This can be done at any point in time without the need to transfer packets. We call this cyclic redistribution (note that this is possible on payment channel networks [12,13,5] and is known as rebalancing) and illustrate it with an example in Fig. 2. In some situations, especially if the cost of closing and recreating a link is extremely large, the possibility of cheaply shifting capacities in cycles can reduce the overall cost to nodes in the network.

Fig. 2. The graph on the left depicts three nodes u, v, w connected in a cycle. The numbers by each link close to a node represent the capacity of a node in a certain link. u can increase their capacity by 10 on the (u, v) link by first sending the excess capacity of 10 to w along the (u, w) link. Then w sends the excess capacity of 10 to v along (w, v). Finally, v sends capacity of 10 back to u along (v, u). The graph on the right depicts the updated capacities of each node on each link after cyclic redistribution.

Let us denote the cost of decreasing capacities by x on the right and increasing it by x on the left using cyclic redistribution by $C(fx + m)$ for some $C \geq 1$ (one can view C as a function of the length of the cycle one sends the capacities along).

Here, we sketch an approximation algorithm that solves the weighted packet selection for a link problem with the possibility of cyclic redistribution. Note that our sketch is not precise, we simply modify Algorithm 2 where we assume the constants are already optimised for the basic problem.

We modify the linear program by adding variables $o_i, i \in [t]$ with constraints $0 \leq o_i \leq M$. The variable o_i denotes the capacity that was shifted from one side to the other before the algorithm processes packet x_i. We also modify the capacity constraints in the following way (for the case where $x_{i-1} \in X_{\leftarrow}$ and $x_i \in X_{\leftarrow}$): $S_{L,i} = S_{L,i-1} - o_i + y_{i-1}$ and $S_{R,i} = S_{R,i-1} + o_i - y_{i-1}$. We change the signs of variables for the other cases. Finally, we add $\sum_i C(f o_i + \frac{o_i}{M} m)$ to the objective in Eq. (1).

We divide the algorithm into epochs. We sum all o_i in the current epoch. If the sum is above $\frac{1}{1+\sqrt{3}} M$ we perform cyclic distribution if needed and start a new epoch. Note that in the current epoch, the optimal algorithm already paid at least $Cf\frac{M+m}{1+\sqrt{3}}$, so, we can move M capacity, incurring a cost at most $1 + \sqrt{3}$ times bigger than the optimal algorithm for cyclic redistribution.

To deal with capacity changes inside each epoch, we increase R_L and R_R. We initialise them in a way that they absorb changes of capacity in the first epoch of our algorithm. After an epoch, we reset them by cyclic redistribution such that they absorb changes of capacity in the next. The increase in R_L and R_R is at most $\frac{1}{1+\sqrt{3}} M$. These changes increase the approximation ratio of our algorithm from $1 + \sqrt{3}$ to $1 + \sqrt{3} + \frac{1}{1+\sqrt{3}}$, which is $\frac{1+3\sqrt{3}}{2}$.

6.2. Going from a link to a general network

Here we show how to extend the weighted packet selection problem from a single link to a general network. We begin by describing the problem for a general network.

Weighted packet selection for general graph The input to the problem is a general graph $G = (V, E)$ where each link in the graph is rechargeable, and an ordered sequence of packet requests $X_t = ((x_1, p_1), \ldots (x_t, p_t))$. $x_i \in \mathbb{R}^+$ denotes the weight of the ith packet and p_i represents the directed path through the graph that the ith packet needs to be routed though. The actions and costs per node are the same as described in Section 1 for the weighted packet selection for a link problem. As in the case of the problem confined to a single link, the goal in this setting is also to optimise over the entire graph. That is, the goal is for involved nodes to collaboratively decide on the initial capacity and distribution for all involved links as well as which packets to accept or to reject, so as to minimise the *total* rejection and capacity costs.

Solution where there are few long paths We now present a simple solution in the case where the input only contains a few packets that have to be routed over multiple links. We first observe that if a packet has to be routed through a path with length > 1, the packet has to be accepted or rejected by *all* the links in its routing path. Let us call a packet *long* if it needs to pass through more than one link.

Thus far, we showed an approximation algorithm that solves the problem if all packets only go through a single link. Suppose we are given the situation where we can bound the number of long packets, say by ℓ. Given a network and packet sequence for which we know only ℓ are long, we can approximate the extended problem with approximation ratio $(1 + \varepsilon)(1 + \sqrt{3})$ in time 2^{ℓ} times the time needed for the problem confined to a single link by simply trying to accept all subsets of paths of long packets. If the packet is accepted or rejected, we can reflect it in the linear program by requiring $y_i = x_i$. Then Algorithm 2 surely accepts this packet and the condition that the packet needs to be accepted by all links it passes through is satisfied.

Heuristic for general graphs We now describe a heuristic for the general case where the input sequence may contain many packets that have to be routed over multiple links. In particular, some links can be used in more than 1 routing path.

The idea of the algorithm for the general case is as follows:

- We create and solve a linear program similar to the one defined in Section 3.2. Compared to the linear program on a single link, this new linear program returns how much a packet should be accepted (the fractional solution, output of the linear program) on the whole path.

– Having the linear program solution for the whole graph, we look only at decisions inside a single (arbitrary) link ℓ and use Algorithm 2. This gives us decisions for packets going through ℓ.
– Respecting the decisions on ℓ, we solve the problem recursively (link ℓ is removed).

Our approach does not give any theoretical guarantees, only that the decisions on a link ℓ are a $(1 + \sqrt{3})$- approximation while respecting the previous decisions. We believe this opens an exciting avenue for future work.

7. Conclusion

We initiated the study of weighted packet selection over a rechargable capacitated link, a natural algorithmic problem e.g., describing the routing of financial transactions in cryptocurrency networks. We showed that this problem is NP-hard and provided a constant factor approximation algorithm.

We understand our work is a first step, and believe that it opens several interesting avenues for future research. In particular, it remains to find a matching lower bound for the achievable approximation ratio, and to study the performance of our algorithm in practice. More generally, it would be interesting to deepen the study of the relationship between the current weighted packet selection problem, and the online version of the problem, a direction proposed by [6], and also and explore competitive algorithms. This version of the problem, when extended to a network, can be seen as a novel version of the classic online call admission problem [3].

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Jakub Svoboda reports financial support was provided by European Research Council. Stefan Schmid reports financial support was provided by European Research Council. Stefan Schmid reports financial support was provided by German Research Foundation.

Acknowledgements

We thank Mahsa Bastankhah and Mohammad Ali Maddah-Ali for fruitful discussions about different variants of the problem. This work is supported by the European Research Council (ERC) Consolidator Project 864228 (AdjustNet), 2020-2025, the ERC CoG 863818 (ForM-SMArt), and the German Research Foundation (DFG) grant 470029389 (FlexNets), 2021-2024.

References

[1] Raiden network, https://raiden.network/, 2017.
[2] S. Arora, B. Barak, Computational Complexity: A Modern Approach, 1st edn., Cambridge University Press, USA, 2009.
[3] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, O. Waarts, On-line routing of virtual circuits with applications to load balancing and machine scheduling, J. ACM 44 (3) (1997) 486–504.
[4] Z. Avarikioti, T. Lizurej, T. Michalak, M. Yeo, Lightning creation games, CoRR, arXiv:2306.16006 [abs], 2023, https://doi.org/10.48550/arXiv.2306.16006.
[5] Z. Avarikioti, K. Pietrzak, I. Salem, S. Schmid, S. Tiwari, M. Yeo, Hide and seek: privacy-preserving rebalancing on payment channel networks, in: Proc. Financial Cryptography and Data Security (FC), 2022.
[6] M. Bastankhah, K. Chatterjee, M.A. Maddah-Ali, S. Schmid, J. Svoboda, M. Yeo, Online admission control and rebalancing in payment channel networks, CoRR, arXiv:2209.11936 [abs], 2022, https://doi.org/10.48550/arXiv.2209.11936.
[7] C. Chekuri, S. Khanna, F.B. Shepherd, The all-or-nothing multicommodity flow problem, in: Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), 2004, pp. 156–165.
[8] M.B. Cohen, Y.T. Lee, Z. Song, Solving linear programs in the current matrix multiplication time, J. ACM 68 (1) (2021) 3, https://doi.org/10.1145/3424305.
[9] C. Decker, R. Wattenhofer, A fast and scalable payment network with bitcoin duplex micropayment channels, in: Symposium on Self-Stabilizing Systems (SSS), Springer, 2015, pp. 3–18.
[10] M. Dotan, Y.A. Pignolet, S. Schmid, S. Tochner, A. Zohar, Survey on blockchain networking: context, state-of-the-art, challenges, in: Proc. ACM Computing Surveys (CSUR), 2021.
[11] P.K. Gupta, P.R. Kumar, The capacity of wireless networks, IEEE Trans. Inf. Theory 46 (2000) 388–404.
[12] R. Khalil, A. Gervais, Revive: rebalancing off-blockchain payment networks, in: B.M. Thuraisingham, D. Evans, T. Malkin, D. Xu (Eds.), Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017, ACM, 2017, pp. 439–453.
[13] R. Pickhardt, M. 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, IEEE, 2020, pp. 1–5.
[14] J. Poon, T. Dryja, The bitcoin lightning network: scalable off-chain instant payments, https://lightning.network/lightning-network-paper.pdf, 2015.
[15] P. Raghavan, C.D. Thompson, Provably good routing in graphs: regular arrays, in: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, 1985, pp. 79–87.