Weighted Packet Selection for Rechargeable Links

Stefan Schmid   Jakub Svoboda   Michelle Yeo

Motivation

Problem

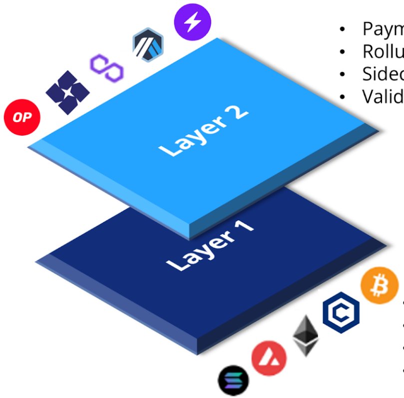
Sending transactions in blockchain is expensive (consensus, security).

Motivation

Problem

Sending transactions in blockchain is expensive (consensus, security).

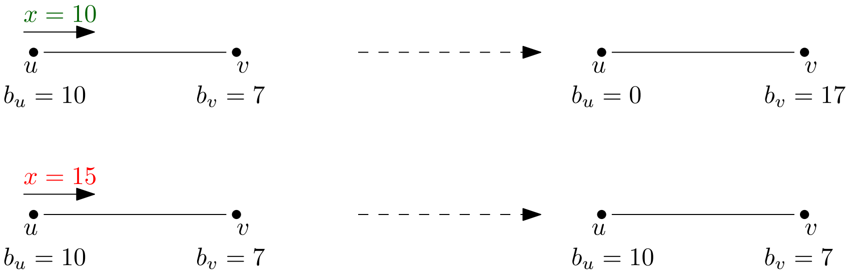
Methods are either Layer 1 or Layer 2 according to their focus (i.e., On-Chain or Off-Chain)



- Payment Channels
- Rollups
- Sidechains
- Validium

- Block Reparameterisation
- Sharding
- Consensus Mechanism
- DAG

Payment channels

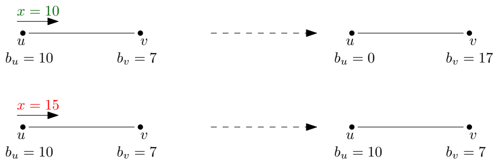


Goal

Create efficient algorithm for a link (two nodes) in PCN that cooperate, but do not trust each other.

Actions for link in PCNs

Action	Capacity/Size	Cost
Create link	c	c
Forward transaction	x	0^*
Reject transaction	x	$fx + m$



Definitions

Cost

Channel creation + Cost for rejection.

Definitions

Cost

Channel creation + Cost for rejection.

Problem

Given sequence of transactions $(\rightarrow, 10), (\leftarrow, 5), (\rightarrow, 3), \ldots$, find the solution of minimal cost.

Definitions

Cost

Channel creation + Cost for rejection.

Problem

Given sequence of transactions $(\rightarrow, 10), (\leftarrow, 5), (\rightarrow, 3), \ldots$, find the solution of minimal cost.

Solution

Initial capacity of the channel and which transactions to reject.

Outline of the talk

NP-hardness

Algorithm

- Linear program
- Approximating the channel capacity
- Tracking the linear program

Hardness

Problem is NP-hard

For set of numbers $x_1, x_2, \ldots$ and X, subset sum problem, we create transactions $(\rightarrow, x_1), (\rightarrow, x_2), \ldots, (\leftarrow, X)$. Problem is yes instance if the optimal solution has small cost.

Linear program

Constants: Fixed capacity M, Input $(\leftarrow, x_i)$.

Variables:

- amount of i-th transaction accepted y_i.
- balance on the left (right) after processing i-th transaction $S_{L,i}$ ($S_{R,i}$).

Linear program

Constants: Fixed capacity M, Input $(\leftarrow, x_i)$.

Variables:

- amount of i-th transaction accepted y_i.
- balance on the left (right) after processing i-th transaction $S_{L,i}$ ($S_{R,i}$).

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

Optimal capacity

If optimal capacity is M', we have

$$M' + reject_{M'}$$

Optimal capacity

If optimal capacity is M', we have

$$M' + reject_{M'}$$

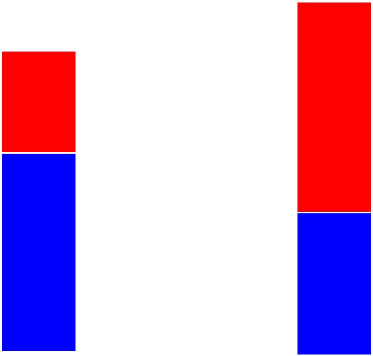
We try all capacities M of the form $(1+\varepsilon)^k$.

$$\min\left(\frac{M}{1+\varepsilon} + reject_M\right)$$

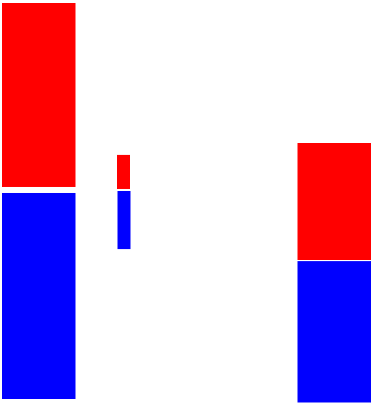
Accepting fully accepted transaction

Transaction where $y_i = x_i$ is *fully accepted*.

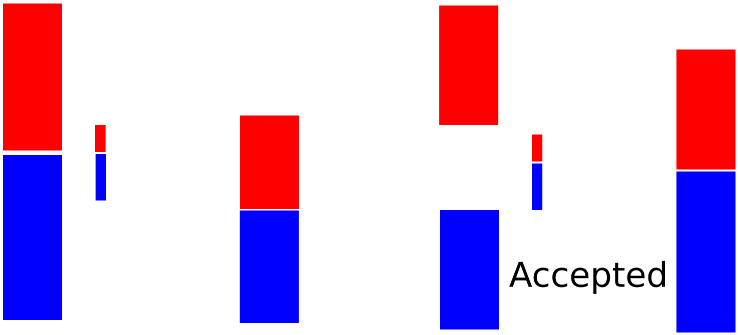
We track S_L and S_R, we add reserves R_L and R_R, such that $R_L + R_R = M$.



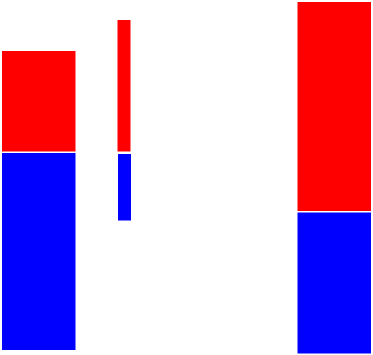
Accepting fully accepted transactions



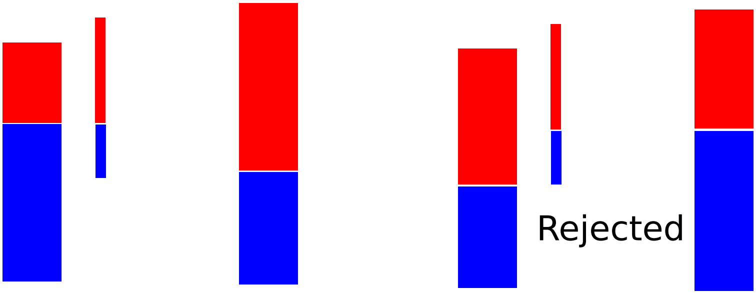
Accepting fully accepted transactions



Accepting fully accepted transactions



Accepting fully accepted transactions



$R_L < x_i - y_i$
$R_R \geq y_i$

cannot accept
$x_i < M; R_L + R_R = M$

Main idea of the algorithm

Transactions are **almost-accepted** if

$$\frac{y_i}{x_i} \geq \frac{\sqrt{3}}{\sqrt{3}+1}\,.$$

Main idea of the algorithm

Transactions are **almost-accepted** if

$$\frac{y_i}{x_i} \geq \frac{\sqrt{3}}{\sqrt{3}+1} \, .$$

Similarly as before, we can increase $R_L + R_R = \sqrt{3}M$ and accept **almost-accepted** transactions.

Main idea of the algorithm

Transactions are **almost-accepted** if

$$\frac{y_i}{x_i} \geq \frac{\sqrt{3}}{\sqrt{3}+1}\,.$$

Similarly as before, we can increase $R_L + R_R = \sqrt{3}M$ and accept **almost-accepted** transactions. This gives $(1+\sqrt{3})(1+\varepsilon)$ algorithm.

Summary

Problem definition

Hardness

$(1+\sqrt{3})(1+\varepsilon)$-approximation algorithm