Trust Is Risk: Introducing a decentralized platform for financial trust

Orfeas Stefanos Thyfronitis Litos
National Technical University of Athens

Dionysis Zindros
University of Athens

1 Abstract

Reputation in centralized systems typically uses stars and review-based trust. These systems require extensive manual intervention and secrecy to avoid manipulation. In decentralized systems this luxury is not available as the reputation system should be autonomous and open source. Previous peer-to-peer reputation systems define trust abstractly and do not allow for financial arguments pertaining to reputation. We propose a concrete sybil-resilient decentralized reputation system in which direct trust is defined as lines-of-credit using bitcoin's 1-of-2 multisig. We introduce a new model for bitcoin wallets in which user coins are split among trusted friends. Indirect trust is subsequently defined using a transitive property. This enables formal game theoretic arguments pertaining to risk analysis. Using our reputation model, we prove that risk and max flows are equivalent and propose several algorithms for the redistribution of trust so that a decision can be made on whether an anonymous third party can be indirectly trusted. In such a setting, the risk incurred by making a purchase from an anonymous vendor remains invariant. Finally, we prove the correctness of our algorithms and provide optimality arguments for various norms.

2 Introduction

3 Tags/Keywords

decentralized, trust, web-of-trust, bitcoin, multisig, line-of-credit, trust-as-risk, flow

4 Related Work

5 Key points

6 Definitions

Definition 6.1 (Players)**.**
The set $\mathcal{M} = V(G)$ is the set of all players in the network, otherwise understood as the set of all pseudonymous identities.

Definition 6.2 (Capital of A, Cap_A)**.**
Total amount of value that exists in p2pkh in the utxo and can be spent by A.

Definition 6.3 (Direct Trust from A to B, $DTr_{A\to B}$)**.**
Total amount of value that exists in $1/\{A, B\}$ multisigs in the utxo, where the money is deposited by A.

Definition 6.4 ((In/Out) Neighbourhood of A, $N^+(A), N^-(A), N(A)$)**.**

1. Let $N^+(A)$ be the set of players B that A directly trusts with any positive value. More formally, $N^+(A) = \{B \in \mathcal{M} : DTr_{A\to B} > 0\}$. $N^+(A)$ is called out neighbourhood of A.
2. Let $N^-(A)$ be the set of players B that directly trust A with any positive value. More formally, $N^-(A) = \{B \in \mathcal{M} : DTr_{B\to A} > 0\}$. $N^-(A)$ is called in neighbourhood of A.
3. Let $N(A)$ be the set of players B that either directly trust or are directly trusted by A with any positive value. More formally, $N(A) = N^+(A) \cup N^-(A)$. $N(A)$ is called neighbourhood of A.
4. Let $N(A)_i$ (respectively $N^+(A)_i, N^-(A)_i$) be the i-th element of set $N(A)$ (respectively of $N^+(A), N^-(A)$), according to an arbitrary but constant enumeration of the set players.

Definition 6.5 (B steals x from A)**.**
B steals value x from A when B reduces the $DTr_{A\to B}$ by x and increases Cap_B by x. This makes sense when $x \leq DTr_{A\to B}$.

Definition 6.6 (Turns)**.**
The game we are describing is turn-based. Let $DTr_{B\to A,j}$ be B's direct trust to A in turn j. In each turn j exactly one player $A \in \mathcal{M}$ chooses an action (according to a certain strategy or at random) that can be one of the following, or a finite combination thereof:

1. Do nothing ($Pass()$).
2. Steal value $y_B, 0 \leq y_B \leq DTr_{B\to A,j-1}$ from $B \in N^-(A)$. $DTr_{B\to A,j} = DTr_{B\to A,j-1} - y_B$. ($Steal(y_B, B)$)
3. Add value $y_B, -DTr_{A\to B,j-1} \leq y_B$ to $B \in N^+(A)$. $DTr_{A\to B,j} = DTr_{A\to B,j-1} + y_B$. When $y_B < 0$, we say that A reduces her trust to B by $-y_B$, when $y_B > 0$, we say that A increases her trust to B by y_B. ($Add(y_B, B)$)
4. Add value y_B to $B \in \mathcal{M} \setminus N^+(A)$. Obviously $DTr_{A\to B,j-1} = 0, DTr_{A\to B,j} = y_B$. We say that A starts directly trusting player B by y_B. ($AddNew(y_B, B)$)

If player A chooses to pass her turn, she may not choose any additional action in the same turn. Moreover, player A is not allowed to choose two actions of the same kind against the same player in the same turn. For example, A cannot steal $y_{B,1} \leq DTr_{B\to A,j-1}$ and $y_{B,2} \leq DTr_{B\to A,j-1}$ from $B \in N^-(A)$ in the same turn, but can instead steal $y_{B,1} + y_{B,2}$ given that $y_{B,1} + y_{B,2} \leq DTr_{B\to A,j-1}$. Also A is allowed to steal $y_B \leq DTr_{B\to A,j-1}$ from B and start directly trusting the same player B by w_B given that $B \in N^-(A) \wedge B \notin N^+(A)$.
The set of actions a player makes in turn j is $Turn_j$. Examples:

- $Turn_{j_1} = \{Pass()\}$
- $Turn_{j_2} = \{Steal(y, B), AddNew(w, B)\}$ (given that $DTr_{B\to A,j_2-1} \leq y, DTr_{A\to B,j_2-1} = 0$, where this is A's turn)
- $Turn_{j_3} = \{Steal(x, B), Add(y, C), AddNew(w, D)\}$ (given that $DTr_{B\to A,j_3-1} \leq x, DTr_{A\to C,j_3-1} \geq -y, DTr_{A\to D,j_3-1} = 0$, where this is A's turn)

Definition 6.7 (A is stolen x)**.**
Let j, j' be two consecutive turns of A. We say that A has been stolen a value x between j and j' if $\sum_{i=1}^{|N^+(A)|} DTr_{A\to N^+(A)_i,j} - \sum_{i=1}^{|N^+(A)|} DTr_{A\to N^+(A)_i,j'} = x > 0$. If turns are not specified, we implicitly refer to the current and the previous turns.

Definition 6.8 (History)**.**
We define History, $\mathcal{H}$, as the sequence of all the sets of actions. $\mathcal{H}_j = Turn_j$.

Definition 6.9 (Honest strategy)**.**
A player A is said to follow the honest strategy if for any value x that has been stolen from her since the previous turn she played, she substitutes it in her turn by stealing from others that trust her value equal to $\min(x, \sum_{B\in\mathcal{M}} DTr_{B\to A})$ and she takes no other action.

Definition 6.10 (Idle strategy)**.**
A player A is said to follow the idle strategy if she passes in her turn.

Definition 6.11 (Evil strategy)**.**
A player A is said to follow the evil strategy if she steals value $y_B = DTr_{B\to A,i-1} \,\forall\, B \in N^-(A)$ (steals all incoming direct trust) and reduces her trust to C by $DTr_{A\to C,i-1} \forall C \in N^+(A)$ (nullifies her outgoing direct trust) in her turn.

Definition 6.12 (Indirect trust from A to B $Tr_{A\to B}$)**.**
Maximum possible value that can be stolen from A if B follows the evil strategy, A follows the idle strategy and everyone else ($\mathcal{M} \setminus \{A, B\}$) follows the honest strategy.

Definition 6.13 (Trust Reduction)**.**
Let $A, B \in \mathcal{M}, x_i$ flow to $N^+(A)_i$ resulting from $maxFlow(A, B), u_i$ current $DTr_{A\to N^+(A)_i}$, u'_i new $DTr_{A\to N^+(A)_i}, i \in \{1, ..., |N^+(A)|\}$.

1. The Trust Reduction on neighbour i, δ_i is defined as $\delta_i = u_i - u'_i$.
2. The Flow Reduction on neigbour i, Δ_i is defined as $\Delta_i = x_i - u'_i$.

We will also use the standard notation for 1-norm and ∞-norm, that is:

1. $||\delta_i||_1 = \sum_{i\in N^+(A)} \delta_i$
2. $||\delta_i||_\infty = \max_{i\in N^+(A)} \delta_i$.

Definition 6.14 (Restricted Flow)**.**
Let $A, B \in \mathcal{M}, i \in \{1, ..., |N^+(A)|\}$.

1. Let $F_{A_i\to B}$ be the flow from A to $N^+(A)_i$ as calculated by the $maxFlow(A, B)$ (x'_i) when $u'_i = u_i$, $u'_j = 0 \,\forall j \in \{1, ..., |N^+(A)|\} \wedge j \neq i$.
2. Let $S \subset N^+(A)$. Let $F_{A_S\to B}$ be the sum of flows from A to S as calculated by the $maxFlow(A, B)$ $(\sum_{i=1}^{|S|} x'_i)$ when $u'_C = u_C \,\forall C \in S, u'_D = 0 \,\forall D \in N^+(A) \setminus S$.

7 Theorems-Algorithms

Theorem 7.1 (Saturation theorem)**.**
Let s source, $n = |N^+(s)|, x_i, i \in [n]$, flows to s's neighbours as calculated by the maxFlow algorithm, u'_i new direct trusts to the n neighbours and x'_i new flows to the neighbours as calculated by the maxFlow algorithm with the new direct trusts, u'_i. It holds that $\forall i \in [n], u'_i \leq x_i \Rightarrow x'_i = u'_i$.

Proof. $\forall i \in [n], x'_i > u'_i$ is impossible because a flow cannot be higher than its corresponding capacity. Thus $\forall i \in [n], x'_i \leq u'_i$. (1)
In the initial configuration of u_i and according to the flow problem setting, a combination of flows y_i such that $\forall i \in [n], y_i = u'_i$ is a valid, albeit not necessarily maximum, configuration with a flow $\sum_{i=1}^{n} y_i$. Suppose that $\exists j \in [n] : x'_j < u'_j$ as calculated by the maxFlow algorithm with the new direct trusts, u'_i. Then for the new maxFlow F' it holds that $F' = \sum_{i=1}^{n} x'_i < \sum_{i=1}^{n} y_i$ since $x'_j < y_j$ which is impossible because the configuration $\forall i \in [n], x'_i = y_i$ is valid since and (1) $\forall i \in [n], y_i = u'_i$ and also has a higher flow, thus the maxFlow algorithm will prefer the configuration with the higher flow. Thus we deduce that $\forall i \in [n], x'_i \geq u'_i$. From (1) and (2) we conclude that $\forall i \in [n], x'_i = u'_i$. □

Theorem 7.2 (Trust flow theorem - TOCHECK)**.**
$Tr_{A\to B} = MaxFlow_{A\to B}$ (Treating trusts as capacities)

Proof.

1. We will show that $Tr_{A\to B} \leq MaxFlow_{A\to B}$. We know that $MaxFlow_{A\to B} = MinCut_{A\to B}$. We will show that, if everybody except A and B follows the honest strategy, $Tr_{A\to B} \leq MinCut_{A\to B}$. Suppose that in round i all the members of the MinCut, P, have stolen the maximum value they can from members that belong in the MaxFlow graph and nobody in the partition in which A belongs has stolen yet any value. Let the total stolen value from the MinCut members be St. It is obvious

that $St_i \leq MinCut_{A\to B}$, because otherwise there would exist $u \in P$ that doesn't follow the honest strategy, since they stole more than they were stolen from. The same argument holds for any round $i' > i$ because in each round an honest player can steal only up to the value she has been stolen. It is also impossible that the St increase further due to stolen value from members of the partition of B since members of P disconnect the two partitions and have already played their turns, thus $\forall i' > i, St_{i'} \leq St_i$. There exists a round, k, when all the honest players stop stealing, so in the worst case A will have been stolen $Tr_{A\to B} = St_k \leq MinCut_{A\to B} = MaxFlow_{A\to B}$.

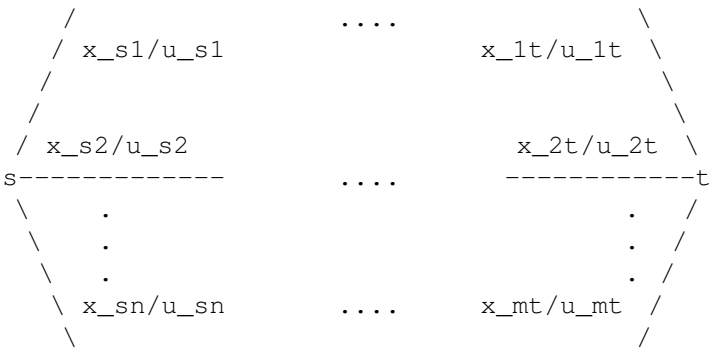
2. We can see that $Tr_{A\to B} \geq MaxFlow_{A\to B}$ because the strategy where each one of the non-idle players steals value equal to the incoming flows from their respective friends is a valid strategy that does not contradict with the honest strategy, since for every honest player w it holds that $\sum_{v\in N^-(w)} x_{vw} = \sum_{v\in N^+(w)} x_{wv}$ and according to the strategy each honest player will have been stolen value equal to $\sum_{v\in N^+(w)} x_{wv}$.

Combining the two results, we see that $Tr_{A\to B} = MaxFlow_{A\to B}$. □

Theorem 7.3 (Honest world theorem)**.**
If everybody follows the honest strategy, nobody steals any amount from anybody.

Proof. Suppose that there exists a subseries of History comprised by stealing actions represented by a vector where $action_i$ ="player i steals value V > 0 from player $i+1$". This vector must have an initial element, $action_1$. However, player 1 follows the honest strategy, thus somebody must have stolen from her as well, so player 1 cannot be the initial element. We have a contradiction, thus there cannot exist a series of stealing actions when everybody is honest. □

Theorem 7.4 (Trust transfer theorem (flow terminology) - TOCHECK)**.**
Let s source, t sink, $n = N^+(s)$
$X = \{x_1, ..., x_n\}$ outgoing flows from s,
$U = \{u_1, ..., u_n\}$ outgoing capacities from s,
V the value to be transferred.
Nodes apart from s, t follow the honest strategy.
Obviously $maxFlow = F = \sum_{i=1}^{n} x_i$.



We create a new graph where

1. $\sum_{i=1}^{n} u'_i = F - V$
2. $\forall i \in [n]\ u'_i \leq x_i$

It holds that $maxFlow' = F' = F - V$.

Proof. From theorem 7.1 we can see that $x'_i = u'_i$. It holds that $F' = \sum_{i=1}^{n} x'_i = \sum_{i=1}^{n} u'_i = F - V$. □

Corollary 7.1 (Requirement for $\sum_{i=1}^{n} u'_{s,i} = F - V$, $u'_{s,i} \leq x_{s,i}$)**.**

In the setting of 7.4, it is impossible to have $maxFlow' = F - V$ *if* $\sum_{i=1}^{n} u'_{s,i} > F - V \wedge \forall i \in [n], u'_{s,i} \leq x_{s,i}$.

Proof. Due to 7.4, $maxFlow' = F - V$ if $\sum_{i=1}^{n} u'_{s,i} = F - V \wedge \forall i \in [n], u'_{s,i} \leq x_{s,i}$. If we create new capacities such that $\forall i \in [n], u''_{s,i} \leq x_{s,i}$, then obviously $maxFlow'' = \sum_{i=1}^{n} u''_{s,i}$. If additionally $\sum_{i=1}^{n} u''_{s,i} > F - V$, then $maxFlow'' > F - V$. □

Theorem 7.5 (Trust-saving Theorem)**.**
A configuration $U' : u'_i = F_{A_i \to B}$ *for some* $i \in [|N^+(A)|]$ *can yield the same maxFlow with a configuration* $U'' : u''_i = u_i, \forall j \in [|N^+(A)|], j \neq i u''_j = u_j$.

Proof. We know that $x_i \leq F_{A_i \to B}$, thus we can see that any increase in u'_i beyond $F_{A_i \to B}$ will not influence x_i and subsequently will not incur any change on the rest of the flows. □

Theorem 7.6 (Invariable trust reduction with naive algorithms)**.**
Let A source, $n = |N^+(A)|$ *and* u'_i *new direct trusts. If* $\forall i \in [n], u'_i \leq x_i$*, Trust Reduction* $||\delta_i||_1$ *is independent of* x_i, u'_i $\forall$ *valid configurations of* x_i

Proof. Since $\forall i \in [n], u'_i \leq x_i$ it is (according to 7.1) $x'_i = u'_i$, thus $\delta_i = u_i - x'_i$. We know that $\sum_{i=1}^{n} x'_i = F - V$, so we have $||\delta_i||_1 = \sum_{i=1}^{n} \delta_i = \sum_{i=1}^{n} (u_i - x'_i) = \sum_{i=1}^{n} u_i - F + V$ independent from x'_i, u'_i □

Theorem 7.7 (Dependence impossibility theorem)**.**
$\frac{\partial x_j}{\partial x_i} = 0$ *with* x_i *the flow from MaxFlow* $\Rightarrow \forall x'_i \leq x_i, \frac{\partial x_j}{\partial x_i} = 0$ *ceteris paribus*

Proof. TODO □

One of the primary aims of this system is to mitigate the danger for sybil attacks whilst maintaining fully decentralized autonomy. Let Eve be a possible attacker. Since participation in the network does not require any kind of registration, Eve can create any number of players. We will call the set of these players $\mathcal{C}$. Moreover, Eve can invest any amount she chooses, thus she can arbitrarily set the direct trusts of any player $C \in \mathcal{C}$ to any player $P \in \mathcal{M}$ ($DTr_{C \to P}$). Moreover we give Eve a set of players $B \in \mathcal{B}$ that she has corrupted, so she fully controls their direct trusts to any player $P \in \mathcal{M}$ ($DTr_{B \to P}$). The players $B \in \mathcal{B}$ are considered to be legitimate before the corruption, thus they can be directly trusted by any player $P \in \mathcal{M}$ ($DTr_{P \to B} \geq 0$). However, players $C \in \mathcal{C}$ can be trusted only by players $D \in \mathcal{B} \cup \mathcal{C}$ ($DTr_{D \to C} \geq 0$) and not by players $A \in \mathcal{M} \setminus (\mathcal{B} \cup \mathcal{C})$ ($DTr_{A \to C} = 0$).

Theorem 7.8 (Sybil resistance)**.**

Proof. TODO □

Here we show three naive algorithms for calculating new direct trusts so as to maintain invariable risk when paying a trusted party. To prove the correctness of the algorithms, it suffices to prove that $\forall i \in [n]$ $u'_i \leq x_i$ and that $\sum_{i=1}^{n} u'_i = F - V$ where $F = \sum_{i=1}^{n} x_i$.

Algorithm 1: First-come, first-served trust transfer

Input : x_i flows, $n = |N^+(s)|$, V value
Output: u'_i capacities

1 $F \leftarrow \sum_{i=1}^{n} x_i$
2 **if** $F < V$ **then**
3 | **return** $\perp$
4 $F_{cur} \leftarrow F$
5 **for** $i \leftarrow 1$ *to* n **do**
6 | $u'_i \leftarrow x_i$
7 $i \leftarrow 1$
8 **while** $F_{cur} > F - V$ **do**
9 | $reduce \leftarrow \min(x_i, F_{cur} - F + V)$
10 | $F_{cur} \leftarrow F_{cur} - reduce$
11 | $u'_i \leftarrow x_i - reduce$
12 | $i \leftarrow i + 1$
13 **return** $U' = \bigcup_{j=1}^{n} \{u'_j\}$

Proof of correctness for algorithm 1.

- We will show that $\forall i \in [n]\ u'_i \leq x_i$.
  Let $i \in [n]$. In line 6 we can see that $u'_i = x_i$ and the only other occurence of u'_i is in line 11 where it is never increased ($reduce \geq 0$), thus we see that, when returned, $u'_i \leq x_i$.

- We will show that $\sum_{i=1}^{n} u'_i = F - V$.
  $F_{cur,0} = F$
  If $F_{cur,i} \geq F - V$, then $F_{cur,i+1}$ does not exist because the *while* loop breaks after calculating $F_{cur,i}$.
  Else $F_{cur,i+1} = F_{cur,i} - \min(x_{i+1}, F_{cur,i} - F + V)$.
  If for some $i, \min(x_{i+1}, F_{cur,i} - F + V) = F_{cur,i} - F + V$, then $F_{cur,i+1} = F - V$, so if $F_{cur,i+1}$ exists, then $\forall k < i, F_{cur,k} = F_{cur,k-1} - x_k \Rightarrow F_{cur,i} = F - \sum_{j=1}^{i} x_j$
  Furthermore, if $F_{cur,i+1} = F - V$ then $u'_{i+1} = x_{i+1} - F_{cur,i} + F - V = x_i - F + \sum_{j=1}^{i-1} x_j + F - V = \sum_{j=1}^{i} x_j - V$, $\forall k \leq i, u'_k = 0$ and $\forall k > i+1, u'_k = x_k$.
  In total, we have $\sum_{j=1}^{n} u'_j = \sum_{j=1}^{i} x_j - V + \sum_{j=i+1}^{n} x_j = \sum_{j=1}^{n} x_j - V \Rightarrow \sum_{j=1}^{n} u'_j = F - V$.

□

Complexity of algorithm 1.
First we will prove that on line 13 $i \leq n + 1$. Suppose that $i > n + 1$ on line 13. This means that $F_{cur,n}$ exists and $F_{cur,n} = F - \sum_{i=1}^{n} x_i = 0 \leq F - V$ since, according to the condition on line 2, $F - V \geq 0$. This means however that the *while* loop on line 8 will break, thus $F_{cur,n+1}$ cannot exist and $i = n + 1$ on line 13, which is a contradiction, thus $i \leq n + 1$ on line 13. Since i is incremented by 1 on every iteration of the *while* loop (line 12), the complexity of the *while* loop is $O(n)$ in the worst case. The complexity of lines 2-4 and 7 is $O(1)$ and the complexity of lines 1, 5-6 and 13 is $O(n)$, thus the total complexity of algorithm 1 is $O(n)$. □

Algorithm 2: Absolute equality trust transfer ($||\Delta_i||_\infty$ minimizer)

Input : x_i flows, $n = |N^+(s)|$, V value
Output: u'_i capacities

1 $F \leftarrow \sum_{i=1}^{n} x_i$
2 **if** $F < V$ **then**
3 **return** $\perp$
4 **for** $i \leftarrow 1$ *to* n **do**
5 $u'_i \leftarrow x_i$
6 $reduce \leftarrow \frac{V}{n}$
7 $reduction \leftarrow 0$
8 $empty \leftarrow 0$
9 $i \leftarrow 0$
10 **while** $reduction < V$ **do**
11 **if** $u'_i > 0$ **then**
12 **if** $x_i < reduce$ **then**
13 $empty \leftarrow empty + 1$
14 **if** $empty < n$ **then**
15 $reduce \leftarrow reduce + \frac{reduce - x_i}{n - empty}$
16 $reduction \leftarrow reduction + u'_i$
17 $u'_i \leftarrow 0$
18 **else if** $x_i \geq reduce$ **then**
19 $reduction \leftarrow reduction + u'_i - (x_i - reduce)$
20 $u'_i \leftarrow x_i - reduce$
21 $i \leftarrow (i + 1) mod\, n$
22 **return** $U' = \bigcup_{j=1}^{n} \{u'_j\}$

We will start by showing some results useful for the following proofs. Let j be the number of iterations of the **while** loop for the rest of the proofs for algorithm 2 (think of i from line 20 without the $mod\, n$).
First we will show that $empty \leq n$. $empty$ is only modified on line 12 where it is incremented by 1. This happens only when $u'_i > 0$ (line 11), which is assigned the value 0 on line 16. We can see that the incrementation of $empty$ can happen at most n times because $|U'| = n$. Since $empty_0 = 0$, $empty \leq n$ at all times of the execution.
Next we will derive the recursive formulas for the various variables.

$$empty_0 = 0$$

$$empty_{j+1} = \begin{cases} empty_j, & u'_{(j+1) \bmod n} = 0 \\ empty_j + 1, & u'_{(j+1) \bmod n} > 0 \ \wedge \ x_{(j+1) \bmod n} < reduce_j \\ empty_j, & u'_{(j+1) \bmod n} > 0 \ \wedge \ x_{(j+1) \bmod n} \geq reduce_j \end{cases}$$

$$reduce_0 = \frac{V}{n}$$

$$reduce_{j+1} = \begin{cases} reduce_j, & u'_{(j+1) \bmod n} = 0 \\ reduce_j + \frac{reduce_j - x_{(j+1) \bmod n}}{n - empty_{j+1}}, & u'_{(j+1) \bmod n} > 0 \ \wedge \ x_{(j+1) \bmod n} < reduce_j \\ reduce_j, & u'_{(j+1) \bmod n} > 0 \ \wedge \ x_{(j+1) \bmod n} \geq reduce_j \end{cases}$$

$$reduction_0 = 0$$

$$reduction_{j+1} = \begin{cases} reduction_j, & u'_{(j+1) \bmod n} = 0 \\ reduction_j + u'_{(j+1) \bmod n}, & u'_{(j+1) \bmod n} > 0 \ \wedge \ x_{(j+1) \bmod n} < reduce_j \\ reduction_j + u'_{(j+1) \bmod n} - x_{(j+1) \bmod n} + reduce_{j+1}, & u'_{(j+1) \bmod n} > 0 \ \wedge \ x_{(j+1) \bmod n} \geq reduce_j \end{cases}$$

In the end, $r = reduce$ is such that $r = \frac{V - \sum_{x \in S} x}{n - |S|}$ where $S = \{$flows y from s to $N^+(s)$ according to $maxFlow$:

$y < r\}$. Also, $\sum_{i=1}^{n} u'_i = \sum_{i=1}^{n} \max(0, x_i - r)$. TOPROVE

Proof of correctness for algorithm 2.

- We will show that $\forall i \in [n]\ u'_i \leq x_i$.
  On line 9, $\forall i \in [n]\ u'_i = x_i$. Subsequently u'_i is modified on line 16, where it becomes equal to 0 and on line 19, where it is assigned $x_i - reduce$. It holds that $x_i - reduce \leq x_i$ because initially $reduce = \frac{V}{n} \geq 0$ and subsequently $reduce$ is modified only on line 14 where it is increased ($n > empty$ because of line 13 and $reduce > x_i$ because of line 11, thus $\frac{reduce - x_i}{n - empty} > 0$). We see that $\forall i \in [n], u'_i \leq x_i$.

- We will show that $\sum_{i=1}^{n} u'_i = F - V$.
  The variable $reduction$ keeps track of the total reduction that has happened and breaks the **while** loop when $reduction \geq V$. We will first show that $reduction = \sum_{i=1}^{n}(x_i - u'_i)$ at all times and then we will prove that $reduction = V$ at the end of the execution. Thus we will have proven that $\sum_{i=1}^{n} u'_i = \sum_{i=1}^{n} x_i - V = F - V$.

  - On line 9, $u'_i = x_i \Rightarrow \sum_{i=1}^{n}(x_i - u'_i) = 0$ and $reduction = 0$.
    On line 16, u'_i is reduced to 0 thus $\sum_{i=1}^{n}(x_i - u'_i)$ is increased by u'_i. Similarly, on line 15 $reduction$ is increased by u'_i, the same as the increase in $\sum_{i=1}^{n}(x_i - u'_i)$.
    On line 19, u'_i is reduced by $u'_i - x_i + reduce$ thus $\sum_{i=1}^{n}(x_i - u'_i)$ is increased by $u'_i - x_i + reduce$. On line 18, $reduction$ is increased by $u'_i - x_i + reduce$, which is equal to the increase in $\sum_{i=1}^{n}(x_i - u'_i)$. We also have to note that neither u'_i nor $reduction$ is modified in any other way from line 10 and on, thus we conclude that $reduction = \sum_{i=1}^{n}(x_i - u'_i)$ at all times.
  - Suppose that $reduction_j > V$ on the line 21. Since $reduction_j$ exists, $reduction_{j-1} < V$. If $x_{j \bmod n} < reduce_{j-1}$ then $reduction_j = reduction_{j-1} + u'_{j \bmod n}$. Since $reduction_j > V$, $u'_{j \bmod n} > V - reduction_{j-1}$. TOCOMPLETE

□

Complexity of algorithm 2.
In the worst case scenario, each time we iterate over all capacities only the last non-zero capacity will become zero and every non-zero capacity must be recalculated. This means that every n steps exactly 1 capacity becomes zero and eventually all capacities (maybe except for one) become zero. Thus we need $O(n^2)$ steps in the worst case. □

A variation of this algorithm using a Fibonacci heap with complexity $O(n)$ can be created, but that is part of further research.

Proof that algorithm 2 minimizes the $||\Delta_i||_\infty$ norm.
Suppose that U' is the result of an execution of algorithm 2 that does not minimize the $||\Delta_i||_\infty$ norm. Suppose that W is a valid solution that minimizes the $||\Delta_i||_\infty$ norm. Let δ be the minimum value of this norm. There exists $i \in [n]$ such that $x_i - w_i = \delta$ and $u'_i < w_i$. Because both U' and W are valid solutions ($\sum_{i=1}^{n} u'_i = \sum_{i=1}^{n} w_i = F - V$), there must exist a set $S \subset U'$ such that $\forall u'_j \in S, u'_j > w_j$ TOCOMPLETE. □

Algorithm 3: Proportional equality trust transfer

Input : x_i flows, $n = |N^+(s)|$, V value
Output: u'_i capacities

1 $F \leftarrow \sum_{i=1}^{n} x_i$
2 **if** $F < V$ **then**
3 | **return** $\perp$
4 **for** $i \leftarrow 1$ *to* n **do**
5 | $u'_i \leftarrow x_i - \frac{V}{F} x_i$
6 **return** $U' = \bigcup_{j=1}^{n} \{u'_j\}$

Proof of correctness for algorithm 3.

- We will show that $\forall i \in [n]\ u'_i \leq x_i$.
  According to line 5, which is the only line where u'_i is changed, $u'_i = x_i - \frac{V}{F} x_i \leq x_i$ since $x_i, V, F > 0$ and $V \leq F$.
- We will show that $\sum_{i=1}^{n} u'_i = F - V$.
  With $F = \sum_{i=1}^{n} x_i$, on line 6 it holds that $\sum_{i=1}^{n} u'_i = \sum_{i=1}^{n} (x_i - \frac{V}{F} x_i) = \sum_{i=1}^{n} x_i - \frac{V}{F} \sum_{i=1}^{n} x_i = F - V$.

□

Complexity of algorithm 3.
The complexity of lines 1, 4-5 and 6 is $O(n)$ and the complexity of lines 2-3 is $O(1)$, thus the total complexity of algorithm 3 is $O(n)$. □

Naive algorithms result in $u'_i \leq x_i$, thus according to 7.6, $||\delta_i||_1$ is invariable for any of the possible solutions U', which is not necessarily the minimum (usually it will be the maximum). The following algorithms concentrate on minimizing two δ_i norms, $||\delta_i||_\infty$ and $||\delta_i||_1$.

Algorithm 4: $||\delta_i||_\infty$ minimizer

Input : $X = \{x_i\}$ flows, $n = |N^+(s)|$, V value, ϵ_1, ϵ_2
Output: u'_i capacities

1 **if** $\epsilon_1 < 0 \vee \epsilon_2 < 0$ **then**
2 | **return** $\perp$
3 $F \leftarrow \sum_{i=1}^{n} x_i$
4 **if** $F < V$ **then**
5 | **return** $\perp$
6 $\delta_{max} \leftarrow \max_{i \in [n]} \{u_i\}$
7 $\delta^* \leftarrow$ `BinSearch(`$0, \delta_{max}, F\text{-}V, n, X, \epsilon_1, \epsilon_2$`)`
8 **for** $i \leftarrow 1$ *to* n **do**
9 | $u'_i \leftarrow \max(u_i - \delta^*, 0)$
10 **return** $U' = \bigcup_{j=1}^{n} \{u'_j\}$

Since trust should be considered as a continuous unit and binary search dissects the possible interval for the solution on each recursive call, inclusion of the ϵ-parameters in `BinSearch` is necessary for the

algorithm to complete in a finite number of steps.

Algorithm 4: function `BinSearch`

Input : bot, top, F', n, X, ϵ_1, ϵ_2
Output: δ^*

```
1  if bot = top then
2  |  return bot
3  else
4  |  for i ← 1 to n do
5  |  |  u'_i ← max(0, u_i − (top+bot)/2)
6  |  if maxFlow < F' − ε1 then
7  |  |  return BinSearch(bot, (top+bot)/2, F', n, X, ε1, ε2)
8  |  else if maxFlow > F' + ε2 then
9  |  |  return BinSearch((top+bot)/2, top, F', n, X, ε1, ε2)
10 |  else
11 |  |  return (top+bot)/2
```

Proof that $maxFlow(\delta)$ is strictly decreasing for $\delta : maxflow(\delta) < F$.
Let $maxFlow(\delta)$ be the $maxFlow$ with $\forall i \in [n], u'_i = max(0, u_i - \delta)$. We will prove that the function $maxFlow(\delta)$ is strictly decreasing for all $\delta \leq \max_{i\in[n]}\{u_i\}$ such that $maxFlow(\delta) < F$.

Supppose that $\exists \delta_1, \delta_2 : \delta_1 < \delta_2 \wedge maxFlow(\delta_1) \leq maxFlow(\delta_2) < F$. We will work with configurations of $x'_{i,j}$ such that $x'_{i,j} \leq x_i, j \in \{1, 2\}$.
Let $S_j = \{i \in N^+(s) : i \in MinCut_j\}$. It holds that $S_1 \neq \emptyset$ because otherwise $MinCut_1 = MinCut_{\delta=0}$ which is a contradiction because then $maxFlow(\delta_1) = F$. Moreover, it holds that $S_1 \subseteq S_2$, since $\forall u'_{i,2} > 0, u'_{i,2} < u'_{i,1}$. Every node in the $MinCut_j$ is saturated, thus $\forall i \in S_1, x'_{i,j} = u'_{i,j}$. Thus $\sum_{i\in S_1} x_{i,2} < \sum_{i\in S_1} x_{i,1}$ and, since $maxFlow(\delta_1) \leq maxFlow(\delta_2)$, we conclude that for the same configurations, $\sum_{i\in N^+(s)\setminus S_1} x_{i,2} > \sum_{i\in N^+(s)\setminus S_1} x_{i,1}$.
However, since $x'_{i,j} \leq x_i, j \in \{1, 2\}$, the configuration $[x''_{i,1} = x'_{i,2}, i \in N^+(s) \setminus S_1], [x''_{i,1} = x'_{i,1}, i \in S_1]$ is valid for $\delta = \delta_1$ and then $\sum_{i\in S_1} x''_{i,1} + \sum_{i\in N^+(s)\setminus S_1} x''_{i,1} = \sum_{i\in S_1} x'_{i,1} + \sum_{i\in N^+(s)\setminus S_1} x'_{i,2} > maxFlow(\delta_1)$, contradiction. Thus $maxFlow(\delta)$ is strictly decreasing. □

We can see that if $V > 0, F' = F - V < F$ thus if $\delta \in (0, \max_{i\in[n]}\{u_i\}] : maxFlow(\delta) = F' \Rightarrow \delta = \min ||\delta_i||_\infty : maxFlow(||\delta_i||_\infty) = F'$.

Proof of correctness for function 5.
Supposing that $[F' - \epsilon_1, F' + \epsilon_2] \subset [maxFlow(top), maxFlow(bot)]$, or equivalently $maxFlow(top) \leq F' - \epsilon_1 \wedge maxFlow(bot) \geq F' + \epsilon_2$, we will prove that $maxFlow(\delta^*) \in [F' - \epsilon_1, F' + \epsilon_2]$.
First of all, we should note that if an invocation of `BinSearch` returns without calling `BinSearch` again (line 2 or 11), its return value will be equal to the return value of the initial invocation of `BinSearch`, as we can see on lines 7 and 9, where the return value of the invoked `BinSearch` is returned without any modification. The case where `BinSearch` is called again is analyzed next:

- If $maxFlow(\frac{top+bot}{2}) < F' - \epsilon_1 < F'$ (line 6) then, since $maxFlow(\delta)$ is strictly decreasing, $\delta^* \in [bot, \frac{top+bot}{2})$. As we see on line 7, the interval $(\frac{top+bot}{2}, top]$ is discarded when the next `BinSearch` is called. Since $F' + \epsilon_2 \leq maxFlow(bot)$, we have $[F' - \epsilon_1, F' + \epsilon_2] \subset [maxFlow(\frac{top+bot}{2}), maxFlow(bot)]$ and the length of the available interval is divided by 2.
- Similarly, if $maxFlow(\frac{top+bot}{2}) > F' + \epsilon_2 > F'$ (line 8) then $\delta^* \in (\frac{top+bot}{2}, top]$. According to line 9, the interval $[bot, \frac{top+bot}{2})$ is discarded when the next `BinSearch` is called. Since $F' - \epsilon_1 \geq maxFlow(top)$, we have $[F' - \epsilon_1, F' + \epsilon_2] \subset (maxFlow(top), maxFlow(\frac{top+bot}{2})]$ and the length of the available interval is divided by 2.

As we saw, $[F' - \epsilon_1, F' + \epsilon_2] \subset [maxFlow(top), maxFlow(bot)]$ in every recursive call and $top - bot$ is divided by 2 in every call. From topology we know that $A \subset B \Rightarrow |A| < |B|$, so the recursive calls

cannot continue infinitely. $|[F' - \epsilon_1, F' + \epsilon_2]| = \epsilon_1 + \epsilon_2$. Let bot_0, top_0 the input values given to the initial invocation of `BinSearch`, bot_j, top_j the input values given to the j-th recursive call of `BinSearch` and $len_j = |[bot_j, top_j]| = top_j - bot_j$. We have $\forall j > 0, len_j = top_j - bot_j = \frac{top_{j-1} - bot_{j-1}}{2} \Rightarrow \forall j > 0, len_j = \frac{top_0 - bot_0}{2^j}$. We understand that in the worst case $len_j = \epsilon_1 + \epsilon_2 \Rightarrow 2^j = \frac{top_0 - bot_0}{\epsilon_1 + \epsilon_2} \Rightarrow j = \log_2(\frac{top_0 - bot_0}{\epsilon_1 + \epsilon_2})$. Also, as we saw earlier, δ^* is always in the available interval, thus $maxFlow(\delta^*) \in [F' - \epsilon_1, F' + \epsilon_2]$. □

Complexity of function 5.
Lines 1-2 have complexity $O(1)$, lines 4-5 have complexity $O(n)$, lines 6-11 have complexity $O(maxFlow) + O(BinSearch)$. As we saw in the proof of correctness for function 5, we need at most $\log_2(\frac{top - bot}{\epsilon_1 + \epsilon_2})$ recursive calls of `BinSearch`. Thus the function 5 has worst-case complexity $O((maxFlow + n)\log_2(\frac{top - bot}{\epsilon_1 + \epsilon_2}))$. □

Proof of correctness for algorithm 4.
We will show that $maxFlow \in [F - V - \epsilon_1, F - V + \epsilon_2]$, with u_i' decided by algorithm 4.
Obviously $maxFlow(0) = F, maxFlow(\max_{i \in [n]}\{u_i\}) = 0$, thus $\delta^* \in \max_{i \in [n]}\{u_i\}$. According to the proof of correctness for function 5, we can directly see that $maxFlow(\delta^*) \in [F - V - \epsilon_1, F - V + \epsilon_2]$, given that ϵ_1, ϵ_2 are chosen so that $F - V - \epsilon_1 \geq 0, F - V + \epsilon_2 \leq F$, so as to satisfy the condition $[F' - \epsilon_1, F' + \epsilon_2] \subset [maxFlow(top), maxFlow(bot)]$. □

Complexity of algorithm 4.
The complexity of lines 1,2 and 4-6 is $O(n)$ and the complexity of line 3 is $O(BinSearch) = O((maxFlow + n)\log_2(\frac{\delta_{max}}{\epsilon_1 + \epsilon_2}))$, thus the total complexity of algorithm 4 is $O((maxFlow + n)\log_2(\frac{\delta_{max}}{\epsilon_1 + \epsilon_2}))$. □

However, we need to minimize $\sum_{i=1}^{n}(u_i - u_i') = ||\delta_i||_1$.

8 Further Research

9 References