Trust Is Risk: Introducing a decentralized platform for financial trust

Orfeas Stefanos Thyfronitis Litos
National Technical University of Athens
orfeas.litos@hotmail.com

Dionysis Zindros[†]
University of Athens
dionyziz@di.uoa.gr

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 [6] 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 define financial risk and prove that risk and max flows are equivalent. We then 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

Modern marketplaces can be roughly categorized as centralized and decentralized. Two major examples are ebay and OpenBazaar. ebay is centralized and as such it is vulnerable to ddos attacks [1] and can be considered as a single point of failure, it charges fees for the use of its services [2] and maintains a private database of personal data, but it can give money-back guarantees [3] since it is run by a single company that has a financial advantage in keeping its clients satisfied. On the other hand, OpenBazaar is a decentralized platform built on bitcoin [6], where individual stores or its search engine are vulnerable to ddos attacks [1], but not the platform as a whole. Additionally, it does not charge fees for its usage [4] and there is no central agent recording all the transactions alongside with private data [4] but it is possible for a buyer or a seller colluding with a moderator to scam the third party and there exists no central authority able to verify the truth of her claim and reimburse her [5]. Even though trust (or distrust) should be directed to each store individually, it is very likely that the whole platform will be discarded as untrusted by the scammed party.

Our goal is to create a distributed marketplace where the trust each user gives to the rest of the users is quantifiable, measurable and expressable in monetary terms. The central concept used throughout this paper is trust-as-risk, or the proposition that a user's A *trust* to another user B is defined to be the *maximum sum of money* A can lose when B is free to choose any strategy she wants. To flesh out trust-as-risk, we will use *lines of credit* as proposed by Washington Sanchez [citation needed]. Joining the network will be done by explicitly entrusting a certain amount of money to another user. If the second user has already entrusted an amount of money to a store, then we trust indirectly the store, thus we can engage in economic interaction with said store.

We thus propose a new kind of wallet where coins are not stored locally, but are placed in 1-of-2 multisigs, a bitcoin construction that permits any one of two pre-designated users to spend the coins contained therein [10].

Our approach drastically changes the user experience. A user no more has to worry about stars and ratings to develop trust towards a store, she can simply check the trust *flowing* from her to the store (which will be a number expressed in bitcoins) and if this number exceeds the price of the product, she is safe to complete the transaction after modifying her direct trust towards her friends in an appropriate way, a

[†]Research supported by ERC project CODAMODA, project #259152

process that can be handled by one of several algorithms that we propose, or in any custom way the user chooses to. On the other side, there is no guarantee that the store will complete is part of the transaction and no central authority can reimburse the money. However, it is possible for the defrauded user to make up for her loss by in turn stealing from other users that trust her, an action that will tautologically reduce their trust to her. The fact that this system can function in a completely distributed fashion will become clear in the following sections, along with the positive result that it is in principle Sybil resilient.

There are several incentives for a user to join this network. First of all, they can have access to a store that is otherwise unaccessible. Moreover, two friends can formalize their mutual trust by entrusting the same amount to each other. A large company that casually subcontracts other companies to complete various tasks can express its trust towards them using this method. A government can choose to entrust its citizens with money and confront them using its legal arsenal if they make irresponsible use of this trust. A bank can provide loans as outgoing and manage savings as incoming trust and thus has a unique opportunity of expressing in a formal and absolute way its credence by publishing its incoming trust. Last but not least, the network can be viewed as a possible field for investment and speculation since it constitutes a completely new area for financial activity.

We should note that the current description of TrustIsRisk refers to a static setting where the game evolves in turns. In each turn only one user changes the state of the network. Also the system we describe needs full network knowledge for all of the users, a situation that may be undesirable because of the privacy-invading implications it brings. Future work is needed to address both limitations.

3 Keywords

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

4 A detailed example

The state of the game at any given moment can be represented by a directed weighted graph where each node represents a pseudonymous identity that is considered to correspond to exactly one user and each edge represents a direct trust, the weight of which is the value which the head entrusts the tail with. [Image]
Each node has a corresponding non-negative number that represents its capital. A node's capital is the total value that the node directly and exclusively controls. More technically, it is the number of bitcoins contained in p2pkh transactions in the utxo of which the private keys are controlled by the node. A rational player would like to maximize her capital in the long term. Let's suppose that player A wants to buy a product that costs 10 from player B, but A has not given any direct trust to B. Does A's friends trust B enough so that A can buy the product? In other words, should B decide to steal all its incoming direct trust and stop trusting directly other users and the rest of the players choose the conservative strategy of stealing from their respective incoming trusts just enough to replenish their loss [rationale needed], what would the eventual loss of A be? If it can be higher than the value of the product, then A is already exposed to a higher risk to B, thus paying for the product can be done without A being exposed to any additional risk towards B, as long as A reduces the direct trust to her friends enough to reduce the risk by the value of the product. If the transaction completes in a satisfying manner, A can replenish the reduced trust to her friends, or can even choose to start directly trusting B.
Suppose that at the moment A is considering of performing a payment to him, B really decides to depart from the network, taking with him all the coins he can. A chain reaction starts throughout the network because each player that B exploited is expected to steal enough value from other users that directly trust her to replenish all the value she lost to B, the newly exploited users will act in the same way and so on, until a new equilibrium is reached (6.1). Going into detail, [Image]
(Completion of example)

5 Definitions

Definition 5.1 (Graph)**.**
TrustIsRisk is represented by a sequence of weighted directed graphs $(\mathcal{G}_j)$ where $\mathcal{G}_j = (\mathcal{V}_j, \mathcal{E}_j), j \in \mathbb{N}$.

Members of $\mathcal{E}_j$ are tuples of two nodes from $\mathcal{V}_j$. More formally, $e \in \mathcal{E}_j \Rightarrow \exists A, B \in \mathcal{V}_j : e = (A, B)$. Also, since the graphs are weighted, there exists a sequence of functions (c_j) with $c_j : \mathcal{E}_j \to \mathbb{R}^+$.

Definition 5.2 (Players)**.**
The set $\mathcal{V}_j = V(\mathcal{G}_j)$ is the set of all players in the network, otherwise understood as the set of all pseudonymous identities.

Definition 5.3 (Capital of A, Cap_A)**.**
Total amount of value that exists in P2PKH in the UTXO and can be spent by A. We also define $Cap_{A,j}$ as the total amount of value that exists in P2PKH in the UTXO and can be spent by A during turn j.

Definition 5.4 (Direct Trust from A to B after turn j, $DTr_{A\to B,j}$)**.**
Total amount of value that exists in $1/\{A, B\}$ multisigs in the UTXO in the end of turn j, where the money is deposited by A.

$$DTr_{A\to B,j} = \begin{cases} c_j(A,B), & if (A,B) \in \mathcal{E}_j \\ 0, & if (A,B) \notin \mathcal{E}_j \end{cases}$$

A function or algorithm that has access to the graph $\mathcal{G}_j$ has implicitly access to all direct trusts of this graph. The exception are the oracles, which in this case have access only to their incoming and outgoing direct trusts.

Definition 5.5 ((In/Out) Neighbourhood of A on turn j, $N^+(A)_j, N^-(A)_j, N(A)_j$)**.**

1. Let $N^+(A)_j$ be the set of players B that A directly trusts with any positive value at the end of turn j. More formally, $N^+(A)_j = \{B \in \mathcal{V}_j : DTr_{A\to B,j} > 0\}$. $N^+(A)_j$ is called out neighbourhood of A on turn j. Let also $S \subset \mathcal{V}_j$. $N^+(S)_j = \bigcup_{A\in S} N^+(A)_j$.

2. Let $N^-(A)_j$ be the set of players B that directly trust A with any positive value at the end of turn j. More formally, $N^-(A)_j = \{B \in \mathcal{V}_j : DTr_{B\to A,j} > 0\}$. $N^-(A)_j$ is called in neighbourhood of A on turn j. Let also $S \subset \mathcal{V}_j$. $N^-(S)_j = \bigcup_{A\in S} N^-(A)_j$.

3. Let $N(A)_j$ be the set of players B that either directly trust or are directly trusted by A with any positive value at the end of turn j. More formally, $N(A)_j = N^+(A)_j \cup N^-(A)_j$. $N(A)_j$ is called neighbourhood of A on turn j. Let also $S \subset \mathcal{V}_j$. $N(S)_j = N^+(S)_j \cup N^-(S)_j$.

4. Let $N(A)_{j,i}$ (respectively $N^+(A)_{j,i}, N^-(A)_{j,i}, N(S)_{j,i}, N^+(S)_{j,i}, N^-(S)_{j,i}, S \subset \mathcal{V}_j$) be the i-th element of set $N(A)_j$ (respectively of $N^+(A)_j, N^-(A)_j, N(S)_j, N^+(S)_j, N^-(S)_j$), according to an arbitrary but fixed enumeration of the set players.

Definition 5.6 (Total incoming/outgoing trust of A in turn j, $in_{A,j}, out_{A,j}$)**.**

$$in_{A,j} = \sum_{v\in N^-(A)_j} DTr_{v\to A,j}$$

$$out_{A,j} = \sum_{v\in N^+(A)_j} DTr_{A\to v,j}$$

Definition 5.7 (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 $A \in \mathcal{V}, A = Player(j)$, chooses an action (according to a certain strategy) that can be one of the following, or a finite combination thereof:

1. Steal value $y_B, 0 \le y_B \le DTr_{B\to A,j-1}$ from $B \in N^-(A)$. $DTr_{B\to A,j} = DTr_{B\to A,j-1} - y_B$. $(Steal(y_B, B))$

2. Add value $y_B, -DTr_{A\to B,j-1} \le y_B$ to $B \in \mathcal{V}$. $DTr_{A\to B,j} = DTr_{A\to B,j-1} + y_B$. 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. If $DTr_{A\to B,j-1} = 0$, then we say that A starts directly trusting B. $(Add(y_B, B))$

If player A chooses no action in her turn, we say that she passes her turn. Also, let Y_{st}, Y_{add} be the total value to be stolen and added respectively by A in her turn, j. For a turn to be feasible, it must hold that $Y_{add} - Y_{st} \leq Cap_{A,j-1}$. We set $Cap_{A,j} = Cap_{A,j-1} + Y_{st} - Y_{add}$. Moreover, player A is not allowed to choose two actions of the same kind against the same player in the same turn.
The set of actions a player makes in turn j is $Turn_j$. Examples:

- $Turn_{j_1} = \emptyset$
- $Turn_{j_2} = \{Steal(y, B), Add(w, B)\}$ (given that $DTr_{B \to A, j_2 - 1} \leq y \wedge -DTr_{A \to B, j_2 - 1} \leq w \wedge y - w \leq Cap_{A, j_2 - 1}$, where $A = Player(j_2)$)
- $Turn_{j_3} = \{Steal(x, B), Add(y, C), Add(w, D)\}$ (given that $DTr_{B \to A, j_3 - 1} \leq x \wedge -DTr_{A \to C, j_3 - 1} \leq y \wedge -DTr_{A \to D, j_3 - 1} \leq w \wedge x - y - w \leq Cap_{A, j_3 - 1}$, where $A = Player(j_3)$)
- $Turn_{j_4} = \{Steal(x, B), Steal(y, B)\}$ is not a valid turn because it contains two $Steal()$ actions against the same player. If $x + y \leq DTr_{B \to A}$, the correct alternative would be $Turn_{j_4} = \{Steal(x + y, B)\}$, where $A = Player(j_4)$.

Definition 5.8 (Previous/Next turn of a player)**.**
Let $j \in \mathbb{N}$ a turn with $Player(j) = A$. We define $prev(j), next(j)$ as the previous and next turn that A is chosen to play respectively. If j is the first turn that A plays, $prev(j) = 0$. More formally,

$$prev(j) = \begin{cases} \max\{k \in \mathbb{N} : k < j \wedge Player(k) = A\}, & \{k \in \mathbb{N} : k < j \wedge Player(k) = A\} \neq \emptyset \\ 0, & \{k \in \mathbb{N} : k < j \wedge Player(k) = A\} = \emptyset \end{cases}$$

$$next(j) = \min\{k \in \mathbb{N} : k > j \wedge Player(k) = A\}$$

$next(j)$ is always well defined with the assumption that eventually everybody plays.

Definition 5.9 (A is stolen x)**.**
Let j, j' be two consecutive turns of A ($next(j) = j'$). We say that A has been stolen a value x between j and j' if $out_{A,j} - out_{A,j'} = x > 0$. If turns are not specified, we implicitly refer to the current and the previous turns.

Definition 5.10 (History)**.**
We define History, $\mathcal{H} = (\mathcal{H}_j)$, as the sequence of all the tuples containing the sets of actions and the corresponding player. $\mathcal{H}_j = (Player(j), Turn_j)$.

Definition 5.11 (Conservative strategy)**.**
A player A is said to follow the conservative strategy in turn j 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 $\min(x, in_{A,j})$ and she takes no other action. More formally, let $j' = prev(j), Damage_j = out_{A,j'} - out_{A,j-1}$. If $Strategy(A) = Conservative$, then $\forall j \in \mathbb{N} : Player(j) = A$ it is

$$Turn_j = \begin{cases} \emptyset, & Damage_j \leq 0 \\ \bigcup_{i=1}^{k} \{Steal(y_i, v_i)\}, & Damage_j > 0, N^-(A)_j = \{v_1, ..., v_k\} \end{cases}$$

In the second case, it is $\sum_{i=1}^{k} y_i = \min(in_{A,j-1}, Damage_j)$.

As we can see, the definition covers a multitude of options for the conservative player, since in case $0 < Damage_j < in_{A,j-1}$ she can choose to distribute the $Steal(s)()$ in any way she chooses, as long as $\forall i, y_i \leq DTr_{N^-(A)_{j,i} \to A, j-1} \wedge \sum_{i=1}^{|N^-(A)_j|} y_i = Damage_j$. The oracle remembers $PrevOutTrust = out_{A,j'}$ for $j' = prev(j)$ and can observe all incoming and outgoing direct trusts of player A, $\forall v \in N^-(A)_{j-1}, DTr_{v \to A, j-1}, \forall v \in$

$N^+(A)_{j-1}, DTr_{A\to v,j-1}$. We note that $N(A)_{j-1} = N(A)_j$.

Algorithm 1: Conservative Oracle

Input : previous graph $\mathcal{G}_{j-1}$
Output: $Turn_j$

$\mathcal{O}_{cons}(\mathcal{G}_{j-1})$:
$NewOutTrust \leftarrow \sum_{v\in N^+(A)_{j-1}} DTr_{A\to v,j-1}$
$NewInTrust \leftarrow \sum_{v\in N^-(A)_{j-1}} DTr_{v\to A,j-1}$
$Damage \leftarrow PrevOutTrust - NewOutTrust$
if $Damage > 0$ **then**
  if $Damage \geq NewInTrust$ **then**
    $Turn_j \leftarrow \emptyset$
    for $v \in N^-(A)_{j-1}$ **do**
      $Turn_j \leftarrow Turn_j \cup \{Steal(DTr_{v\to A,j-1}, v)\}$
  else
    $(y_1, ..., y_{|N^-(A)_{j-1}|}) \leftarrow$ `SelectSteal` $(DTr_{N^-(A)_{j-1,1}\to A,j-1},\dots, DTr_{N^-(A)_{j-1,|N^-(A)_{j-1}|}\to A,j-1}, Damage)$
    $Turn_j \leftarrow \emptyset$
    for $i \leftarrow 1$ *to* $|N^-(A)_{j-1}|$ **do**
      $Turn_j \leftarrow Turn_j \cup \{Steal(y_i, N^-(A)_{j-1,i})\}$
else
  $Turn_j \leftarrow \emptyset$
return $Turn_j$

`SelectSteal()` returns $y_i, i \in [|N^-(A)_j|] : \forall i, y_i \leq DTr_{N^-(A)_{j,i}\to A}, \sum_{i=1}^{|N^-(A)_j|} y_i = Damage.$

Definition 5.12 (Idle strategy)**.**
A player A is said to follow the idle strategy if she passes in her turn. More formally, if $Strategy(A) = Idle$, then $\forall j \in \mathbb{N} : Player(j) = A$ it is $Turn_j = \emptyset$.

Algorithm 2: Idle Oracle

Input : previous graph $\mathcal{G}_{j-1}$
Output: $Turn_j$

$\mathcal{O}_{idle}(\mathcal{G}_{j-1})$:
return $\emptyset$

Definition 5.13 (Evil strategy)**.**
A player A is said to follow the evil strategy if she steals value $y_B = DTr_{B\to A,j-1} \,\forall\, B \in N^-(A)_j$ (steals all incoming direct trust) and reduces her trust to C by $DTr_{A\to C,j-1} \,\forall\, C \in N^+(A)_j$ (nullifies her outgoing direct trust) in her turn. More formally, if $Strategy(A) = Evil$, then $\forall j \in \mathbb{N} : Player(j) = A$ it is $Turn_j = \{Steal(y_1, N^-(A)_{j,1}), ..., Steal(y_m, N^-(A)_{j,m}), Add(w_1, N^+(A)_{j,1}), ..., Add(w_l, N^+(A)_{j,l})\}$ where $m = |N^-(A)_j|, l = |N^+(A)_j|, \forall i \in [m], y_i = DTr_{N^-(A)_{j,i}\to A,j-1}, \forall i \in [l], w_i = -DTr_{A\to N^+(A)_{j,i},j-1}$. We note again that $N(A)_{j-1} = N(A)_j$.

Algorithm 3: Evil Oracle

Input : previous graph $\mathcal{G}_{j-1}$
Output: $Turn_j$

$\mathcal{O}_{evil}(\mathcal{G}_{j-1})$:
$Turn_j \leftarrow \emptyset$
for $v \in N^-(A)_{j-1}$ **do**
  $Turn_j \leftarrow Turn_j \cup \{Steal(DTr_{v \to A,j-1}, v)\}$
for $w \in N^+(A)_{j-1}$ **do**
  $Turn_j \leftarrow Turn_j \cup \{Add(-DTr_{A \to v,j-1}, w)\}$
return $Turn_j$

Definition 5.14 (Indirect trust from $A \in \mathcal{V}_j$ to $B \in \mathcal{V}_j$, $Tr_{A \to B,j}$)**.**
Maximum possible value that can be stolen from A if B follows the evil strategy, A follows the idle strategy and everyone else ($\mathcal{V} \setminus \{A, B\}$) follows the conservative strategy. More formally,

$$Tr_{A \to B,j} = \max_{j':j'>j, configurations} [out_{A,j} - out_{A,j'}]$$

where $Strategy(A) = Idle, Strategy(B) = Evil, \forall C \in \mathcal{V} \setminus \{A, B\}, Strategy(C) = Conservative$.

Definition 5.15 (Indirect trust from $A \in \mathcal{V}_j$ to $S \subset \mathcal{V}_j$, $Tr_{A \to S,j}$)**.**
Maximum possible value that can be stolen from A if all players in S follow the evil strategy, A follows the idle strategy and everyone else ($\mathcal{V} \setminus (S \cup \{A\})$) follows the conservative strategy. More formally,

$$Tr_{A \to S,j} = \max_{j':j'>j, configurations} [out_{A,j} - out_{A,j'}]$$

where $Strategy(A) = Idle, \forall E \in S, Strategy(E) = Evil, \forall C \in \mathcal{V} \setminus \{A, E\}, Strategy(C) = Conservative$.

Definition 5.16 (Trust Reduction)**.**
Let $A, B \in \mathcal{V}, x_i$ flow to $N^+(A)_i$ resulting from $maxFlow(A, B), u_i = DTr_{A \to N^+(A)_i, j-1}, u'_i = DTr_{A \to N^+(A)_i, j}$, $i \in [|N^+(A)|], j \in \mathbb{N}$.

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 5.17 (Restricted Flow)**.**
Let $A, B \in \mathcal{V}, i \in [|N^+(A)|]$.

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 $u'_i = u_i$, $u'_k = 0 \, \forall k \in [|N^+(A)|] \wedge k \neq i$.
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 $u'_C = u_C \, \forall C \in S, u'_D = 0 \, \forall D \in N^+(A) \setminus S$.

Definition 5.18 (Collusion)**.**
A collusion of players $S \subset \mathcal{V}$ is a set of players that is entirely controlled by a single entity in the physical world. From a game theoretic point of view, other players ($v \in \mathcal{V} \setminus S$) perceive the collusion as independent players with a distinct strategy each, whereas in reality they are all subject to a single strategy dictated by the controlling entity.

6 Theorems-Algorithms

The following algorithm has read access to direct trusts in $\mathcal{G}_{j-1}$ and write access to direct trusts in $\mathcal{G}_j$.

Algorithm 4: Execute Turn

Input : player A, old graph $\mathcal{G}_{j-1}$, old capital $Cap_{A,j-1}$, $ProvisionalTurn$
Output: new graph $\mathcal{G}_j$, new capital $Cap_{A,j}$, new history $\mathcal{H}_j$

`executeTurn(`A, $\mathcal{G}_{j-1}$, $Cap_{A,j-1}$, $ProvisionalTurn$`)` :
$(Turn_j, NewCap) \leftarrow$ `validateTurn(`A, $\mathcal{G}_{j-1}$, $Cap_{A,j-1}$, $ProvisionalTurn$`)`
return `commitTurn(`A, $mathcalG_{j-1}$, $NewCap$, $Turn_j$`)`

Algorithm 5: Validate Turn

Input : player A, old graph $\mathcal{G}_{j-1}$, old capital $Cap_{A,j-1}$, $ProvisionalTurn$
Output: $Turn_j$, new capital $Cap_{A,j}$

`validateTurn(`A, $\mathcal{G}_{j-1}$, $Cap_{A,j-1}$, $ProvisionalTurn$`)` :
$Y_{st} \leftarrow 0$
$Y_{add} \leftarrow 0$
for $action \in ProvisionalTurn$ **do**
  $action$ **match do**
  case $Steal(y, w)$ **do**
  if $y > DTr_{w \to A,j-1} \vee y < 0$ **then**
    return $\emptyset$, $Cap_{A,j-1}$
  else
    $Y_{st} \leftarrow Y_{st} + y$
  case $Add(y, w)$ **do**
  if $y < -DTr_{A \to w,j-1}$ **then**
    return $\emptyset$, $Cap_{A,j-1}$
  else
    $Y_{add} \leftarrow Y_{add} + y$
if $Y_{add} - Y_{st} > Cap_{A,j-1}$ **then**
  return $\emptyset$, $Cap_{A,j-1}$
else
  return $ProvisionalTurn$, $Cap_{A,j-1} + Y_{st} - Y_{add}$

Algorithm 6: Commit Turn

Input : player A, old graph $\mathcal{G}_{j-1}$, old capital $Cap_{A,j-1}$, $ProvisionalTurn$
Output: new graph $\mathcal{G}_j$, new capital $Cap_{A,j}$, new history $\mathcal{H}_j$

`commitTurn(`A, $\mathcal{G}_{j-1}$, $Cap_{A,j-1}$, $Turn_j$`)` :
for $(v, w) \in \mathcal{E}_j$ **do**
  $DTr_{v \to w,j} \leftarrow DTr_{v \to w,j-1}$
for $action \in Turn_j$ **do**
  $action$ **match do**
  case $Steal(y, w)$ **do**
  $DTr_{w \to A,j} \leftarrow DTr_{w \to A,j-1} - y$
  case $Add(y, w)$ **do**
  $DTr_{A \to w,j} \leftarrow DTr_{A \to w,j} + y$
$Cap_{A,j} \leftarrow NewCap$
$\mathcal{H}_j \leftarrow (Player(j), Turn_j)$
return $\mathcal{G}_j$, $Cap_{A,j}$, $\mathcal{H}_j$

Algorithm 7: TrustIsRisk Game

$j \leftarrow 0$
while $True$ **do**
  $j \leftarrow j + 1$
  $v \xleftarrow{\$} \mathcal{V}_j$
  $ProvisionalTurn \leftarrow \mathcal{O}_v(\mathcal{G}_{j-1})$
  $(G_j, Cap_{v,j}, H_j) \leftarrow$ `executeTurn(`v, $\mathcal{G}_{j-1}$, $Cap_{v,j-1}$, $ProvisionalTurn$`)`

On turn 0, there is already a network in place.

Algorithm 8: Transitive Steal

Input : A idle player, E evil player
Output: $\mathcal{H}$ history

$Angry \leftarrow \emptyset$
$Happy \leftarrow \emptyset$
$Sad \leftarrow \emptyset$
for $v \in \mathcal{V}_0 \setminus \{E\}$ **do**
  $Loss_v \leftarrow 0$
  if $v \neq A$ **then**
    $Happy \leftarrow Happy \cup \{v\}$
$j \leftarrow 0$
while *True* **do**
  $j \leftarrow j + 1$
  $v \xleftarrow{\$} \mathcal{V}_j \setminus \{A\}$
  $Turn_j \leftarrow \mathcal{O}_v(\mathcal{G}_{j-1})$
  `executeTurn` $(\mathcal{G}_{j-1}, Cap_{v,j-1}, Turn_j)$
  for $action \in Turn_j$ **do**
    $action$ **match do**
    case $Steal(y, w)$ **do**
    $exchange \leftarrow y$
    $Loss_w \leftarrow Loss_w + exchange$
    if $v \neq E$ **then**
      $Loss_v \leftarrow Loss_v - exchange$
    if $w \neq A$ **then**
      $Happy \leftarrow Happy \setminus \{w\}$
      if $in_{w,j} = 0$ **then**
        $Sad \leftarrow Sad \cup \{w\}$
      else
        $Angry \leftarrow Angry \cup \{w\}$
  $Angry \leftarrow Angry \setminus \{v\}$
  if $in_{v,j} = 0 \wedge Loss_v > 0$ **then**
    $Sad \leftarrow Sad \cup \{v\}$
  if $Loss_v = 0$ **then**
    $Happy \leftarrow Happy \cup \{v\}$

Let j_0 be the first turn on which E is chosen to play. Until then, according to theorem 6.4, all players will pass their turn. Given that $Damage_{v,j} = out_{v,j'} - out_{v,j}$ where $j' = prev(j)$, the algorithm generates turns:

$$Turn_j = \begin{cases} \emptyset, & Damage_{v,j-1} = 0 \\ \bigcup_{i=1}^{k} \{Steal(y_i, v_i)\}, & Damage_j > 0, N^-(A)_j = \{v_1, ..., v_k\} \end{cases}$$

In the second case, it is $\sum_{i=1}^{k} y_i = \min(in_{v,j-1}, Damage_{v,j-1})$. From the definition of $Damage_{v,j}$ and knowing that no strategy in this case can increase any direct trust, it is obvious that $Damage_{v,j} \geq 0$. Also, we can see that $Loss_{v,j} \geq 0$ because if $Loss_{v,j} < 0$, then v has stolen more value than she has been stolen, thus she would not be following the conservative strategy.

Lemma 6.1 (*Loss* equivalent to *Damage*)**.**
Let $j \in \mathbb{N}, v \in \mathcal{V}_j \setminus \{A, E\}, v = Player(j)$. *Then* $\min(in_{v,j}, Loss_{v,j}) = \min(in_{v,j}, Damage_{v,j})$.

Proof. $j \in \mathbb{N} : v = Player(j)$.

- $v \in Happy_{j-1}$. Then

1. $v \in Happy_j$ because $Turn_j = \emptyset$,
2. $Loss_{v,j} = 0$ because otherwise $v \notin Happy_j$,
3. $Damage_{v,j} = 0$, or else any reduction in direct trust to v would increase equally $Loss_{v,j}$ (line 18), which cannot be decreased again but during an Angry player's turn (line 20).
4. $in_{v,j} \geq 0$

Thus $\min(in_{v,j}, Damage_{v,j}) = \min(in_{v,j}, Loss_{v,j}) = 0$.

- $v \in Sad_{j-1}$. Then

1. $v \in Sad_j$ because $Turn_j = \emptyset$,
2. $in_{v,j} = 0$ (lines 28 - 29),
3. $Damage_{v,j} \geq 0 \wedge Loss_{v,j} \geq 0$.

Thus $\min(in_{v,j}, Damage_{v,j}) = \min(in_{v,j}, Loss_{v,j}) = 0$.

- $v \in Angry_{j-1} \wedge v \in Happy_j$. Then the same argument as in the first case holds, if we ignore the argument (1).
- $v \in Angry_j \wedge v \in Sad_j$. Then the same argument as in the second case holds, if we ignore the argument (1).

□

Theorem 6.1 (Trust convergence theorem)**.**
Let $A, E \in \mathcal{V} : Strategy(A) = Idle$, $Strategy(E) = Evil$, $\forall C \in \mathcal{V} \setminus \{A, E\}, Strategy(C) = Conservative$ and $j_0 \in \mathbb{N} : Player(j_0) = E$. Given that all players eventually play, there exists a turn $j' > j_0 : \forall j \geq j', Turn_j = \emptyset$.

Proof. First of all, $\forall j > j_0 : Player(j) = E \Rightarrow Turn_j = \emptyset$ because E has already nullified his incoming and outgoing direct trusts in $Turn_{j_0}$, the evil strategy does not contain any case where direct trust is increased or where the evil player starts directly trusting another player and the other players do not follow a strategy in which they can choose to $Add()$ trust to E, thus player E can do nothing $\forall j > j_0$. Also $\forall j > j_0 : Player(j) = A \Rightarrow Turn_j = \emptyset$ because of the idle strategy that A follows. As far as the rest of the players are concerned, consider the algorithm 8, which is a variation of the TrustIsRisk Game.
As we can see from lines 5 and 18 - 20, $\forall j, \sum_{v \in \mathcal{V}_j} Loss_v = in_{E,j_0-1}$, that is the total loss is constant and equal to the total value stolen by E. Also, as we can see in lines 3 and 29, which are the only lines where the Sad set is modified, once a player enters the Sad set, it is impossible to exit from this set. Also, we can see that players in $Sad \cup Happy$ always pass their turn. We will now show that eventually the $Angry$ set will be empty, or equivalently that eventually every player will pass their turn. Suppose that it is possible to have an infinite amount of turns that players do not choose to pass. We know that the number of nodes is finite, thus this is possible only if $\exists j_1 : \forall j \geq j_1, |Angry_j \cup Happy_j| = c > 0 \wedge Angry_j \neq \emptyset$ (the total number of angry and happy players cannot increase because no player leaves the Sad set and if it were to be decreased, it would eventually reach 0). Since $Angry_j \neq \emptyset$, a player v that will not pass her turn will eventually be chosen to play. According to algorithm 8, v will either deplete her incoming trust and enter the Sad set (line 29), which is contradicting $|Angry_j \cup Happy_j| = c$, or will steal enough value to enter the $Happy$ set, that is v will achieve $Loss_{v,j} = 0$. Suppose that she has stolen m players. They, in their turn, will steal total value at least equal to the value stolen by v (since they cannot go sad, as explained above). However, this means that, since the total value being stolen will never be reduced and the turns this will happen are infinite, the players must steal an infinite amount of value, which is impossible because the direct trusts are finite in number and in value. More precisely, let $\forall j \in \mathbb{N}, DTr_j = \sum_{w,w' \in \mathcal{V}} DTr_{w \to w',j}$. Also, without loss of generality, suppose that $\forall j \geq j_1, out_{A,j} = out_{A,j_1}$.
In $Turn_{j_1}$, v steals $St_{j_1} = \sum_{i=1}^{m} y_i$. Thus $DTr_{j_1} = DTr_{j_1-1} - St_{j_1}$. Eventually there is a turn j_2 when every

player in $N^-(v)$ will have played. Then $S_{j_2} \leq DTr_{j_1} - St_{j_1} = DTr_{j_1-1} - 2St_{j_1}$, since all players in $N^-(v)$ follow the conservative strategy, except maybe for A, who will not have been stolen anything due to the supposition.
Suppose that $\exists k > 1 : j_k > j_{k-1} > j_1 \Rightarrow DTr_{j_k} \leq DTr_{j_{k-1}} - St_{j_1}$. Then there exists a subset of the *Angry* players, S, that have been stolen at least value St_{j_1} in total between the turns j_{k-1} and j_k, thus there exists a turn j_{k+1} such that all players in S will have played and thus $DTr_{j_{k+1}} \leq DTr_{j_k} - St_{j_1}$. We have proven by induction that $\forall n \in \mathbb{N}, \exists j_n \in \mathbb{N} : DTr_{j_n} \leq DTr_{j_1-1} - nSt_{j_1}$. However $DTr_{j_1-1}, St_{j_1} \in \mathbb{N}$, thus $\exists n' \in \mathbb{N} : n'St_{j_1} > DTr_{j_1-1} \Rightarrow DTr_{j'_n} < 0$. We have a contradiction because $\forall w, w' \in \mathcal{V}, \forall j \in \mathbb{N}, DTr_{w \to w',j} \geq 0$, thus eventually $Angry = \emptyset$ and everybody passes. □

Theorem 6.2 (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 $\exists k \in [n] : x'_k < u'_k$ 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 $x'_k < y_k$ and (1) which is impossible because the configuration $\forall i \in [n], x'_i = y_i$ is valid since $\forall i \in [n], y_i = u'_i$ and also has a higher flow, thus the maxFlow algorithm will prefer the configuration with the higher flow. Thus we deduce that $\forall i \in [n], x'_i = u'_i$. □

Let G be a weighted directed graph. According to [8] p. 709, if we consider each edge's capacity as its weight ($\forall e \in E(G), c_e = c(e)$), we say that a flow assignment $X = [x_{vw}]_{V(FG) \times V(FG)}$ with a source A and a sink B is valid when:

1. $\forall (v, w) \in E(FG), x_{vw} \leq c_{vw}$
2. $\forall v \in V(FG) \setminus \{A, B\}, \sum_{w \in N^+(v)} x_{wv} = \sum_{w \in N^-(v)} x_{vw}$
3. $\sum_{v \in N^+(A)} x_{Av} = \sum_{v \in N^-(B)} x_{vB}$

Theorem 6.3 (Trust flow theorem - TOCHECK)**.**
Let $\mathcal{G}$ be a game graph at a specific turn and $A, B \in V(\mathcal{G})$. It holds that $Tr_{A \to B} = MaxFlow(A, B)$

Proof.
Without loss of generality, we can suppose that the turn in which we are interested is 0 ($\mathcal{G} = \mathcal{G}_0$). First we will show that the $MaxFlow$ can be a result of a valid execution of algorithm 8 and afterwards we will show that each valid execution of algorithm 8 corresponds to a valid flow from A to B. Thus we will have proven that $Tr_{A \to B} = MaxFlow(A, B)$.

- We will first show that there exists an execution of algorithm 8 such that $Loss_A = maxFlow(A, B)$. Let X be the flows as returned by an execution of the $maxFlow(A, B)$ algorithm on $\mathcal{G}$. It is known that all flows are DAGs [citation needed] and that all DAGs are a partial order of their nodes based on the partial ordering $x_{vw} > 0 \Rightarrow v < w$ [citation needed]. From this partial order, we can create a total order with an algorithm such as topoSort [7]. The maximum element of the total order is a node that does not have any outgoing flow. Removing any node from a DAG cannot create a cycle, thus the graph that remains after removing a node from a DAG is also a DAG, thus it has a total order as well, which can be chosen to be the same total order as before removing the node, except for the removed node. If the removed node was maximum or minimum, the new total order is obvious. We will prove our claim using induction.

- Player B is the maximum node in turn 0 because she is the sink of the MaxFlow algorithm, thus she is the first to be chosen to play and steals all her incoming and outgoing trust. $\forall v \in N^-(B)_0, x_{vB} \leq DTr_{v\to B,0}$ and $\sum_{v\in N^-(B)_0} x_{vB} = maxFlow(A,B)$. The graph $\mathcal{G}_1 = \mathcal{G}_0 \setminus \{B\}$ is also a DAG and corresponds to the previous total order if we remove the maximum element, B.
- Suppose that $\forall j \in [k], k > 0$, the player v corresponding to the maximum element is chosen to play for the first time, that $\forall w \in N^-(v)_{j-1} (= N^-(v)_0), x_{wv} \leq y$ where $Steal(y,w) \in Turn_k \wedge \sum_{w\in N^-(v)_0} x_{wv} = \sum_{w\in N^+(v)_0} x_{vw}$.
- For $j = k+1$, $Player(k+1) = v'$ corresponds to the maximum element of the previous total order with the element v removed and it is the first time player v' plays, since $v > v'$ in all previous steps thus v' was not maximum. It also holds that $\forall w \in N^-(v)_0, x_{wv'} \leq DTr_{w\to v',0}$ since the $x_{wv'}$ are chosen by the maxFlow algorithm with corresponding capacities the direct trusts and, since $\sum_{w\in N^-(v')_0} x_{wv'} = \sum_{w\in N^+(v')_0} x_{v'w}$ and player v' has already been stolen value equal to $\sum_{w\in N^+(v')_0} x_{v'w}$ (since she has no outgoing flow in turn j), player v' can choose to steal from each player $w \in N^-(v')_0$ value at least equal to $x_{wv'}$ without violating the conservative strategy.

We have proven using induction that if the algorithm chooses only maximum nodes, after exactly $|V(\mathcal{G}_0)| - 1$ turns (we do not count idle player A) every player except for A will have stolen at least value equal to the flow passing through them and player A will have been stolen value exactly equal to $maxFlow(A,B) \Rightarrow Loss_A = maxFlow(A,B)$.

- We will now show that for any valid execution of algorithm 8 there exists at least one valid flow from A to B, such that $Loss_A = \sum_{v\in N^+(A)_0} x_{Av}$. Let j be a turn where 8 has converged (j exists, according to theorem 6.1). Then $Loss_{A,j} = out_{A,0} - out_{A,j}$. We create a new graph $\mathcal{G}'$ such that $V(\mathcal{G}') = V(\mathcal{G}) \cup T, E(\mathcal{G}') = E(\mathcal{G}) \cup \{(T,v) : v \in Sad_j\} \cup (T,A), \forall (v,w) \in E(\mathcal{G}), c'_{vw} = DTr_{v\to w,0} - DTr_{v\to w,j}, \forall v \in Sad_j, c'_{Tv} = c'_{TA} = \infty$ (T is an auxiliary source that trusts infinitely A and all the Sad nodes). We execute the $MaxFlow(T,B)$ algorithm on $\mathcal{G}'$ and we get a flow $X' : \forall v,w \in V(\mathcal{G}), x'_{vw} = c'_{vw}$ (all the edges, except for the auxiliary ones, saturated). Thus $\sum_{v\in N^+(A)_0} x_{Av} = Loss_{A,j}$. If we create a new graph $\mathcal{G}''$ with $V(\mathcal{G}'') = V(\mathcal{G}'), E(\mathcal{G}'') = E(\mathcal{G}'), \forall v \in Sad_j, c''_{Tv} = 0, c''_{TA} = \infty, \forall v,w \in V(\mathcal{G}), c''_{vw} = c_{vw}$ (the auxiliary node trusts only A) and execute the $MaxFlow(T,B) = X''$ algorithm on $\mathcal{G}''$, $\sum_{v\in N^+(A)_0} x''_{Av} = \sum_{v\in N^+(A)_0} x'_{Av}$ (the outgoing flow from A will remain the same as in $\mathcal{G}'$) since no capacity accesible from A has been modified (the only changed capacities are those that begin from T and there is no incoming flow to T) thus $\sum_{v\in N^+(A)_0} x''_{Av} \geq \sum_{v\in N^+(A)_0} x'_{Av}$ and $\sum_{v\in N^+(A)_0} x'_{Av} = \sum_{v\in N^+(A)_0} c'_{Av} = \sum_{v\in N^+(A)_0} c''_{Av}$ (the outgoing edges from A were already saturated) thus $\sum_{v\in N^+(A)_0} x''_{Av} \leq \sum_{v\in N^+(A)_0} x'_{Av}$. Thus the resulting flow is equal to $Loss_{A,j}$, or $\sum_{v\in N^+(A)_0} x''_{Av} = \sum_{v\in N^+(A)_0} c_{Av} = Loss_{A,j}$.

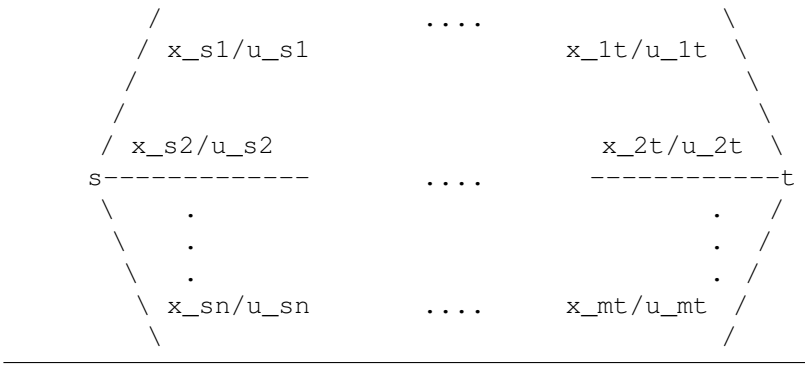
We finally conclude that $Tr_{A\to B} = MaxFlow(A,B)$. □

Theorem 6.4 (Conservative world theorem)**.**
If everybody follows the conservative strategy, nobody steals any amount from anybody.

Proof.
Suppose that there exists a subseries of History, $(Turn_{j_k})$, where $Turn_{j_k} = \{Steal(y_1, B_1), ..., Steal(y_m, B_m)\}$. This subseries must have an initial element, $Turn_{j_1}$. However, $Player(j_1)$ follows the conservative strategy, thus somebody must have stolen from her as well, so $Player(j_1)$ cannot be the initial element. We have a contradiction, thus there cannot exist a series of stealing actions when everybody is conservative. □

Theorem 6.5 (Trust transfer theorem (flow terminology))**.**
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 conservative 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 6.2 we can see that $x'_i = u'_i$. It holds that $F' = \sum_{i=1}^{n} x'_i = \sum_{i=1}^{n} u'_i = F - V$. □

Lemma 6.2 (Flow limit lemma)**.**
It is impossible for the outgoing flow x_i *from* A *to an out neighbour of* A *to be greater than* $F_{A_i \to B}$. *More formally,* $x_i \leq F_{A_i \to B}$.

Proof. Suppose a configuration where $\exists i : x_i > F_{A_i \to B}$. If we reduce the capacities $u_k, k \neq i$ the flow that passes from i in no case has to be reduced. Thus we can set $\forall k \neq i, u'_k = 0$ and $u'_i = u_i$. Then $\forall k \neq i, x'_k = 0, x'_i = x_i$ is a valid configuration and thus by definition $F_{A_i \to B} = x'_i = x_i > F_{A_i \to B}$, which is a contradiction. Thus $\forall i \in [|N^+(A)|], x_i \leq F_{A_i \to B}$. □

Theorem 6.6 (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* $U'' : u''_i = u_i, \forall k \in [|N^+(A)|], k \neq i, u''_k = u'_k$.

Proof. We know that $x_i \leq F_{A_i \to B}$ (lemma 6.2), 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 6.7 (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 6.2) $x'_i = u'_i$, 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 6.8 (Dependence impossibility theorem)**.**
$\frac{\partial x_k}{\partial x_i} = 0$ *with* x_i *the flow from* $MaxFlow \Rightarrow \forall x'_i \leq x_i, \frac{\partial x_k}{\partial x_i} = 0$ *ceteris paribus*

Proof. TODO □

Note: The maxFlow is the same in the following two cases: When a player chooses the evil strategy and when the same player chooses a variation of the evil strategy where she does not nullify her outgoing direct trust.

Theorem 6.9 (Trust to multiple players)**.**
Let $S \subset \mathcal{V}, T$ auxiliary player such that $\forall B \in S, DTr_{B\to T} = \infty$. It holds that $\forall A \in \mathcal{V} \setminus S, Tr_{A\to S} = maxFlow(A, T)$.

Proof. If T chooses the evil strategy and all players in S play according to the conservative strategy, they will have to steal all their incoming direct trust, thus they will act in a way identical to following the evil strategy as far as maxFlow is concerned, thus, by 6.3, $Tr_{A\to T} = maxFlow(A, T) = Tr_{A\to S}$. □

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 $\mathcal{C}$, or Sybil set. 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 $P \in \mathcal{V}$ ($DTr_{C\to P}$) and can also steal all incoming direct trust to these players. Additionally, we give Eve a set of players $B \in \mathcal{B}$ that she has corrupted (the corrupted set), so she fully controls their direct trusts to any player $P \in \mathcal{V}$ ($DTr_{B\to P}$) and can also steal all incoming direct trust to these players. The players $B \in \mathcal{B}$ are considered to be legitimate before the corruption, thus they can be directly trusted by any player $P \in \mathcal{V}$ ($DTr_{P\to B} \geq 0$). 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 $A \in \mathcal{V} \setminus (\mathcal{B} \cup \mathcal{C})$ ($DTr_{A\to C} = 0$).

Lemma 6.3 ($\forall S \subset \mathcal{V}, v \in S \Rightarrow \forall w \in \mathcal{V}, (v, w) \notin MinCut$)**.**
Let $S \subset \mathcal{V}$. When calculating $maxFlow(A, S)$, it is impossible to have an edge $(v, w) \in MinCut : v \in S$.

Proof. Let T be the auxiliary node ($\forall v \in S, c_{vT} = \infty$). Since $out_A < \infty$, $maxFlow(A, S) < \infty$. All edges in the $MinCut$ are saturated, thus $\nexists v \in S : (v, T) \in MinCut$, or else $maxFlow(A, S) = \infty$. Suppose that $\exists v \in S, w \in \mathcal{V} : (v, w) \in MinCut$. Then this edge must be saturated, that is $x_{vw} = c_{vw} > 0$. However, there exists an alternative flow configuration X' where $\forall (u, u') \in \mathcal{E} \setminus \{(v, w), (v, T)\}, x'_{u,u'} = x_{u,u'}, x'_{vw} = 0, x'_{vT} = x_{vT} + x_{vw}$, which is valid because $\sum_{w\in N^+(v)} x_{vw} = \sum_{w\in N^+(v)} x'_{vw} \wedge c_{vT} = \infty \Rightarrow x'_{vT} \leq c_{vT}$ and X' is maximum as well because it carries exactly the same flow as X. Thus $(v, w) \notin MinCut$. □

Theorem 6.10 (Sybil resilience)**.**
Let $\mathcal{B} \cup \mathcal{C} \subset \mathcal{V}$ ($\mathcal{B} \cap \mathcal{C} = \emptyset$) be a collusion of players who are controlled by an adversary, Eve. Eve also controls the number of players in the Sybil set $\mathcal{C}, |\mathcal{C}|$, but players $C \in \mathcal{C}$ are not directly trusted by players outside the collusion, contrary to players $B \in \mathcal{B}$, the corrupted set, who may be directly trusted by any player in $\mathcal{V}$. It holds that $Tr_{A\to\mathcal{B}\cup\mathcal{C}} = Tr_{A\to\mathcal{B}}$.

Proof. Suppose two separate games $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1), \mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ with $(\mathcal{B} \subset \mathcal{V}(\mathcal{G})_1 \wedge \mathcal{B} \subset \mathcal{V}(\mathcal{G})_2) \wedge (\mathcal{C} \subset \mathcal{V}_1 \wedge \mathcal{C} \cap \mathcal{V}_2 = \emptyset)$. Let $T_1 \in \mathcal{V}_1$ be an auxiliary player such that $\forall D \in \mathcal{B}\cup\mathcal{C}, DTr_{D\to T_1} = \infty$ and $T_2 \in \mathcal{V}_2$ be another auxiliary player such that $\forall B \in \mathcal{B}, DTr_{B\to T_2} = \infty$. Suppose also that there exist $|\mathcal{B} \cup \mathcal{C}|$ consecutive turns for the first game and $|\mathcal{B}|$ consecutive turns for the second game during which all the colluding players choose actions according to the evil strategy. More formally, suppose that $\exists j \in \mathbb{N} : \forall d_1 \in [|\mathcal{B} \cup \mathcal{C}|], Player(j + d) \in \mathcal{B}\cup\mathcal{C} \wedge \forall d_1, d_2 \in [|\mathcal{B}\cup\mathcal{C}|], d_1 \neq d_2, Player(j+d_1) \neq Player(j+d_2) \wedge \forall d \in [|\mathcal{B}\cup\mathcal{C}|], Strategy(Player(j+d)) = Evil \wedge \forall j' \in [j]Player(j') \notin \mathcal{B}\cup\mathcal{C}$ for the first and likewise for the second game. According to 6.9, $Tr_{A\to\mathcal{B}\cup\mathcal{C}} = maxFlow(A, T_1), Tr_{A\to\mathcal{B}} = maxFlow(A, T_2)$. From lemma 6.3, we know that $\forall (v, w) \in MinCut, v \notin \mathcal{B} \cup \mathcal{C}$ and thus $e \in MinCut_1 \Rightarrow e \in \mathcal{E}_2 \wedge e \in MinCut_2 \Rightarrow e \in \mathcal{E}_1 \wedge \forall e \in MinCut_1 \cup MinCut_2, c_1(e) = c_2(e)$. Thus the flow X_1 resulting from $maxFlow(A, T_1)$ can be used to construct a valid flow of equal value for the second case if we set $\forall v \in \mathcal{V}_2 \setminus \mathcal{B}, \forall w \in \mathcal{V}_2, x_{vw,2} = x_{vw,1} \wedge \forall v \in \mathcal{B}, x_{vT_2,2} = \sum_{w\in N^+(v)} x_{vw,1} \wedge \forall v, w \in \mathcal{B} \cup \mathcal{C}, x_{vw,2} = 0$. Likewise, the flow X_2 resulting from $maxFlow(A, T_2)$ can be used to construct a valid flow of equal value for the first case if we set $\forall v \in \mathcal{V}_1 \setminus (\mathcal{B} \cup \mathcal{C}), \forall w \in \mathcal{V}_1, x_{vw,1} = x_{vw,2} \wedge \forall v \in \mathcal{B}, x_{vT_1,1} = \sum_{w\in N^+(v)} x_{vw,2} \wedge \forall v \in \mathcal{C}, \forall w \in \mathcal{V}_1, x_{vw,1} = 0$. From these two observations, we deduce that there exists a function, say $F_2(X_1)$, that transforms the $MaxFlow_1$ of the first graph into a valid flow for the second graph that has the

same amount of flow as $MaxFlow_1$ and there also exists a similar function $F_1(X_2)$ that transforms the $MaxFlow_2$ of the second graph into a valid flow for the first graph that has the same amount of flow as $MaxFlow_2$. Suppose that $maxFlow_1 < maxFlow_2$. Then $F_1(MaxFlow_2) > maxFlow_1$ which is a contradiction. The same contradiction arises if we suppose that $maxFlow_1 > maxFlow_2$. We conclude that $maxFlow(A, T_1) = maxFlow(A, T_2) \Rightarrow Tr_{A\to\mathcal{B}} = Tr_{A\to\mathcal{B}\cup\mathcal{C}}$. □

We have proven that controlling $|\mathcal{C}|$ is irrelevant for Eve, thus Sybil attacks are meaningless.

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 9: 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** $U' = \bigcup_{k=1}^{n} \{u'_k\}$

Proof of correctness for algorithm 9.

- 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 $\forall k < i, F_{cur,k} = F_{cur,k-1} - x_k \Rightarrow F_{cur,i} = F - \sum_{k=1}^{i} x_k$
  Furthermore, if $F_{cur,i+1} = F - V$ then $u'_{i+1} = x_{i+1} - F_{cur,i} + F - V = x_i - F + \sum_{k=1}^{i-1} x_k + F - V = \sum_{k=1}^{i} x_k - V$, $\forall k \leq i, u'_k = 0$ and $\forall k > i+1, u'_k = x_k$.
  In total, we have $\sum_{k=1}^{n} u'_k = \sum_{k=1}^{i} x_k - V + \sum_{k=i+1}^{n} x_k = \sum_{k=1}^{n} x_k - V \Rightarrow \sum_{k=1}^{n} u'_k = F - V$.

□

Complexity of algorithm 9.
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 9 is $O(n)$. □

Algorithm 10: 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** $U' = \bigcup_{k=1}^{n} \{u'_k\}$

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 10 (think of i from line 21 without the $mod\, n$).
First we will show that $empty \leq n$. $empty$ is only modified on line 13 where it is incremented by 1. This happens only when $u'_i > 0$ (line 11), which is assigned the value 0 on line 17. 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 $r = \frac{V - \sum\limits_{x \in S} x}{n - |S|}$ where $S = \{\text{flows } y \text{ from } s \text{ to } N^+(s) \text{ according to } maxFlow : y < r\}$. Also, $\sum\limits_{i=1}^{n} u'_i = \sum\limits_{i=1}^{n} \max(0, x_i - r)$. TOPROVE

Proof of correctness for algorithm 10.

- We will show that $\forall i \in [n]\ u'_i \leq x_i$.
  On line 5, $\forall i \in [n]\ u'_i = x_i$. Subsequently u'_i is modified on line 17, where it becomes equal to 0 and on line 20, 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 15 where it is increased ($n > empty$ because of line 14 and $reduce > x_i$ because of line 12, thus $\frac{reduce - x_i}{n - empty} > 0$). We see that $\forall i \in [n], u'_i \leq x_i$.

- We will show that $\sum\limits_{i=1}^{n} u'_i = F - V$.
  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 $reduction = \sum\limits_{i=1}^{n}(x_i - u'_i)$ at all times and then we will prove that $reduction = V$ at the end of the execution. Thus we will have proven that $\sum\limits_{i=1}^{n} u'_i = \sum\limits_{i=1}^{n} x_i - V = F - V$.

  - On line 5, $u'_i = x_i \Rightarrow \sum\limits_{i=1}^{n}(x_i - u'_i) = 0$ and $reduction = 0$.
    On line 17, u'_i is reduced to 0 thus $\sum\limits_{i=1}^{n}(x_i - u'_i)$ is increased by u'_i. Similarly, on line 16 $reduction$ is increased by u'_i, the same as the increase in $\sum\limits_{i=1}^{n}(x_i - u'_i)$.
    On line 20, u'_i is reduced by $u'_i - x_i + reduce$ thus $\sum\limits_{i=1}^{n}(x_i - u'_i)$ is increased by $u'_i - x_i + reduce$. On line 19, $reduction$ is increased by $u'_i - x_i + reduce$, which is equal to the increase in $\sum\limits_{i=1}^{n}(x_i - u'_i)$. 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 $reduction = \sum\limits_{i=1}^{n}(x_i - u'_i)$ at all times.
  - Suppose that $reduction_j > V$ on the line 22. 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 10.
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 10 minimizes the $||\Delta_i||_\infty$ norm.
Suppose that U' is the result of an execution of algorithm 10 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 11: 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** $U' = \bigcup_{k=1}^{n} \{u'_k\}$

Proof of correctness for algorithm 11.

- 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 11.
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 11 is $O(n)$. □

Naive algorithms result in $u'_i \leq x_i$, thus according to 6.7, $||\delta_i||_1$ 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 12: $||\delta_i||_\infty$ 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** $U' = \bigcup_{k=1}^{n} \{u'_k\}$

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 12: 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** $i \leftarrow 1$ *to* n **do**
5 | | $u'_i \leftarrow \max(0, u_i - \frac{top+bot}{2})$
6 | **if** $maxFlow < F' - \epsilon_1$ **then**
7 | | **return** `BinSearch` $(bot, \frac{top+bot}{2}, F', n, X, \epsilon_1, \epsilon_2)$
8 | **else if** $maxFlow > F' + \epsilon_2$ **then**
9 | | **return** `BinSearch` $(\frac{top+bot}{2}, top, F', n, X. \epsilon_1, \epsilon_2)$
10 | **else**
11 | | **return** $\frac{top+bot}{2}$

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 13.
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 13.
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 13, we need at most $\log_2(\frac{top - bot}{\epsilon_1 + \epsilon_2})$ recursive calls of `BinSearch`. Thus the function 13 has worst-case complexity $O((maxFlow + n)\log_2(\frac{top - bot}{\epsilon_1 + \epsilon_2}))$. □

Proof of correctness for algorithm 12.
We will show that $maxFlow \in [F - V - \epsilon_1, F - V + \epsilon_2]$, with u'_i decided by algorithm 12.
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 13, 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 12.
The complexity of lines 1 - 2 and 4 - 5 is $O(1)$, the complexity of lines 3, 6, 8 - 9 and 10 is $O(n)$ and the complexity of line 7 is $O(BinSearch) = O((maxFlow + n)\log_2(\frac{\delta_{max}}{\epsilon_1 + \epsilon_2}))$, thus the total complexity of algorithm 12 is $O((maxFlow + n)\log_2(\frac{\delta_{max}}{\epsilon_1 + \epsilon_2}))$. □

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

7 Related Work

8 Further Research

While our trust network can form a basis for risk-invariant transactions in the anonymous and decentralized setting, more research is required to achieve other desirable properties. Some directions for future research are outlined below.

8.1 Zero knowledge

Our network evaluates indirect trust by computing the max flow in the graph of lines-of-credit. In order to do that, complete information about the network is required. However, disclosing the network topology may be undesirable, as it subverts the identity of the participants even when participants are treated pseudonymously, as deanonymization techniques can be used. To avoid such issues, exploring the ability to calculate flows in a zero knowledge fashion may be desirable. However, performing network queries in zero knowledge may allow an adversary to extract topological information. More research is required to establish how flows can be calculated effectively in zero knowledge and what bounds exist in regards to information revealed in such fashion.

9 References

[1] Distributed Denial of Service Attacks - The Internet Protocol Journal - Volume 7, Number 4

[2] Standard ebay selling fees

[3] ebay money back guarantees

[4] What is OpenBazaar

[5] Can bitcoin and multisig reduce identity theft and fraud?

[6] Nakamoto, Satoshi (2008). Bitcoin: A Peer-to-Peer Electronic Cash System

[7] Kahn, Arthur B. (1962), "Topological sorting of large networks", Communications of the ACM

[8] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009) [1990]. Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill.

[9] Zindros, Dionysis S. (2015). Trust in decentralized anonymous marketplaces

[10] Bitcoin Magazine (2014). Bitcoin Multisig Wallet: The Future of Bitcoin