Description

The objective of this project is to improve the results presented in the "On Bitcoins and Red Balloons" paper. For this, I'm going to review the main concepts, theorems and results shown in the paper and show how to apply them to a more general model than its authors assume.

Motivation – The Bitcoin Protocol

Bitcoin is a decentralized electronic currency system, that relies on a peer-to-peer network to verify and authorize each of its transactions. Suppose that Alice wants to transfer 10 bitcoins to Bob. In order to do this, Alice creates and signs a digital transaction to transfer 10 bitcoins to Bob. Alice will send this transaction to a small number of nodes in the networks. Each node that receives Alice's transaction will do the following:

1. Relay it to its neighbours.
2. Verify that Alice indeed has signed the transaction and that she does indeed the amount of bitcoins she is trying to transfer.
3. Try to authorize the transaction by solving a computational hard problem – inverting a hash function.

Once a node successfully authorizes a transaction, it sends notifies of it to all its neighbours, which will also relay this information to their own neighbours and so on. Once all nodes in the network are aware of the transaction, they can all agree that 10 of the bitcoins that Alice owned have now been transferred to Bob.

In order to encourage nodes to try to authorize transactions, they are offered payments in bitcoins for successful authorizations. This means that all the nodes in the network that are aware of a transaction will compete with each other for the reward that comes with the successful authorization of the transaction. With this in mind, it's not clear whether a node that is notified of a transaction will relay it to its neighbours. Quite the opposite, a node that learns of a transaction has an incentive to keep the information to itself - this will reduce the potential competition and increase its chances of authorizing the transaction first and receive its associated reward. The problem with this behaviour is that the expected amount of time that it takes to authorize a transaction depends on the number of nodes working on it: the more nodes are trying to authorize a transaction, the less time it will take for it to be authorized. In the current bitcoin implementation, the time that a single node is expected to authorize a transaction is 60 days. On the other hand, if every node in the network is competing, this is expected to take 10 minutes.

As noted above, the way Bitcoin is specified doesn't provide an incentive to its nodes to broadcast transactions that they are aware of. In the paper "On Bitcoins and Red Balloons" the authors augment the protocol and propose a Sybil-proof scheme that rewards information propagation.

The Model

We assume that there is only one transaction waiting to be authorized. The authorization process is divided in two phases:

1. Distribution phase, where each node that learns of the transaction may or may not send it to its neighbours.
2. Computation phase, in which every node that has learned of the transaction will try to authorize it.

We also assume that the network consists of a forest of t d-ary directed trees, each of them of height H.

In the distribution phase the t roots are made aware of the transaction. The information of the transaction flows from the root towards the leaves. Each node v can send the transaction to any of its direct descendants after adding any number of fake identities of itself (clones).

In the distribution phase each node that is aware of the transaction will try to authorize it. The paper assumes that every node in the network has the same processing power. Therefore if there are k nodes that are aware of transaction, every one of them has the same probability of $\frac{1}{k}$ to be the one who authorizes it first. Here is where what I'm going to show in this project comes into play. Rather than assuming that every node has the same processing power, I'll assume that only nodes that are in the same tree have the same processing power – nodes that belong to different trees might have different processing power. If a node v that is aware of the transaction has processing power CPU_v then the we'll define the probability that v authorizes the transaction first as $\frac{CPU_v}{\sum_u CPU_u}$, where $\sum_u CPU_u$ includes all the nodes that are aware of the transaction.

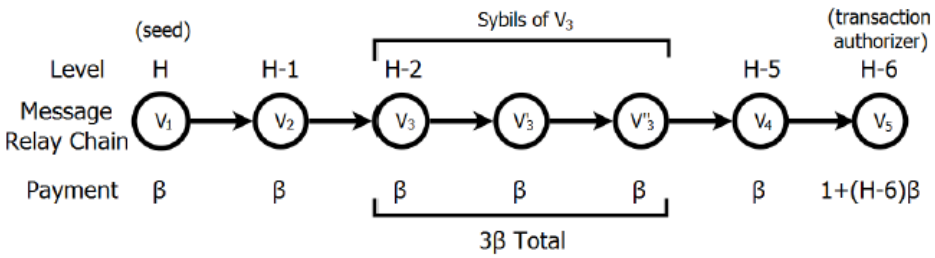
Note that fake identities don't increase the processing power of a node, and therefore they don't increase the probability of receiving the reward. A node v that authorizes the transaction can declare it did so with any identity p_h it created. This determines a chain of identities $p_1, \dots, p_h$, which is called the *authorizing chain*. The chain starts with p_1, an identity of the seed that roots v's tree and ends with p_h, the declared identity of the authorizer node. This chain is a superset of the real path in the tree, potentially including clones of some nodes (nodes are not able to remove predecessor identities from the chain due to the use of cryptographic signatures). Rewards are allocated to the identities that appear in the authorizing chain. The reward each node receives is the sum of the rewards of all its identities (true and fake).

New Reward Schemes

Almost-Uniform Schemes

The (β, H)-almost-uniform scheme pays the authorizing node in a chain of length h a reward of $1 + \beta \times (H - h + 1)$. The rest of the nodes in the chain get a reward of β. If the chain length is greater than H no node receives a reward. The idea behind giving the authorizing node a reward

of $1 + \beta \times (H - h + 1)$ is to mimic the reward that the node would have gotten if it duplicated itself $H - h$ times. Therefore, we can assume that no node duplicates itself before trying to authorize the transaction, and focus only on duplication before sending to its children. Also note that the total payment is always $1 + \beta \times H$.



Theorem 1: Let $d \geq 3$. For every node v we define CPU_{-v} to be the combined processing power of all the nodes that are aware of the transaction that are not v nor descendants of v. Suppose that for every node v in the network it holds that $CPU_{-v} > CPU_v^2 \cdot (2\beta^{-1} + 16) + CPU_v \cdot (d^{H+1} \cdot 4)$. In the (β, H)-almost-uniform scheme only profiles of strategies in which every node of depth at most H never duplicates and always propagates information survive in every iterated removal of weakly dominated strategies.

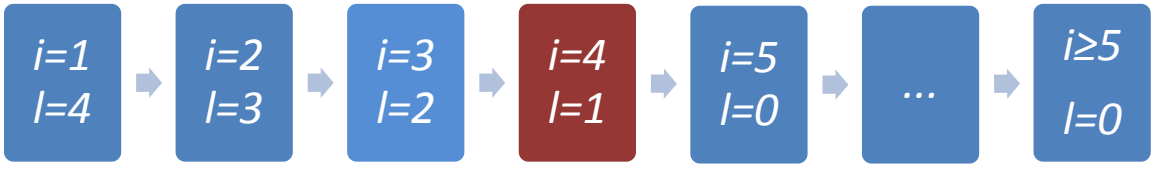
Proof

The proof of the theorem is divided in the following 3 sections:

1. Notation and definition of key concepts for the (β, H)-almost-uniform scheme
2. Definition of φ-subgames
3. Proofs of an extra lemma
4. Actual Proof of the Theorem

1. Notation and definition of key concepts for the (β, H)-almost-uniform scheme

A node v that receives information from his parent observes a chain of identities $p_1, p_2, \ldots, p_h$, ending with its own identity ($p_h = v$) which follows its parent's identity (p_{h-1} is v's parent if v is not a seed. Otherwise the chain has only one element $p_1 = v$). The chain $p_1, p_2, \ldots, p_h = v$ which v observes defines v's level $l(v)$, which is the maximal number of identities (true and fake) v can add in any completion of the observed chain to a chain of length H. In other words, how many identities the node needs to add to the chain it sees so that its size will be H. Thus, for $i \leq H$ the level of v is $l(v) = H - i + 1$ (note that the level of each seed is defined to be H). For $i > H$ we define $l(v) = 0$. For example if $H = 4$ then the level of each node in the following chain would be:



A strategy for a node can only depend on the length of the chain it observes. We can also think of *v*'s strategy as a function of its level $l(v)$.

When a node v receives the transaction, it decides whether it wants to send the transaction onward, as well as the number of times it will duplicates itself before sending. This decision is made separately with respect to each child. One additional decision is whether the node wants to duplicate itself after authorizing the transaction. Both decisions are based solely on *v*'s level. For each node v and level l, $c_i^{l,v}$ denotes the number of times v clones itself before sending the transaction to its i'th child. Also $p^{l,v}$ denotes the number of times a node duplicates itself before it tries to authorize the transaction.

After having defined $c_i^{l,v}$ it will be useful to define $y_i^{l,v} = l - c_i - 1$. This is the actual space left in the chain after the nodes has cloned itself c_i times (includes its real identify and the c_i fake identities).

The strategy S_v of a node v is represented as follows: for every $1 \leq l \leq H$, there is a tuple $S_v(l) = (c_1^{l,v}, \dots, c_d^{l,v})$ where $0 \leq c_i^{l,v} \leq l - 1$ and a number $0 \leq p^{l,v} \leq l - 1$. If node v of level l decides not to send the transaction to its i's child it can implement this by setting $c_i^{l,v} = l - 1$.

We say that a node v propagates information and does not duplicate at level l if $S_v(l) = (0, \dots, 0)$ and $p^{l,v} = 0$. We also say that a node v never duplicates and always propagates information, if for every level l node v propagates information and does not duplicate at level l.

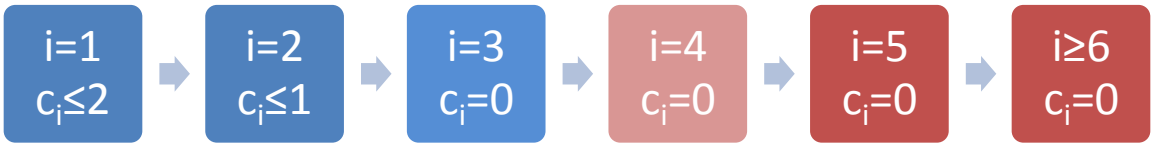
Given a strategy profile S and a reward scheme the utility of each player is defined as follows. An authorizer node w is chosen at random among the players that are aware of the transaction, according to the distribution of their processing power. Note that once we've fixed the strategies of all nodes, the chain each node v observes (if at all) is fixed. Let $p_1, \dots, p_2, \dots, p_h$ be the authorizing chain – it ends with the last identity of w and starts with $p_1 = s$ where s is the seed that roots w's tree. Denote $l = H - h + 1$. Then, we allocate rewards to all identities in the chain: w gets a reward of $r_{1,l}^S$, its predecessor gets $r_{2,l}^S$ and so on. The total reward a node receives is the sum of rewards of its true and fake identities in the successful chain.

2. Definition of φ-subgames

A φ-subgame is the (β, H)-game with restricted strategy spaces: the strategy space of node every node v includes only strategies such that for every remaining strategy profile of the rest of the players, every i and $l(v)$: $c_i^{l(v),v} \leq \max\{l(v) - \varphi - 1, 0\}$.

The intuition is that in a φ-subgame a node that has $H - l = H - (H - i + 1) = i - 1$ ancestors (including clones), does not clone itself and propagates information if $l \leq \varphi + 1$. Otherwise for every child it duplicates itself at most $l - \varphi - 1$ times.

The idea is that every node will leave space in the chain for at least φ more identities. For example if $H = 5$ and $\varphi = 2$:



Special case: if $\varphi = 0$ then $c_i^{l(v),v} \leq \max\{l(v) - 1, 0\}$. That is, every node can clone itself at most $l(v) - 1$ times. Note that $l(v)$ is the maximum number of identities a node can create. Assuming every node always creates at least one true identity, the limitation of cloning itself at most $l(v) - 1$ times is not really a limitation. Therefore a 0-subgame is simply the (β, H)-game.

3. Proof of an extra lemma

Lemma 1: Let $d \geq 3$. In the φ-subgame, for every node v we define CPU_{-v} to be the combined processing power of all the nodes that are aware of the transaction that are not v nor descendants of v. Suppose that for every node v in the network it holds that $CPU_{-v} > CPU_v^2 \cdot (2\beta^{-1} + 16) + CPU_v \cdot (d^{H+1} \cdot 4)$. Then, any strategy $\left(c_1^{l(v),v}, \dots, c_d^{l(v),v}\right)$ of node v such that some $c_i^{l(v),v} = l(v) - \varphi - 1$ is dominated by the strategy

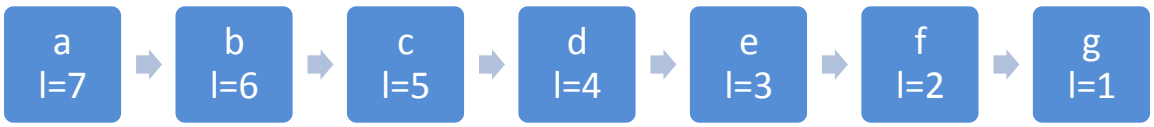
$$\left(c_1^{l(v),v}, \dots, c_{i-1}^{l(v),v}, c_i^{l(v),v} - 1, c_{i+1}^{l(v),v}, \dots, c_d^{l(v),v}\right).$$

Proof: Let v be a node at level $l = l(v)$. For convenience, we drop the superscript $(l(v), v)$ and denote a strategy by $(c_1, \dots, c_d)$. Without loss of generality we assume $c_1 \leq c_2 \leq \cdots \leq c_d$. That implies that $y_1 \geq y_2 \geq \cdots \geq y_d$.

Without loss of generality we'll prove that $(c_1, \dots, c_d)$, where $c_d = l(v) - \varphi - 1$ is dominated by $(c_1, \dots, c_d - 1)$.

Note that $(c_1, \dots, c_d)$ is equivalent to $(y_1, \dots, y_d)$ and $(c_1, \dots, c_d - 1)$ is equivalent to $(y_1, \dots, y_d + 1)$ as there is a one-to-one mapping between c_i and y_i.

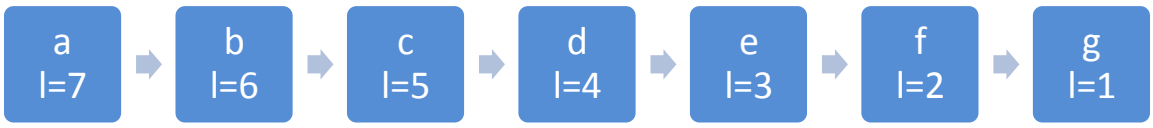
Note that if we assume that $c_d = l - \varphi - 1$, then $y_d = l - c_d - 1 = l - l + \varphi + 1 - 1 = \varphi$. Also if a node clones itself $c_d = l - \varphi - 1$ times, then none of its descendants will be able to clone itself and the node will span a subtree of height φ. Example: if H=7 and $\varphi = 2$



If node a clones itself $l - \varphi - 1 = 7 - 2 - 1 = 4$ times we'll have that:

The descendants of a will be b and c. Note that after a has clones itself 4 times, no other node will be able to clone itself. Take b for example, it can only clone itself at most $l_b - \varphi - 1 = 2 - 2 - 1 = -1 < 0$ times.

On the other hand if a node clones itself $c_d = l - \varphi - 2$ times, then none of its descendants will be able to clone itself either and the node will span a subtree of height $\varphi + 1$. Example: if H=7 and $\varphi = 2$



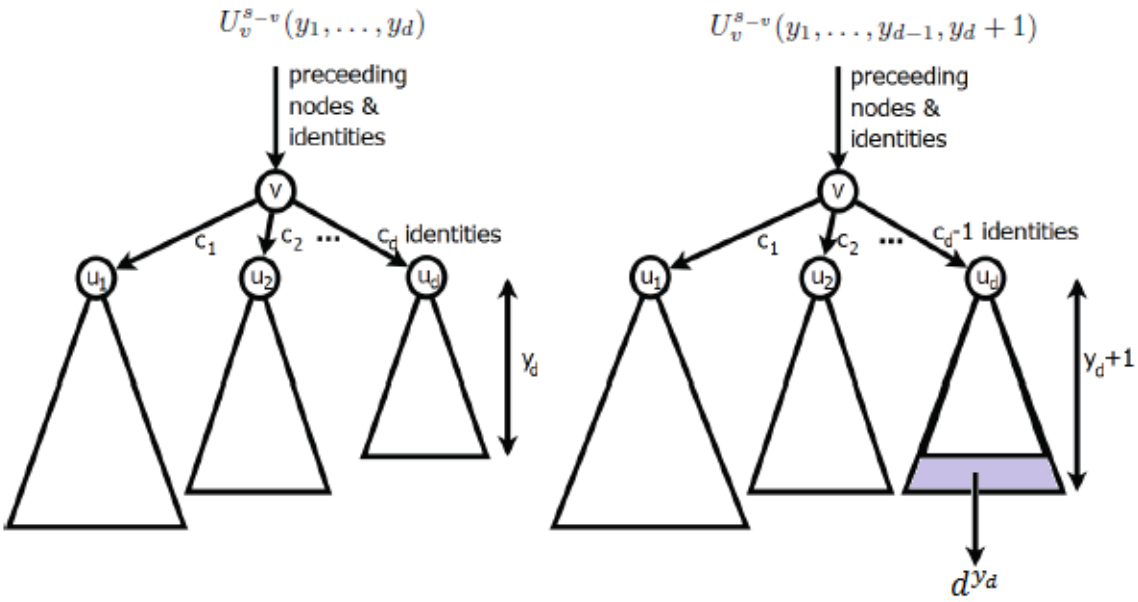
If node a clones itself $l - \varphi - 2 = 7 - 2 - 2 = 3$ times we'll have that:

The descendants of a will be b,c and d. Note that after a has cloned itself 3 times, no other node will be able to clone itself. Take b for example, it can only clone itself at most $l_b - \varphi - 1 = 3 - 2 - 1 = 0$ times.

Let S_{-v} be a strategy profile of all other nodes except v. Let's define the following variables:

- $A_{S_{-v}}(y_i)$ to be the size of the subtree rooted at the i'ths child of v.
- $CPU_{p_{A(y_i)}}$ to be the combined processing power of all the nodes in the subtree rooted at the i'ths child of v.
- $CPU_k = CPU_{-v}$ to be the combined processing power of all the nodes that are aware of the transaction that are not v nor descendants of v.
- CPU_{p_d} is the processing power of the extra d^{Y_d} nodes that are added to the subtree rooted at the d'ths child of v when it decides to clone itself one less time.

We need to show that $\forall S_{-v} \in T_{-v}\ U_v^{S_{-v}}(y_1, \dots, y_d + 1) > U_v^{S_{-v}}(y_1, \dots, y_d)$. The two scenarios are depicted in the image below:



We define:

$$U_v^{S-v}(y_1,\ldots,y_d)=\frac{CPU_v\cdot(1+\beta\cdot l)+\sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}}\cdot(\beta\cdot(l-y_i))\right)}{CPU_v+CPU_k+\sum_{i=1}^{d}CPU_{p_{A(y_i)}}}$$

$$U_v^{S-v}(y_1,\ldots,y_d+1)=\frac{CPU_v\cdot(1+\beta\cdot l)+\sum_{i=1}^{d-1}\left(CPU_{p_{A(y_i)}}\cdot(\beta\cdot(l-y_i))\right)+\left(CPU_{p_{A(y_d)}}+CPU_{p_d}\right)\cdot(\beta\cdot(l-y_d-1))}{CPU_v+CPU_k+\sum_{i=1}^{d}CPU_{p_{A(y_i)}}+CPU_{p_d}}$$

In both formulas, the denominators represent the combined processing power of all the nodes that are aware of the transaction. Notice that the total processing power increases in the second formula, as more nodes are aware of the transaction.

$$U_v^{S-v}(y_1,\ldots,y_d+1)>U_v^{S-v}(y_1,\ldots,y_d)\Leftrightarrow$$

$$\frac{CPU_v\cdot(1+\beta\cdot l)+\sum_{i=1}^{d-1}\left(CPU_{p_{A(y_i)}}\cdot(\beta\cdot(l-y_i))\right)+\left(CPU_{p_{A(y_d)}}+CPU_{p_d}\right)\cdot(\beta\cdot(l-y_d-1))}{CPU_v+CPU_k+\sum_{i=1}^{d}CPU_{p_{A(y_i)}}+CPU_{p_d}}$$

$$>\frac{CPU_v\cdot(1+\beta\cdot l)+\sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}}\cdot(\beta\cdot(l-y_i))\right)}{CPU_v+CPU_k+\sum_{i=1}^{d}CPU_{p_{A(y_i)}}}$$

Multiplying by the denominator in both sides:

$$\left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(CPU_v \cdot (1+\beta \cdot l\,) + \sum_{i=1}^{d-1}\left(CPU_{p_{A(y_i)}} \cdot (\beta \cdot (l - y_i))\right)\right.$$
$$\left. + \left(CPU_{p_{A(y_d)}} + CPU_{p_d}\right) \cdot (\beta \cdot (l - y_d - 1))\right)$$
$$> \left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}} + CPU_{p_d}\right)\left(CPU_v \cdot (1+\beta \cdot l\,)\right.$$
$$\left. + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot (\beta \cdot (l - y_i))\right)\right) \Leftrightarrow$$

$$\left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(CPU_v \cdot (1+\beta \cdot l\,) + \sum_{i=1}^{d-1}\left(CPU_{p_{A(y_i)}} \cdot (\beta \cdot (l - y_i))\right)\right)$$
$$+ \left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(\left(CPU_{p_{A(y_d)}} + CPU_{p_d}\right) \cdot (\beta \cdot (l - y_d - 1))\right)$$
$$> \left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(CPU_v \cdot (1+\beta \cdot l\,)\right.$$
$$\left. + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot (\beta \cdot (l - y_i))\right)\right)$$
$$+ CPU_{p_d}\left(CPU_v \cdot (1+\beta \cdot l\,) + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot (\beta \cdot (l - y_i))\right)\right) \Leftrightarrow$$

Dividing both sides by β:

$$\left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(CPU_v \cdot (\beta^{-1} + l\,) + \sum_{i=1}^{d-1}\left(CPU_{p_{A(y_i)}} \cdot (l - y_i)\right)\right)$$
$$+ \left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(\left(CPU_{p_{A(y_d)}} + CPU_{p_d}\right) \cdot (l - y_d - 1)\right)$$
$$> \left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(CPU_v \cdot (\beta^{-1} + l\,) + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot (l - y_i)\right)\right)$$
$$+ CPU_{p_d}\left(CPU_v \cdot (\beta^{-1} + l) + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot (l - y_i)\right)\right) \Leftrightarrow$$

$$\left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(\left(CPU_{p_{A(y_d)}} + CPU_{p_d}\right) \cdot (l - y_d - 1)\right)$$
$$> \left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(CPU_{p_{A(y_d)}} \cdot (l - y_d)\right)$$
$$+ CPU_{p_d}\left(CPU_v \cdot (\beta^{-1} + l) + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot (l - y_i)\right)\right) \Leftrightarrow$$

$$\left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(CPU_{p_{A(y_d)}} \cdot (l - y_d - 1)\right)$$
$$+\left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(CPU_{p_d} \cdot (l - y_d - 1)\right)$$
$$-\left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(CPU_{p_{A(y_d)}} \cdot (l - y_d)\right)$$
$$> CPU_{p_d}\left(CPU_v \cdot (\beta^{-1} + l) + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot (l - y_i)\right)\right) \Leftrightarrow$$

$$\left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(CPU_{p_{A(y_d)}} \cdot (l - y_d)\right) - \left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right) CPU_{p_{A(y_d)}}$$
$$+\left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(CPU_{p_d} \cdot (l - y_d - 1)\right)$$
$$-\left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(CPU_{p_{A(y_d)}} \cdot (l - y_d)\right)$$
$$> CPU_{p_d}\left(CPU_v \cdot (\beta^{-1} + l) + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot (l - y_i)\right)\right) \Leftrightarrow$$

$$\left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(CPU_{p_d} \cdot (l - y_d - 1)\right) - \left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right) CPU_{p_{A(y_d)}}$$
$$> CPU_{p_d}\left(CPU_v \cdot (\beta^{-1} + l) + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot (l - y_i)\right)\right) \Leftrightarrow$$

$$\left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left(CPU_{p_d} \cdot (l - y_d - 1) - CPU_{p_{A(y_d)}}\right)$$
$$> CPU_{p_d}\left(CPU_v \cdot (\beta^{-1} + l) + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot (l - y_i)\right)\right) \Leftrightarrow$$

$$\left(CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}}\right)\left((l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}\right)$$
$$> CPU_v \cdot (\beta^{-1} + l) + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot (l - y_i)\right) \Leftrightarrow$$

Note that:

- $l - y_d = l - (l - c_d - 1) = c_d + 1 \geq 2$ (as we're assuming that v clones itself at least 1 time)
- $\frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \leq 1$ (as $CPU_{p_{A(y_d)}} \leq CPU_{p_d}$)

Therefore $(l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} > 0$

We can divide both sides both sides by $(l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}$

$$CPU_v + CPU_k + \sum_{i=1}^{d} CPU_{p_{A(y_i)}} > \frac{CPU_v \cdot (\beta^{-1} + l) + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot (l - y_i)\right)}{(l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}} \Leftrightarrow$$

$$CPU_k > \frac{CPU_v \cdot (\beta^{-1} + l) + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot (l - y_i)\right)}{(l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}} - \sum_{i=1}^{d} CPU_{p_{A(y_i)}} - CPU_v \Leftrightarrow$$

$$CPU_k > \frac{CPU_v \cdot (\beta^{-1} + l) + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot (l - y_i)\right) - \sum_{i=1}^{d}\left((l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}\right) \cdot CPU_{p_{A(y_i)}} - \left((l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}\right) \cdot CPU_v}{(l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}}$$

$\Leftrightarrow$

$$CPU_k > \frac{CPU_v \cdot (\beta^{-1} + l) + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot \left(l - y_i - l + y_d + 1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}\right)\right) - \left((l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}\right) \cdot CPU_v}{(l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}}$$

$\Leftrightarrow$

$$CPU_k > \frac{CPU_v \cdot (\beta^{-1} + l) + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot \left(-y_i + y_d + 1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}\right)\right) - \left((l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}\right) \cdot CPU_v}{(l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}}$$

$\Leftrightarrow$

$$CPU_k \cdot \left((l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}\right) > CPU_v \cdot (\beta^{-1} + l) + \sum_{i=1}^{d}\left(CPU_{p_{A(y_i)}} \cdot \left(-y_i + y_d + 1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}\right)\right) - \left((l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}\right) \cdot CPU_v \Leftrightarrow$$

$$CPU_k \cdot \left((l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right) > CPU_v \cdot \left(\beta^{-1} + l - (l - y_d - 1) + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right) + \sum_{i=1}^{d} \left(CPU_{p_{A(y_i)}} \cdot \left(-y_i + y_d + 1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right) \right) \Leftrightarrow$$

$$CPU_k \cdot \left((l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right) > CPU_v \cdot \left(\beta^{-1} + l + y_d + 1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right) + \sum_{i=1}^{d} \left(CPU_{p_{A(y_i)}} \cdot \left(-y_i + y_d + 1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right) \right) \Leftrightarrow$$

Recall that $y_1 \geq y_2 \geq \cdots \geq y_d$

$$CPU_k \cdot \left((l - y_d - 1) - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right)$$

$$> CPU_v \cdot \left(\beta^{-1} + l + y_d + 1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right) + \sum_{i:y_i=y_d}^{d} \left(CPU_{p_{A(y_i)}} \cdot \left(1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right) \right)$$

$$+ \sum_{i:y_i=y_d+1}^{d} \left(CPU_{p_{A(y_i)}} \cdot \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right)$$

$$+ \sum_{i:y_i>y_d+1}^{d} \left(CPU_{p_{A(y_i)}} \cdot \left(-1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right) \right) \leq^{(*)} CPU_v$$

$$\cdot \left(\beta^{-1} + l + y_d + 1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right) + \sum_{i:y_i=y_d}^{d} \left(CPU_{p_{A(y_i)}} \cdot \left(1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right) \right)$$

$$+ \sum_{i:y_i=y_d+1}^{d} \left(CPU_{p_{A(y_i)}} \cdot \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right)$$

$$\leq CPU_v \cdot (\beta^{-1} + l + y_d + 2) + \sum_{i:y_i=y_d}^{d} \left(CPU_{p_{A(y_i)}} \cdot \left(1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right) \right)$$

$$+ \sum_{i:y_i=y_d+1}^{d} \left(CPU_{p_{A(y_i)}} \cdot \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right)$$

$$= CPU_v \cdot (\beta^{-1} + l + y_d + 2) + \sum_{i:y_i=y_d}^{d} \left(CPU_{p_{A(y_d)}} \cdot \left(1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right) \right)$$

$$+ \sum_{i:y_i=y_d+1}^{d} \left(CPU_{p_{A(y_i)}} \cdot \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right)$$

$$\leq CPU_v \cdot (\beta^{-1} + l + y_d + 2) + \sum_{i:y_i=y_d}^{d} \left(CPU_{p_{A(y_d)}} \cdot \left(1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right) \right)$$

$$+ \sum_{i:y_i=y_d+1}^{d} \left(CPU_{p_{A(y_i)}} \cdot \left(1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right) \right)$$

$$= CPU_v \cdot (\beta^{-1} + l + y_d + 2) + \sum_{\substack{i:y_i=y_d \\ \vee\ y_i=y_d+1}}^{d} \left(CPU_{p_{A(y_d)}} \cdot \left(1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \right) \right)$$

$$\leq CPU_v \cdot (\beta^{-1} + l + y_d + 2) + \sum_{\substack{i:y_i=y_d \\ \vee\ y_i=y_d+1}}^{d} \left(CPU_{p_{A(y_d)}} \cdot 2 \right)$$

$$\leq CPU_v \cdot (\beta^{-1} + l + y_d + 2) + CPU_{p_{A(y_d)}} \cdot 2 \cdot d$$

$(*)$ Last summation (when $y_i > y_d + 1$):

- $y_i > y_d + 1 \rightarrow 0 > -y_i + y_d + 1 \rightarrow -y_i + y_d + 1 \leq -1$

- $\frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \leq 1$
- Therefore $-y_i + y_d + 1 + \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} \leq 0$

We're looking for CPU_k such that:

$$CPU_k > \frac{CPU_v \cdot (\beta^{-1} + l + y_d + 2) + CPU_{p_{A(y_d)}} \cdot 2 \cdot d}{l - y_d - 1 - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}} = (*)$$

Case 1: $l \geq 2\mathrm{y_d} + 4$

In this case we use:

- $\frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}} < 1$
- $l \geq 2\mathrm{y_d} + 4 \leftrightarrow \frac{\mathrm{l}}{2} \geq \mathrm{y_d} + 2 \rightarrow l - y_d - 2 \geq \frac{l}{2}$
- $l \geq 2\mathrm{y_d} + 4 \geq 4$

$$\begin{aligned}(*) &\leq \frac{CPU_v \cdot (\beta^{-1} + l + y_d + 2) + CPU_{p_{A(y_d)}} \cdot 2 \cdot d}{l - y_d - 2} \\ &\leq \frac{CPU_v \cdot (\beta^{-1} + l + y_d + 2) + CPU_{p_{A(y_d)}} \cdot 2 \cdot d}{\frac{l}{2}} \\ &\leq CPU_v \cdot \left(1 + \frac{2\beta^{-1} + 2y_d + 4}{l}\right) + \frac{CPU_{p_{A(y_d)}} \cdot 4 \cdot d}{l} \\ &\leq CPU_v \cdot \left(2 + \frac{\beta^{-1}}{2}\right) + CPU_{p_{A(y_d)}} \cdot d \leq CPU_v \cdot \left(2 + \frac{\beta^{-1}}{2}\right) + d^{H+1}\end{aligned}$$

Case 2: $l \leq 2\mathrm{y_d} + 3$

In this case we use:

- By definition $l \geq y_d + 2 \rightarrow l - y_d \geq 2$
- $A(y_d) \geq y_d \rightarrow\ l \leq 2A(y_d) + 3$
- $d \geq 3$

$$
\begin{aligned}
(*) &\leq \frac{CPU_v \cdot (\beta^{-1} + 2y_d + 3 + y_d + 2) + CPU_{p_{A(y_d)}} \cdot 2 \cdot d}{2 - 1 - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}} \\
&\leq \frac{CPU_v \cdot (\beta^{-1} + 3y_d + 5) + CPU_{p_{A(y_d)}} \cdot 2 \cdot d}{1 - \frac{CPU_{p_{A(y_d)}}}{CPU_{p_d}}} \\
&\leq \frac{CPU_v \cdot (\beta^{-1} + 3y_d + 5) + CPU_{p_{A(y_d)}} \cdot 2 \cdot d}{1 - \frac{A(y_d)}{d^{y_d}} CPU_v} \\
&= \frac{CPU_v \cdot (\beta^{-1} + 3y_d + 5) + CPU_{p_{A(y_d)}} \cdot 2 \cdot d}{1 - \frac{A(y_d)}{d^{y_d}}} \\
&\leq \frac{CPU_v \cdot (\beta^{-1} + 3y_d + 5) + CPU_{p_{A(y_d)}} \cdot 2 \cdot d}{1 - \frac{1}{d^{y_d}} \frac{d^{y_d} - 1}{d - 1}} \\
&\leq \frac{CPU_v \cdot (\beta^{-1} + 3y_d + 5) + CPU_{p_{A(y_d)}} \cdot 2 \cdot d}{1 - \frac{1}{d^{y_d}} \frac{d^{y_d}}{d - 1}} \\
&\leq \frac{d - 1}{d - 2} \left(CPU_v \cdot (\beta^{-1} + 3y_d + 5) + CPU_{p_{A(y_d)}} \cdot 2 \cdot d \right) \\
&\leq \frac{3 - 1}{3 - 2} \left(CPU_v \cdot (\beta^{-1} + 3y_d + 5) + CPU_{p_{A(y_d)}} \cdot 2 \cdot d \right) \\
&= 2 \left(CPU_v \cdot (\beta^{-1} + 3y_d + 5) + CPU_{p_{A(y_d)}} \cdot 2 \cdot d \right) \\
&\leq 2 \cdot CPU_v \cdot (\beta^{-1} + 3y_d + 5) + CPU_{p_{A(y_d)}} \cdot 4 \cdot d \\
&\leq 2 \cdot CPU_v \cdot (\beta^{-1} + 3A(y_d) + 5) + CPU_{p_{A(y_d)}} \cdot 4 \cdot d \\
&\leq 2 \cdot CPU_v \cdot (\beta^{-1} + 3 \cdot CPU_v + 5) + CPU_{p_{A(y_d)}} \cdot 4 \cdot d \\
&\leq CPU_v \cdot (2\beta^{-1} + 6 \cdot CPU_v + 10) + CPU_{p_{A(y_d)}} \cdot 4 \cdot d \\
&\leq CPU_v^2 \cdot (2\beta^{-1} + 16) + CPU_{p_{A(y_d)}} \cdot 4 \cdot d \\
&= CPU_v^2 \cdot (2\beta^{-1} + 16) + A(y_d) \cdot CPU_v \cdot 4 \cdot d \\
&= CPU_v^2 \cdot (2\beta^{-1} + 16) + CPU_v \cdot (A(y_d) \cdot 4 \cdot d) \\
&\leq CPU_v^2 \cdot (2\beta^{-1} + 16) + CPU_v \cdot (d^H \cdot 4 \cdot d) \\
&= CPU_v^2 \cdot (2\beta^{-1} + 16) + CPU_v \cdot (d^{H+1} \cdot 4)
\end{aligned}
$$

Therefore we have that if $CPU_k > CPU_v^2 \cdot (2\beta^{-1} + 16) + CPU_v \cdot (d^{H+1} \cdot 4)$ for any node v the strategy of choosing $c_i = l(v) - \varphi - 1$ for some child i is dominated by the strategy $(c_1, \dots, c_{i-1}, c_i - 1, c_{i+1}, \dots, c_d)$.

■

4. Actual Proof of the Theorem

Suppose that for every node v in the network it holds that $CPU_{-v} > CPU_v^2 \cdot (2\beta^{-1} + 16) + CPU_v \cdot (d^{H+1} \cdot 4)$.

We'll use the above lemma to prove the theorem. Like we noted before, the 0-game is simply the (β, H)-game.

Let's consider the set of all possible strategies for the (β, H)-game. We can apply the above lemma to eliminate all strategies $\left(c_1^{l(v),v}, \dots, c_d^{l(v),v}\right)$ such that some $c_i^{l(v),v} = l(v) - 1$. That means that the maximum number of times any node v can clone itself is $l(v) - 1 - 1$ times. Thus, we got a 1-game. We can apply the lemma again and get a 2-game. We repeatly apply the lemma until we get a $(H - 1)$-game. Under this contrainst any node v can clone itself at most $\max\{l(v) - H - 1,0\} = \max\{H - i + 1 - H - 1,0\} = \max\{-i, 0\} = 0$ times. In other words, the only surviving strategy of each node is $(0, \dots ,0)$ for every $l(v)$: that is, each node propagates information and does not duplicate. This shows that there exists an elimination order in which every node propagates information and does not duplicate.

■

Conclusion

The authors of 'On Bitcoins and Red Balloons' tackle a concrete problem on a concrete protocol (Bitcoin) used by hundreds of thousands of people every day. The solution that the authors present is not only elegant but also has important properties like Sybil-proofness and low overhead.

The authors assume a very constrained model for this paper, which means that their results can be extended to more general models. With this in mind, in this project I relaxed one of their model's constraints and proved a similar result for a more general case, one in which not all the nodes in the network have the same processing power. In preparation, I had to completely understand every proof for every theorem and lemma presented in the paper in order to be able to use and modify (when necessary) them so that they would work under the new more general model.