

# P2P bitcoins trading system

Gzmask

November 16, 2011

## **Abstract**

This is a paper of a P2P system that let users trade digital works online, kind of like freelancing. There are some basic peer-review processes that guarantee the works are paid fairly. However, current design is prone to fraud client attacks.

## **1 Introduction**

Online trading using credit cards or paypal-like system is popular. A famous example is Ebay.com. All these systems are centralized, thus the trades are slow and controlled by some private parties in some degree. As bitcoins emerged, pure P2P system is getting recognized. Here I proposed a very basic and simple design for a P2P trading system.

## **2 Important constants(Skip this if you are reading first time)**

N: Total user amount (updates weekly)

K: Total voter amount (updates three days)

O: offer of the posted Need

P: process fee of the dispute.

$$P = \text{smaller}(\frac{O}{10}, \frac{O}{K})$$

R: reward for each voter.

$$R = 2 \times \frac{P}{(K \times \frac{\text{correct\_votes}}{\text{total\_votes}})}$$

## 3 Models

### 3.1 Node

This model stores in all (N) nodes

Properties

- IP\_address
- Email\_address
- Email\_password
- reputation
- is\_voter

reputation: if ratings less than 10, shows actually numbers of ratings. i.e. 5:1 where 5 rates good and 1 rates bad; if ratings more than 10, only shows percentages. This prevents profile bias towards worker nodes.

### 3.2 Need

This model stores in Log(N) nodes

Properties

- time\_to\_live
- offer
- src\_node\_address

### 3.3 Work

This model stores in posted node  
Properties

- time\_to\_live
- proposal
- accepted
- price
- src\_node\_address
- des\_node\_address
- solution

### 3.4 Dispute

This model stores in posted node  
Properties

- needer\_ip\_address
- worker\_ip\_address
- vote
- needer\_process\_fee
- worker\_process\_fee
- next\_voter\_address

## 4 Actions

### 4.1 Post a Need

1. a node  $U_1$  broadcasts, to  $\text{Log}(N)$  nodes, a need  $N_1$  in the network while deposits  $O$  bitcoins into  $N_1$  as an offer.
2. all other nodes  $U_x$  receive the broadcast of  $N_1$  and store it locally
3. after the TTL expired, each node removes  $N_1$  permanently, and deposit  $O$  returns to  $U_1$
4. when  $U_1$  logs off, it will not longer be able to accept proposed works.  $U_1$  is supposed to be online until a solution is accepted.

### 4.2 Propose a work

1. any node can propose a work for a need.
2. let  $U_2$  be a node proposing a work  $W_1$ .  $U_2$  send a Email to  $U_1$ , telling  $U_1$  that  $W_1$  is proposed at  $U_2$ .
3.  $U_1$  gets notified that  $W_1$  is proposed. There can be multiple works that are proposed by other nodes.
4.  $U_1$  can accept one of the proposed work, or wait. If  $N_1$  expires, all proposed works expire at the same time and the case is over.

### 4.3 Accept a work solution

1. let  $U_1$  accepts  $W_1$  from  $U_2$ . Then  $U_1$  sends a message to  $U_2$ ,  $W_1$  is accepted and  $U_2$  can give a solution to  $W_1$ .
2. After  $U_2$  submits a solution for  $W_1$ ,  $U_1$  receives a message notification that he can review the solution from  $U_2$ . Now  $U_1$  can either chose to accept or reject the solution.
3. If the solution is accepted, the deposited bitcoins in  $N_1$  will be transferred to  $U_2$ .

4. If the solution is rejected, then  $U_1$  submits a dispute  $D_1$ . When  $D_1$  is created,  $U_1$  needs to deposit  $P$  bitcoins into the  $D_1$  and the offer  $O$  in  $W_1$  will be deducted by  $P$ . Then it starts the peer-review voting process.

#### 4.4 Resolve a dispute

1. nodes can choose to peer-review dispute cases to earn bitcoins. The reward for an effective vote is  $R$ , that is, when the vote agrees with the end result. A dispute is resolved only after reviewed by  $\log(K)$  randomly chosen nodes.
2. when  $U_1$  files  $D_1$ , it searches for other voter Nodes and pick one randomly, say  $U_3$ .
3.  $U_1$  sets  $D_1$ 's next\_voter\_address to  $U_3$ , and  $U_3$  dupes  $D_1$  locally.
4.  $U_3$  reviews the case, vote, and picks another voter node randomly, until reaches  $U_{\log(K)}$ .
5.  $U_{\log(K)}$  notifies needer, worker and other voters nodes the result.
6.  $U_1$  either keeps the deposit  $O - P$  or gives  $O - P$  to  $U_2$  depends on the result. And the reward  $R$  is given to each voter node.

### 5 Super nodes: needer nodes or voter nodes

Since needers have to be staying online to wait for solutions, thus they are most likely potential voter nodes. These nodes can be seen as super nodes that can be used to handle extra services.

### 6 Security

It's easy to see that the system is prone to fraud client attacks. A fraud node can post needs without a deposit and the worker node giving solution to that need therefore is cheated. I am still figuring out a decentralize mechanism

to solve this problem. For now, to solve this we need a centralize Need, Dispute, Deposit and Processing Fee hosting service. Or, simply give the Needer nodes trust while the on going work is not too costly(say, answering questions).

## **6.1 Account reputation directory**

A potential solution to the fraud client problem is to add an P2P account reputation directory, so that Need posting nodes will have a reputation reference similar to ebay's seller ratings/feedback system.

## **References**