Deanonymizing Bitcoin Blockchain Transaction Networks

515030910365-王裕杰

1. Introduction

Bitcoin is a cryptocurrency and worldwide payment system. It is the first decentralized digital currency, as the system works without a central bank or single administrator. The system was designed to work as a peer-to-peer network, a network in which transactions take place between users directly, without an intermediary. These transactions are verified by network nodes through the use of cryptography and recorded in a public distributed ledger called a blockchain. Because of its perceived anonymity properties, Bitcoin is widely used and becomes a popular alternative to flat money. However, recent attacks on Bitcoin's peer-to-peer (P2P) network demonstrated that its gossip-based flooding protocols, which are used to ensure global network consistency, may enable user deanonymize the linkage of a user's IP address with her pseudonym in the Bitcoin network.

In order to analyze the anonymity properties of the Bitcoin P2P network, I will introduce a model to demonstrate the spreading protocols of Bitcoin P2P network and do some simulation to prove the probability of deanonymizing Bitcoin blockchain transaction networks.

Report structure is as follows. In section 2, I will introduce the model in detail. In section3, I will do some simulation and analyze the result of simulation. In section 4, I will conclude the report. Finally, I will add the code for simulation in the attached page for testing.

2. Model

In this section, I will introduce the model to explain how the bitcoin networks spread messages.

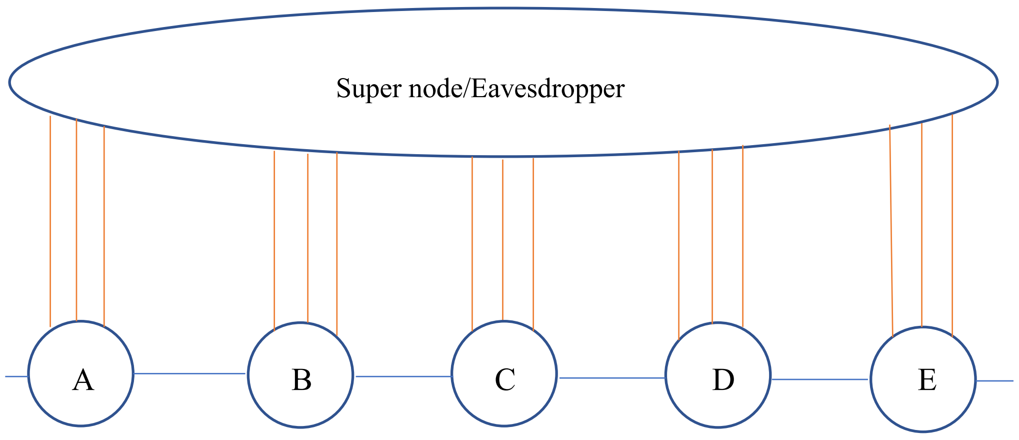

Figure 2-1 Model of bitcoin spreading protocol with eavesdropper adversary node

Figure 2-1 shows the model of bitcoin spreading protocol with eavesdropper adversary node. As we know, every time a transaction (or a block) is completed, it is broadcast over the network. Each node will randomly orders its neighbors wo have not seen the message. Then it transmits the message to its neighbors according to the ordering with a constant delay of 200ms between transmissions. In this model, each node randomly orders its uninfected neighbors and transmits the message to one neighbor per subsequent time step.

All of above are related to common nodes while there are also an eavesdropper adversary in the model. The adversary can make deanonymization attacks which are cheap, scalable and simple and represent realistic threats to the network. These attacks use a supernode that connects to most of the nodes in the Bitcoin network. The supernode can make multiple connections to each node without being noticed. Therefore, the supernode can receive the message from common nodes and record corresponding timestamp but don't transmit any messages.

Now let's return to Figure 2-1, we can see five common nodes and a supernode in this network. Each common node connects to **d** (In this figure,d=2) common nodes and the supernode connects to each common node **θ** (In this figure, θ=3) times. Assuming that a message is originated from node C at the first timestamp, C finds that it connects to 5(θ+d) nodes so it randomly chooses a node to transmit the message. If C chooses B, B will receive the message and at next timestamp B and C will check their remaining nodes and keep transmitting until there are no remaining nodes. If C chooses the supernode(C doesn't know it isn't a common node), the supernode will receive the message and record that C sends the message at the first timestamp. Anytime the supernode receive the message, it will record the node and the corresponding timestamp. After collecting the message from all the node, the supernode will be able to find which node originated this message. According to different strategy, the supernode may get different result. In our model, we take the First-Timestamp Estimator which means the supernode will regard the first node to send the message to it as the origin.

Now the introduction of the model has been finished, let's turn to section 3 to see the simulation result.

3. Simulation

In this section, I will show the simulation result of detection probability(the probability of finding the real node that originated the message).

Before introducing my simulation result, I'd like to introduce some related work. In a paper named " Anonymity Properties of the Bitcoin P2P Network", it has explained some of the results. In this paper, it gives a formula for calculating the probability under the model:

$$P = \frac{\theta}{d \log 2}[Ei(2^d \log p) - Ei(\log p)] \text{, where } p = \frac{d-1}{d-1+\theta} \text{ and } Ei(x) = -\int_{-x}^{\infty} \frac{e^{-t} dt}{t}$$

θ and d have been introduced in section 2.

This paper has also given the simulation graph to describe the relationship between detection probability and parameter d which has been shown in Figure 3-1.

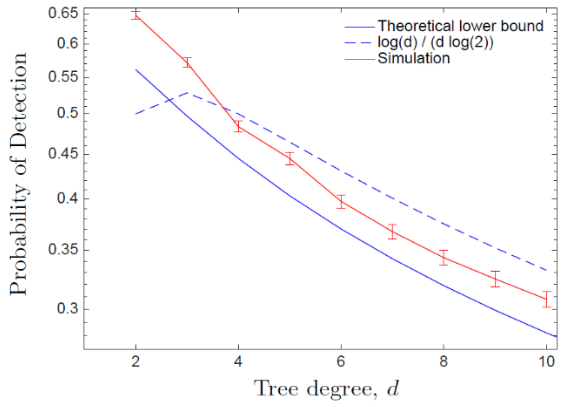

Figure 3-1 relationship between detection probability and parameter d (θ=1)

All of above are the work done by this paper. It gives a formula to describe the relationship between the detection probability and parameter θ and d. It also gives its simulation result about parameter d.

However, this paper didn't concern about the simulation result about parameter θ. So in order to verify the result and to find the relationship between probability and parameter θ, I writer a code with Python and get my simulation result.(See Figure 3-2)

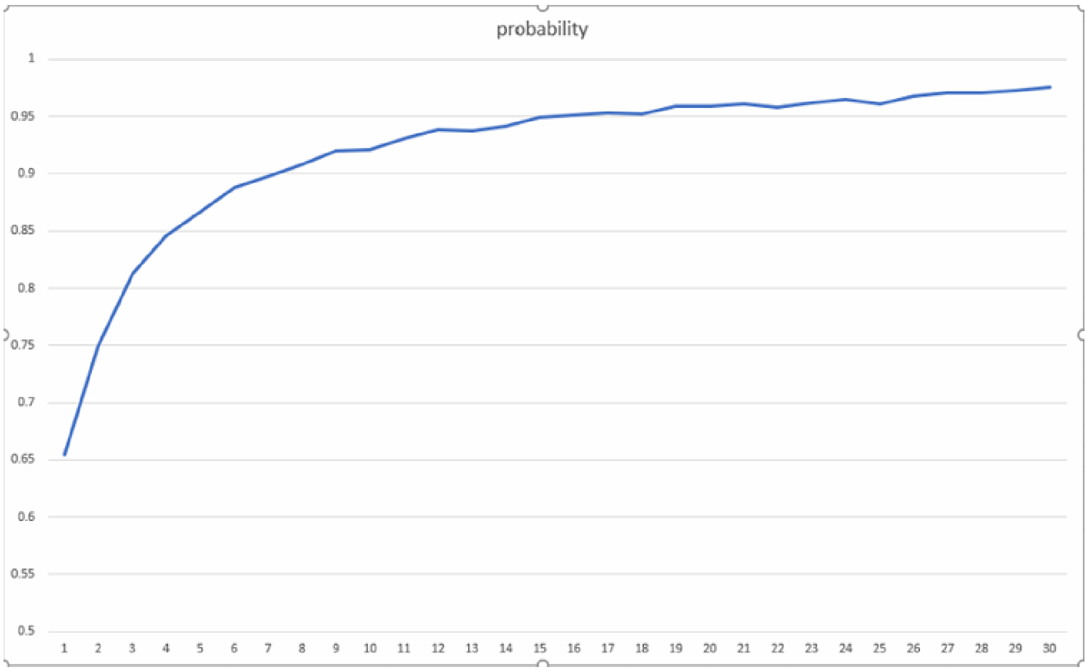

Figure 3-2 Relationship between detection probability and parameter θ (d=2)

From Figure3-1, we can easily find that with the increase of d, the detection probability declines. But from Figure3-2, the detection probability rises with the rise of θ. Both of them are obvious and reasonable. When increasing d, it means a common node is more likely to transmit a message to another common node instead of the supernode. In contrast, when the parameter θ rises, the supernode becomes more likely to receive the message at the first timestamp. From Figure3-2, when θ comes to 30, a supernode has set up 30 connections to each node. It means the message origin node C will detect 32 neighbors and 30 of them are from the supernode. So there is really high probability to send the message to the supernode at the first timestamp. In addition, to ensure that the model created by my code is indeed the model in Chapter 2, I choose some common data points with the paper. We can clearly see that the first point of both Figure 3-1 and Figure 3-2 actually share the same parameter(θ=1,d=2). Their corresponding probability are also the same: about 0.65%. So this ensures that the model I create in my code is actually the model in Section 2.

4. Conclusion

In conclusion, although Bitcoin uses the blockchain to ensure its safety, it still has poor anonymity properties on networks and the potential to suffer from deanonymization attacks which may lead to leak of private information. As shown in the two figures in section 3, if an adversary really established so many connections, our private information in Bitcoin network will be easily exposed to him. That will be too terrible.

5. Code

This section contains my simulation code (Python 3.5.1).

```
import time
import random
import csv

class Block:
    def __init__(self,index=0,data=0,link=[],special=[]):
        self.index=index
        self.data=data
        self.link=link
        self.special=special
#establish a Blockchain

def create_genesis_block(Theta):
    return Block(1,0,[2]+[0]*Theta,[])

def next_blockchain(last_block,Theta):
    this_index=last_block.index+1
    this_data=0
    this_link=[this_index+1,this_index-1]+[0]*Theta
    this_special=[]
    return Block(this_index,this_data,this_link,this_special)

def create_supercode(l=1):
    this_index=0
    this_data=0
    this_link=[]
    for i in range(l):
        this_link.append(i+1)
    this_special=[]
    return Block(this_index,this_data,this_link,this_special)

def create_signal(blockchain,num):
    start=random.randint(1,num)
    blockchain[start].data=1
    return blockchain,start
#originate the message from some node

def check_signal(blockchain,num):
    idn=[]
    for i in range(num):
            if (blockchain[i+1].data==1):
                idn.append(i+1)
        return idn
#find all codes that have received the message

def transmit_signal(blockchain,num,l):
        if (len(blockchain[l].link)==0):
            return blockchain
        else:
            number=random.randint(0,len(blockchain[l].link)-1)
            link_temp=int(blockchain[l].link[number])
            blockchain[l].link.remove(link_temp)
            if (link_temp==0):
                blockchain[0].special.append(l)
                return blockchain
            elif(link_temp>num):
                return blockchain
            else:
                blockchain[link_temp].data=1
                blockchain[link_temp].link.remove(l)
                return blockchain
#simulate the process of transmitting message

def check_transmit_finish(blockchain,num):
        flag=1
        for i in range(num):
            flag=flag*blockchain[i+1].data
        return flag
#check whether all nodes have received the message

def clear(blockchain,num):
        for i in range(num+1):
            blockchain[i].index=0
            blockchain[i].data=0
            blockchain[i].link=[]
            blockchain[i].special=[]
        return blockchain
#clear all the data

def main(num=5,Theta=2):#num is the number of the common node
        blockchain=[]
        blockchain=[create_supercode(num)]
        blockchain.append(create_genesis_block(Theta))
        previous_block=blockchain[1]
```

```
    for i in range(num-1):
        block_to_add=next_blockchain(previous_block,Theta)
        blockchain.append(block_to_add)
        previous_block=block_to_add
    #create a blockchain
    blockchain,start=create_signal(blockchain,num)
    flag=0
    while(flag==0):
        idn=check_signal(blockchain,num)
        for i in idn:
            blockchain=transmit_signal(blockchain,num,i)
        flag=check_transmit_finish(blockchain,num)
    end=blockchain[0].special[0]
    return start,end

start=time.time()
f=open("test.csv",'w',newline='') #output the probability to "test.csv"
writer=csv.writer(f)
writer.writerow(['θ',"probability"])
for i in range(30):#i is the range of the parameter θ
    count=0
    for j in range(10000):
        a,b=main(10,i+1)
        if (a==b):
            count=count+1
    p=count/10000
    writer.writerow([i+1,p])
f.close()
end=time.time()
print("Total time used: ",int(end-start)/60,"min")
```