Peer Discovery With Transitive Trust in Distributed System

ChangLiang Luo

Delft University of Technology

Peer Discovery With Transitive Trust in Distributed System

Master's Thesis in Computer Science

Parallel and Distributed Systems group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Changliang Luo

11th October 2017

Author
Changliang Luo

Title
Peer Discovery With Transitive Trust in Distributed System

MSc presentation
October 20, 2017

Graduation Committee

Dr.ir.J.Pouwelse	Delft University of Technology
Dr.N.Yorke-Smith	Delft University of Technology
Dr.sc.ETH.J.S.Rellermeyer	Delft University of Technology

Abstract

It is a common case that resources belong to different people who are distributed across the world. So, for making good use of resources, people have to cooperate with each other. Cooperation is important not only in the physical world but also in cyberspace. Computers holding different resources also need to cooperate with others. The problems that people cooperate with people and computers cooperate with computers can be abstracted into a high level one – peers cooperate with other peers in a network. In such problems, the first step to initiate a cooperation is locating other peers – you can not cooperate with a peer when you are not even aware of its existence. The task to locate other peers is called "peer discovery", it is not an easy task, especially in distributed fashion. Peers need to acquire information of other peers from somewhere, if there is not a central party, the only place to acquire such information is other peers. Malicious peers may provide toxic information to other peers. Therefore, unconditionally trust other peers is very dangerous. For security concern, peers need to find a way to decide who is trustworthy and who is not. This thesis aims to establish trust among peers basing on the historical behaviors of other peers. We believe that by using the established trust, a Peer-to-Peer system will be more resilient to Sybil Attack where attackers create a lot of fake identities to deceive honest peers. Our trust system will be implemented and tested in the peer discovery system of Tribler, which is a distributed system helping people share files.

Preface

A few years ago, when I put my eyes on the civilization of human beings, it looks like a growing onion. Something lies in the core of the onion, wrapped by technology, culture, art. Now, I know what is the core – cooperation. In stone age, hunters cooperate to hunt mammoth; in a feudal village, people cooperate to farm and defend; in a industrialized state, people cooperate for producing; in the global community, people cooperate for business. We can see a clear trend that as our civilization develops, the scale of cooperate becomes larger and larger. In this sense, we can also say the essence of the civilization is cooperation. The foundation of cooperation is trust – a belief that cooperating with specific people will make the outcome better. However, trust can also be dangerous, trust people you should not trust can be a tragedy. As the scale of cooperation grows larger, the cost of trust wrong people grows heavier. In the hunter team, trust a selfish teammate will cost tens of lives; in the feudal village, the trust for an unqualified alcade may ruin the whole village; in the industrialized state, the trust for an treacherous general may put the whole state in fire; in the global community, trust for an dishonest banker may cause a soaring financial crisis, making billions of people at stake. In the arsenal of computer science, we can also see the cost of trusting wrong people, for example, Sybil Attack in Main Line Bittorent. The potential damage of mistrust gives me motivation to improve the peer discovery module of Tribler, adding a trust mechanism without a centralized server. I believe by using such mechanism, we can mitigate the damage of Sybil Attack.

I cannot complete my thesis without the help of many people. First, I need to thank Johan Pouwelse for daily supervision; he provides me with inspiration and valuable advices, sheds light on the path towards to research goals. I also appreciate the help from Martijn de Vos and Quinten Stokkink, they kindly help me warm up with Tribler project. I am also grateful to Kelong Cong, he offers invaluable experience and advice on my thesis.

Changliang Luo

Delft, The Netherlands
11th October 2017

Contents

Preface **v**

1 Introduction **1**

2 Problem Description **9**
2.1 Peer Discovery 9
2.2 Potential Attacks 10
2.3 Defense mechanism 12
2.4 Research goal 14

3 Trust Based Peer Discovery **15**
3.1 Peer Discovery in Dispersy 15
3.1.1 Message 16
3.1.2 Community 16
3.1.3 Walker and Peer Discovery 17
3.1.4 Misleading Attack in Dispersy Walker 20
3.2 Storing Historical Behaviors 20
3.2.1 Blockchain 21
3.2.2 Trustchain 22
3.3 Scoring System and Walking strategies 24
3.3.1 Scoring System 24
3.3.2 Walking Strategies 25
3.4 Sybil Attack Defense 27

4 Design of Transitive-Trust Walker **33**
4.1 Design Requirement 33
4.2 Walker Architecture 34
4.3 Block Collecting 35
4.4 Reputation System 37
4.5 Walking Strategies 38
4.5.1 Bias Random Walking 38
4.5.2 Teleport Walking 39

5 Validation **43**
5.1 Validation List 43
5.2 Simulated Network 44
5.3 Coverage Validation 45
5.4 Load Balance Validation 48
5.5 Sybil Prevention Validation 50
5.6 Discussion 52
5.7 Unsolved Problem 53

6 Conclusions and Future Work **57**
6.1 Conclusions 57
6.2 Limitation 58
6.3 Future Work 59

Chapter 1

Introduction

The topic of this thesis might sound technical and obscure, but we will try to explain it in a friendly way. We try to use the following story to shed some lights on the concept of "peer discovery"

Imagine that you live in the 1970s and you are seeking for an old friend that you have not seen for 10 years. You have no idea what job he is doing, which country he stays, what city he lives in, what telephone number he is using, and there is no Facebook or Twitter at the 1970s. The only thing you know is he was your classmate in B University 10 years ago. How can you find him? The most ideal case is that there is an address book containing the living address and contact information for all people in this world. If there is such a book somewhere, the address book should be around 35 kilometers thick (1 page for 20 people, then 1000 pages for 20 thousand people with 10 centimeters thick), it is better to call it "address tower". We all know it is unrealistic.

The more realistic way is that you visit the B University, then visit people and request the contact information of your old friend. It is likely they do not have such contact information, but they have a small probability to know other people who may know the contact information of your old friend, and they introduce such people to you. Then you visit the introduced people, request information from them and move on. By doing so, you get closer and closer to your old friend, and finally "discover" him in a corner of this vast world. But there is another story, you accidentally visit some liars, they lie to you about the contact information of your friends, mislead you to the other side of the Earth. And, unsurprisingly, you end up with finding nothing there.

The way you find your friend is actually happening in many Peer-to-Peer systems; the task to find another peer is called "peer discovery". Just like the two stories above, in peer discovery, you need to make decisions on two aspects: first, who to visit; second, who to trust.

The story above might give you the first impression of "peer discovery", with this impression, we can now introduce the "peer discovery" in a formal way. But before discussing peer discovery, we need to discuss the concept of Distributed

System and Peer-to-Peer System.

A distributed system can be seen as a set of peers cooperating with each other, in the story we above, the people who participant in the introduction cooperation can be seen as a distributed system. However, when we mention "distributed systems" today, we usually refer to a set of computers. A formal definition of distributed system is provided by Tanenbaum and Van Steen [1]:

> *A distributed system is a collection of independent computers that appear to its users as a single coherent system.*

In this sense, the Internet is a typical distributed system: computers own by individuals or organizations distributed across the world, those computers are "independent computers" and for a typical user, the Internet appears to him/her as a "single coherent system".

A distributed system can be further categorized as different subclass according to their architecture, a "traditional" one is Client/Server Architecture, a network consisting of multiple clients and a web server is a typical type of Client/Server architecture. Clients and web servers cooperate with each other but they play different roles in such system – clients request web pages and web server rely with web pages. To be more accurate, Client/Server architecture can be defined as followings [2]:

> *A Client/Server network is a distributed network which consists of one higher performance system, the Server, and several mostly lower performance systems, the Clients. The Server is the central registering unit as well as the only provider of content and service. A Client only requests content or the execution of services, without sharing any of its own resources.*

This definition emphasizes the difference between Server and Client on the aspects of resources and roles. The Server has more resources than Clients, and the roles of Server and Clients are quite different. In addition, the definition assume servers to be more powerful than clients

Besides the Client/Server architecture, there is a more "modern" type of distributed system: the Peer-to-Peer architecture. In the Client/Server architecture, we enforce different roles on participants – Client and Server. Participants with different roles have different functionalities and responsibilities, they are born to be unequal. An alternative of this design is that we do not enforce roles on participants in the system. A architecture with such design is called Peer-to-Peer architecture.There are multiple definitions for Peer-to-Peer system, but here I use the one provided by [2]:

> *A distributed network architecture may be called a Peer-to-Peer (P-to-P, P2P,...) network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link*

> *capacity, printers,...). These shared resources are necessary to provide the Service and content offered by the network (e.g. file sharing or shared workspaces for collaboration). They are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource (Service and content) providers as well as resource (Service and content) requestors (Servent-concept)*

This definition implies the difference between Client/Server architecture and Peer-to-Peer architecture: in Peer-to-Peer architecture every peer can share part of their resources, so there is not necessarily a higher performance peer; all peers play the same role in Peer-to-Peer architecture, unlike Client/Server architecture where servers and clients have distinct roles.

Peer-to-Peer System is not a new concept, but it does not raise public attention until the success of some famous applications, like BitTorrent, a popular file sharing application released in 2001. By the time of 2015, it still dominated the upstream traffic of Internet – having an upstream share of 28.56% [3]

BitTorrent is not a specific software, it is a protocol described in BitTorrent Protocol Specification [4]. Softwares that support the BitTorrent protocol can join the network and upload or download files to/from other users. The downloading and uploading happen between clients without passing through intermediary entity. That raises a problem – how to locate the clients you want to contact. There are billions of machines running on the Internet. According to the white paper of Cisco in 2011 [5], there are 12.5 billion devices running on the Internet by the time of 2010, the number of devices connected to the Internet should be greater in 2017. Among all these devices,only part of them are running BitTorrent clients, and only part of the BitTorrent clients have the files you want to download, hence the first step to initiate a downloading or uploading operation is figuring out the address of the machines which run the clients that contain the files you want.

BitTorrent is not the only one who faces this problem. In fact, for all Peer-to-Peer systems, the foundation of cooperation among peers are peer discovery. For all those systems, peer discovery problem can be described as "given some clues, find out the address of other peers", but since different Peer-to-Peer systems varies on many aspects, they may choose different peer discovery approaches. To better explain the idea of different peer discovery approaches, we will describe some typical applications which use different peer discovery approaches as examples. The examples are: ARP request in Ethernet, old fashion BitTorrent, Mainline Distributed Hash Table (MLDHT) based BitTorrent, Device to Device (D2D) communication.

The first example for peer discovery is APR request in Ethernet. As we know, the Internet is consist of multiple layers of protocols, according to OSI model [6], there are seven layers. From low level to high level, the seven layers are: Physical Layer, Data Link Layer, Network Layer, Transport Layer, Session Layer, Presentation Layer, Application Layer. The famous IP address is the address used in Network Layer. However, for lower layers like Physical Layer and Data Link Layer, applications running in such layers cannot "understand" IP address. For example,

in Data Link Layer, applications have no idea what is IP address; instead of using IP address, the applications use another address to uniquely locate machines, that address is called MAC address. In most case, there is a mapping from IP address to MAC address, for example, Machine B has an IP address 35.150.1.2 and a MAC address 00:00:00:00:00:02; so both 35.150.1.2 and 00:00:00:00:00:02 refer to Machine B, we say the 35.150.1.2 is mapped to 00:00:00:00:00:02. Unfortunately, this mapping is not known by all machines in this world. In some cases, a machine (say Machine A) has some packet to send to IP address 35.150.1.2, but the packet needs to be sent through Ethernet where machines use MAC address rather than IP address (we skip some technique details here, they are irrelevant to this thesis); unluckily, Machine A does not know the mapping between 35.150.1.2 and 00:00:00:00:00:02, in other words, Machine A does not know the MAC address of Machine B is 00:00:00:00:00:02. For Machine A, the problem is now a peer discovery problem: the clues for Machine A is "the target machine has IP address 35.150.1.2", and the goal is "find out the address (MAC address) of target machine". To solve the peer discovery problem, Machine A will use Address Resolution Protocol (ARP), the procedures are: Machine A sends an ARP request to all machines in this Ethernet asking "who has IP address 35.150.1.2, tell me your MAC address". All machines in this Ethernet will receive this ARP request, they will check whether they have IP address 35.150.1.2, if not, they will ignore it. So, everyone in the network will ignore this ARP request, except Machine B. Once receive this APR request, B will reply an ARP response to Machine A, saying "I have the IP address 35.150.1.2, my MAC address is 00:00:00:00:00:02". When A receiving the ARP response, it knows the MAC address of Machine B, mission accomplished.

The second example is old version BitTorrent clients. Though BitTorrent is a typical Peer-to-Peer system, in its early years, the peer discovery is conducted in a centralized way. In BitTorrent, people download or upload different files, so BitTorrent protocol group peers according to files, all peers downloading or uploading the same file are grouped together, and such group is called "swarm". For each swarm, there is a file containing meta information associated with it, the file is called "torrent". A torrent file contains a URL for a centralized server, called "tracker". A tracker is responsible for one or more swarms. To start downloading a file, one needs to join the swarm; to join the swarm, one needs a torrent file. For a newly joined peer, it has no knowledge about the address of other peers in this swarm, hence the peer faces a peer discovery problem, the peer solve this problem by asking the tracker in this swarm for information about other peers. After receiving the response from the tracker, the peer will obtain the address of other peers in this swarm hence can start download or upload with those peers.

The third example is Main Line DHT based BitTorrent clients. For some considerations, BitTorrent starts to use a distributed hash table to replace the central tracker. The most popular implementation of this is MLDHT based BitTorrent. MLDHT implements the functionality of Kademlia Distributed Hash Table [7]. In Kademlia Distributed Hash Table, the peers and contents are stored in a distributed

way, but since we are talking about peer discovery, we are only interested in the way to store peers. For every peer, Kademlia Protocol assigns an ID to it where the ID is a 160 bits hash; this hash uniquely identifies the peer. Kademlia Protocol introduces a concept of "distance" between two ID, hence we can calculate the distance between every two peers. For a peer, it is only responsible to store an (IP address, UDP port, Peer ID) pair for peers close to it. The term "close to it" means the distance is less than a certain value d, d can be specified in the configuration. For a peer A, if it wants to discover the peer with certain Peer ID, it can do it in following way:

- **Step 1**: Pick out k-closest peers from A's peer list, send a FIND NODE message which contains the target Peer ID to them.
- **Step 2**: For every peer received FIND NODE message from peer A, it will check its peer list, if it knows the address associated with this peer ID, it will return its address to A, otherwise it will introduce its k-closest peers to A
- **Step 3**: If A receives the address of the target peer from any peer, mission accomplished; otherwise it will store the address of the introduced peers, and go back to step 1.

The fourth example is network-assisted peer discovery approaches of D2D network [8]. To support communication among mobile devices like our cell phones, network service providers place base stations all over the world. The base stations divide the world into many sections called "cell". For two people (Alice and Bob) stay in two sections which are faraway with each other, when they communicate with each other (e.g. take a phone call), they are actually doing the followings: Alice send her datagram to the base station (Station A) in her cell, and Station A sends it to the base station in Bob's cell (Station B) and vice versa. But if Alice and Bob are close to each other (e.g. they are in the same cell), there is a better way to communicate with each other – directly exchange datagrams between each other rather than send datagrams to the base station. However, for exchanging datagram directly, they must be aware of the existence of each other, in other words, they need to discover each other. It is also a peer discovery problem. The most straightforward way is one device (say the device of Alice) broadcasts its existence. However, our mobile devices are usually not able to generate signals that sufficiently strong, so that the broadcast can only cover a small area. The method to solve this problem is requiring the base station to broadcast for the mobile device. In other words, the mobile devices inform the base station of its existence and the base station broadcasts its existence in the whole cell.

There are more examples of peer discovery in different applications. For example, Gnutella broadcasting PING message to discover other peers [9][10], the idea behind this method is similar to the one behind ARP Request. The Konark [11] discovers contents relying on the introduction of other peers. The idea is similar to the one behind MLDHT BitTorrent clients. Because the ideas behind

these applications are close to the ideas of the examples we mentioned above, we will not elaborate on it.

In the examples above, we see three different strategies for peer discovery, ARP request uses broadcast strategy, old version BitTorrent uses central service and MLDHT BitTorrent uses an introduction based peer discovery strategy. For the fourth example, the base station serves as a central service. So the peer discovery in the fourth example is done by requesting central party service for broadcasting. It is a broadcasting strategy and it is also a central service strategy, so it can be seen as a hybrid strategy.

Though all these strategies work well in their cases, we can still see the pros and cons for them.

The broadcast strategy will consume too much bandwidth in a large network. For a network with n nodes, if all nodes launch a broadcast, there will be $n(n-1)$ messages generated, if n is large, the bandwidth consumption is not acceptable. Hence the broadcast strategy is not scalable. For ARP request, because the size of an Ethernet is usually small, broadcast will not cause serious traffic jam, but for applications with a large network, broadcast is not an ideal strategy

Introducing a centralized entity will easily solve the peer discovery problem, but it will introduce new problems: the centralized entity will be the performance bottleneck. Besides the performance issue, a centralized tracker will also perform as the single point of failure, once it is taken down by attackers, the whole swarm will break down.

The strategy of DHT BitTorrent client is a desirable one. The idea that peers rely on other peers to discover new peers, rather than relying on a centralized tracker, shows significant potential. The central node will not serve as a performance bottleneck – because there is not a central node at all. But besides the performance aspect, we also need to concern about security issues. Unfortunately, such strategies are not perfect in security aspect, though there is not a single point of failure, it is still vulnerable to some kind of attacks. The previous research demonstrates feasible attacks on Main Line DHT BitTorrent Clients [12]. Given the popularity of Peer-to-Peer system, the security issue of peer discovery strategies similar to Main Line DHT BitTorrent clients should be investigated.

The Tribler Project provides a chance to investigate similar peer discovery strategies. Dispersy is a module of Tribler which is responsible for peer discovery, packet handling and persistent storage; the peer discovery functionality lies in a module of Dispersy, named Walker. Walker conducts peer discovery relying on the help of other peers (other Walkers), this strategy is similar to the strategy of Main Line DHT based BitTorrent, hence it is also vulnerable to Sybil Attack like the one in MLDHT BitTorrent [12]. The Tribler team believes the vulnerability to Sybil Attack roots in the unconditional trust to other peers, hence it can be solved by creating a reputation system to judge whether a peer is trust worthy. Adding "trust" is difficult especially in a distributed setting where there is not a central service or a central entity that everyone trusts. But we focus on advancing the state-of-art, investigating the probability to defend attackers in the network where the majority

of the peers are controlled by attackers.

This thesis is organized as follows: Chapter 2 will describe the problem of peer discovery in a more formal way and investigate the problem we are facing with abstract models. In Chapter 3, the high-level problem in the previous chapter will be specified. Details of Tribler and Dispersy will be introduced, and the related previous works to our problem will also be discussed. Chapter 4 will demonstrate the design of the improved Walker – Transitive Trust Walker. Chapter 5 will validate the Walker Experimentally. Chapter 6 is a summarization of this thesis, we will also point out the limitation of our approaches and propose some potential methods for future work.

Chapter 2

Problem Description

Modern technology like the Internet allows people to initiate large-scale cooperation. Such cooperation does not assume existed relationship among participants – they do not need to "know" each other beforehand. In other words, it allows cooperation among strangers. However, the foundation for cooperation is trust among people but strangers have no trust among each other; cooperation without trust is dangerous, attacks may happen in anytime and anywhere. However, establishing trust among strangers is hard, especially in the large-scale cooperation on the Internet.

The easiest way to establish trust is introducing a central party that is trusted by all participants in the system. However, such system is sometimes not realistic – there is not such a central party that everyone trusts. Even so, the central party can be the performance bottleneck or single point of failure. Hence we have to face the fact that we usually do not have a central party, we have to establish trust in a distributed fashion. It is very challenging to establish trust in a distributed system where there is no central trusted party This thesis will focus on this unsolved problem. We will improve the peer discovery module of Tribler by establishing trust among peers. Our goal is to combat attacks using the established trust.

In this chapter, we will give the formal definition of peer discovery and trust. We will also propose some potential attacks and then propose a defense mechanism against them.

2.1 Peer Discovery

Before discussing the way to establish trust, we should first introduce the concept of peer discovery. In the previous chapter, we use a story to introduce the concept of peer discovery: you want to find out the living address of your old friend, but you have no idea who knows that, so you have to ask other people about the address. Other people might not know the address, but they can introduce you with other people who might know the address. By this way, you will get closer and closer to the correct address.

Now we need to formalize the concept of peer discovery:

Definition 1 (Network Model) *In a directed graph $G = (V, E)$ where V is the set of nodes representing peers and E is a set of directed edges (i, j) indicating peer i knows the address of peer j, where i,j $\in V$. V = $(V_h) \cup (V_s)$ where (V_h) is a set of honest peer and (V_s) is a set of evil peers. For an edge (i, j), if i is an honest peer and j is an evil peer, we call the edge (i, j) as attack edge.*

Definition 2 (Peer Discovery) *A node i in the Network in Definition 1 can freely visit any node j asking for the address of node k when there is an edge (i,j),node k can be a specific node or a random node; node j will then reply i with address of node k or other relevant nodes; knowing a new node will result in adding a new edge in the directed graph. In such process, we call node i as introduction requester, node j as introduction responder, node k as target node or target peer.*

The address in Definition 1 is an attribute that uniquely locates a peer, it is the only information you need to contact with that peer. For example, it can be a (street number, house number) pair for a person or an (IP address, port) pair for a running machine on the Internet.

2.2 Potential Attacks

The model we show above actually enforce two implicit rules on all peers in the network:

- First, peers have the responsibility to help other peers. That is to say, as a peer, when other peers ask for help, you have to help them – either give the address of the target peer or introduce other peers to them.
- Second, when a peer receives an introduction from another peer, it should unconditionally trust the introduced peer.

The two rules are very rigid hence can be utilized by attackers to impair the benefits of other users or directly do harm to the whole network. We propose two kinds of potential attacks using the two rules:

First, the attacker can utilize the introduction responsibility of a peer. Because the peer is obligated to respond to any introduction request, the attacker can attack the peer by sending an enormous amount of introduction request. Hearing this attacking approach, one might come out with the concept of Denial of Service Attack (DDoS), where attackers flood the target machine with superfluous requests, that will keep target machine too busy to react to other legitimate requests. As a consequence, attackers can make the resources of the target machine unavailable to legitimate users [13]. For DDoS attack, there are already many researchers investigating it, Mirkovic creates a taxonomy for DDoS attack and defense [14]. For example, a peer as a victim can protect himself by Filtering Strategy [15],

where it can partially abandon its introduction responsibility to some other peers. But there is no strategy which can perfectly protect the victim from DDoS attack unless it completely abandons his introduction responsibility, but that will greatly undermine the cooperation protocol. Though such attack is hard to defend, it still has downsides for attackers: the attackers also need to pay the price, attackers need to mobilize a lot of machines to send messages to the victim; but mobilization is never free. Sometimes the DDoS attackers have to pay more than the victims, that will limit the usefulness of DDoS attack. In later chapters, we will show the fact that in Tribler arsenal, introduction request based DDoS attack will inflict more resources loss on attackers than the victims.

Second, attackers can utilize the unconditional trust from peers to impair the system. Honest peers believe the introduction from other peers can help them get closer and closer to the target address, but with attackers in place, this belief does not hold. An attacker can lie to honest peers by intentionally introducing irrelevant peers to them. A peer may spend a few minutes or even hours to make a progress in discovering the target peer, but an attacker can ruin all those efforts by introducing it with a faraway peer. The attacker can even introduce the honest peer with one of its accomplices. By doing so, the attacker can completely prevent the honest peer to locate the target peer. For convenience, we will call this attack as Misleading Attack. The cost of the attacker to launch Misleading Attack is very low – spending a second or even a few milliseconds to make a lie, but the gain of such attack is a lot – ruins all previous effort of a peer; the negligible cost and the high gain make this attack attractive for attackers. The only difficulty for the attack is: the attackers have to passively wait for the introduction request from other peers – if the victims do not come, the attack has no way to begin. Therefore, such attack is infeasible in our physical world – the attackers can, of course, mobilize people to help him, but compared with the honest people in the world, the helpers are few in number, hence the chance that honest peers encounter attackers or their helpers is slim. However, in the cyberspace, the difficulty can be overcome. In the physical world, the interaction between you and others are done through your body, your body is actually an agent of your mind. In this world, everyone has only one body. The mind and body have a one to one mapping. However, in the cyberspace, that is not the case. People in cyberspace interact with others via identities like Facebook account, email account, some forum accounts or other accounts. A person can easily have multiple email accounts or other accounts hence obtains multiple identities. Before further discussion, we need to define following terminology:

- **Entity**: A natural person, an organization or anything which has a mind.
- **Identity**: An agent of an entity, which is used for interaction with other entities.

Though people in cyberspace can, in theory, have multiple identities in the same domain like Twitter and Facebook, many systems still have an implicit assumption

that an entity only has a single or a few identities; so that a single entity can only impose limited influence on the system. Not surprisingly, the assumption does not always hold true. In such cases, an entity creates a lot of identities and pretend that those identities are controlled by different entities. We call such identities as "Sybil". The term "Sybil" is introduced by Douceur in 2002 [16]. A famous example for Sybil is Trump's bot event [17]. People find out that one of the Trump's fan account is actually a bot controlled by an unknown organization, to be even worse, people doubt that there are more bots in Trump's fans account. That is to say, some organizations create some fake identities and use them for certain goals. They may want to create an illusion that Trump's opinions have a lot of supporter or opponents. We do not want to investigate the goals of the organizations which control these bots, we just want to show the ease of creating new identities in cyberspace. Let's get back to the second kind of attack we discuss above. In the network, when we talk about a peer, we are actually talking about the entity behind its identity. A natural logic is that we "weight" to entities rather than identities, hence the votes from multiple identities controlled by a single entity should be count as one; and when a peer already get the introduction from one entity, it should not request introductions from other identities (Sybils) controlled by this entity. However, because it is hard to reveal the entity behind this identity, we cannot tell the difference between a normal identity and a Sybil. As a result, when we encounter multiple Sybils controlled by a single entity, we will still consider them as multiple entities. Therefore, attackers can create a lot of Sybils to create an illusion that there are a lot of entities in the network. Image that there are 90% of identities are Sybils controlled by a single attacker, in such case, the victim will have a very high probability to encounter Sybils controlled by the attacker, the probability is high enough to make Misleading Attack feasible. Notice that the resource needed for creating Sybils in cyberspace is usually few, that makes the cost of Misleading Attack acceptable.

Compared the cost and gain of attacks in the two scenarios, we believe the Misleading Attack in the second scenario is more critical. So, this thesis will focus on developing a defense strategy against such attack. Now we formally define Misleading Attack:

Definition 3 (Misleading Attack) *In the peer discovery process defined in Definition 2, the behavior that an introduction responder intentionally introduces irrelevant peers to the introduction requester is called Misleading Attack.*

2.3 Defense mechanism

In our daily life, people have already developed defense mechanisms to Misleading Attack, people will not buy a flight ticket to the other side of this planet based on the introduction from a stranger; but if the introduction comes from your parents or good friends, you may do so. The difference is that you trust your parents and your friends more than a stranger. The trust roots in the logic that your parents and your

Figure 2.1: one of the bot in Trump's fans

friends brings you a lot of benefits in the past, so they are not likely to suddenly obtain some motivations to do harm to you. The defense mechanism basing on such trust has been proved to be successful, with the trust among people, we can bravely embrace various kinds of cooperation with significantly less chance to be hurt. Given the success of the trust in daily life, we believe by establishing trust among peers, we can combat the Misleading Attack in cyberspace. Now we define the trust and the trust-based peer discovery strategy in our model:

Definition 4 (Trust) *In a network of Definition 1, for a Peer A, trust is a value assigned to all peers based on the A's knowledge about their historical behaviors. The trust is calculated by* $trust_{Ai} = T(H_{Ai})$ *where* $trust_{Ai}$ *is the trust that Peer A gives to Peer i,* H_{Ai} *is the historic behaviors of Peer i in the perspective of Peer A. And* T *is a function mapping the historical behaviors to a real number.*

Definition 5 (Peer Discovery Strategy) *A Discovery Strategy of Peer A can be defined as* $v_{result} = S(G_A, trust_A)$*, where* G_A *is the network topology in the Peer A's perspective, and the* $trust_A$ *is the trust value for other peers in Peer A's perspective. And* v_{result} *is the next peer to visit, which is the output of the strategy.*

2.4 Research goal

In this thesis, our goal is to investigate the usefulness of peer discovery strategies basing on trust in combating the misleading attack with Sybils.

However, the models in Definition 1 to Definition 5 are all very abstract, they cannot be directly applied and tested. To achieve our research goal, we need a specific application who meets our network model and peer discovery model in Definition 1 and Definition 2. And then specify a Misleading Attack defined in Definition 3, basing on this specific application. Addressing the specific Misleading Attack in a given application, we should develop an specific function T for Trust calculation and develop an specific Peer Discovery Strategy basing on the trust algorithm.

As we mentioned in the previous chapter, Tribler project provides a chance for us to achieve our research goal. Tribler has a module named Dispersy and Dispersy has a module called Walker, which is responsible for peer discovery task. Peer discovery in the network of Dispersy exactly meets our model in Definition 1 and 2. Hence Dispersy is also vulnerable to the Misleading Attack in Definition 3. But for now, Dispersy Walker does not have a bookkeeping system for historical behaviors of other peers. So there is not a Trust calculation function or a trust-based peer discovery strategy. Given this, we can summarize our research goals of this thesis below:

- Improve the Walker by adding a bookkeeping system to store the historic behaviors of other peers
- Improve the Walker by adding a trust calculation system basing on the historic behaviors of other peers.
- Improve the Walker by adding a trust-based peer discovery strategy.
- Validate the peer discovery strategy experimentally

The relevant details of Tribler and related background knowledge will be discussed in the next chapter.

Chapter 3

Trust Based Peer Discovery

We already introduce the concept of trust-based peer discovery in the previous chapter, but as we mentioned, before we test the usefulness of the trust-based peer discovery strategy, we have to apply it to a specific application. The application we choose is Dispersy. By now, we have not introduced any engineering details of Dispersy. In this chapter, we will introduce Dispersy in details as well as the related work for the thesis.

3.1 Peer Discovery in Dispersy

Dispersy is a software which provides convenience for developing distributed systems. It provides functionalities of port listening, conversion between Message and binary string, persistent storage, Message creation and handling, peer discovery and NAT puncturing. By using Dispersy, developers of distributed systems can put attention on designing the protocols in higher levels without worrying about details in lower level.

By installing Dispersy on all machines in a distributed system, developers can enforce protocols on all those machines. Dispersy provides support on many aspects of distributed system development. Many of its functionalities are out of scope of this thesis. In this chapter, we will only introduce the relevant functionalities.

We have introduced some terminologies in the previous chapter, but in Dispersy context, they have more specific meanings. In the following introduction, we will use following terminologies:

- **Peer**: A Peer is a running Dispersy instance
- **Entity**: An entity is an individual or organization who controls at least one peer.
- **Identity**: An identity is a unique name of a peer. Dispersy uses two kinds of identity, the first kind is a public key generated by a peer itself, the second kind is a 20 bytes SHA1 hash of the public key.

3.1.1 Message

Message is the basic unit sent between peers for communication, as its name, a Message instance is a message used to exchange information among peers. A Message instance consists of a Header, an Authentication section, a Distribution section, a Resolution section, a Payload section. The Payload section can be customized by developers while the rest sections are determined by preset policies.

- **Authentication**: this section is used to indicate the identity of relevant peers of this Message. Dispersy supports multiple kinds of Authentication, but only two kinds are relevant with this article – "MemberAuthentication" and "NoAuthentication". In MemberAuthentication, the Authentication section contains the identity of the Message sender. If a Message applies MemberAuthentication policy, it will append a signature using the public key contained in its Authentication section. For NoAuthentication policy, the Authentication section in the Message is empty, and there will be no signature appending to the Message.
- **Header**: the Header contains the Dispersy version, Community version and the Community ID of the sender. The Header indicates which Community creates this Message and which Community should handle this Message.
- **Distribution**: In this thesis, the only relevant Distribution is "DirectDistribution" which contains the sending time of the Message. The time is based on distributed clock similar to Lamport Clock [18]
- **Resolution**: In this thesis the only relevant Resolution is "PublicResolution", which is empty, contains no data.
- **Payload**: Payload is namely the payload of the Message, it is the effective content of the Message. The content of the Payload vary over different Message types, we will discuss it later.

3.1.2 Community

A Community is an overlay of the network, it contains its own Messages set. The overlay is established among peers by deploying same Community instance on all peers. Messages are defined in Community, only defined Message can be created and handled by the Community. A single Dispersy instance can run multiple Community instances. Community instances of a specific Community type share the same Community ID, which is a 20 bytes string uniquely identify the type of Community, for example, all Community instances of Multichain Community share the same Community ID while all Community instances of Channel Community share another Community ID. A Dispersy instance is responsible for port listening, it will deliver Messages to one of its own Community instances running locally according to the Community ID in the Message Header. For other Community instances which should not receive this Message, this Message is invisible.

While different types of Community instance have different Message set, they share a common set of Messages which are responsible for peer discovery. The logic which is relevant to creating and handling those Messages are called "Walker". The Walker is not an independent module, it is a set of functions scattering in a Community class, hence it is hard to maintain and further develop. The Tribler Team has a plan to take out the Walker and form an independent module. It provides a good chance to add new features to the Walker.

3.1.3 Walker and Peer Discovery

The peer discovery in Dispersy fits the peer discovery model in the previous chapter, peers rely on the introduction of other peers to discover new peers.

Similar with peer discovery of old-version BitTorrent clients, there are trackers in Tribler network for bootstrapping use. Every peer in the network starts with a peer list, which contains nothing but the address of trackers. Hence the trackers are the only peers that a newly joined peer can contact. A tracker, once being visited, will reply with the address of a random peer it knows, and store the contacting peer in peer list. After getting the address of another peer from trackers, the fresh peer can contact that peer, get the address of another peer and move on.

There are some Messages involved in the peer discovery process: introduction-request, introduction-response, missing-identity, dispersy-identity, puncture-request and puncture. The details of those Messages are listed below:

- introduction-request: It is a message generated when a peer wants to request another peer for introducing the address of a random peer to it. For example, when peer A wants peer B to introduce a random peer in peer B's peer list, peer A should send an introduction-request to peer B.

- introduction-response: It will be created when a peer receiving an introduction-request. The introduction-response contains the address of a random peer, it should be sent to the requesting peer.

- missing-identity: When peer A is contacting with a peer B without knowing its public-key, A can send a missing-identity to request its public key. In version 1 Community instances, the peers put the SHA-1 hash of their public key as identity in Authentication section of the Message instance, instead of a public key. That causes frequent use of missing-identity message, the Tribler team has a plan to upgrade the Community, require all instances to put their public key instead of SHA-1 hash in the Message, once the upgrade finished, the missing-identity message will be removed

- dispersy-identity: It is a message used to reply to a missing-identity message. It contains the public key of the requested peer. Like missing-identity Message, once the Tribler Team upgrades all Community to version 2, the dispersy-identity Message will be removed.

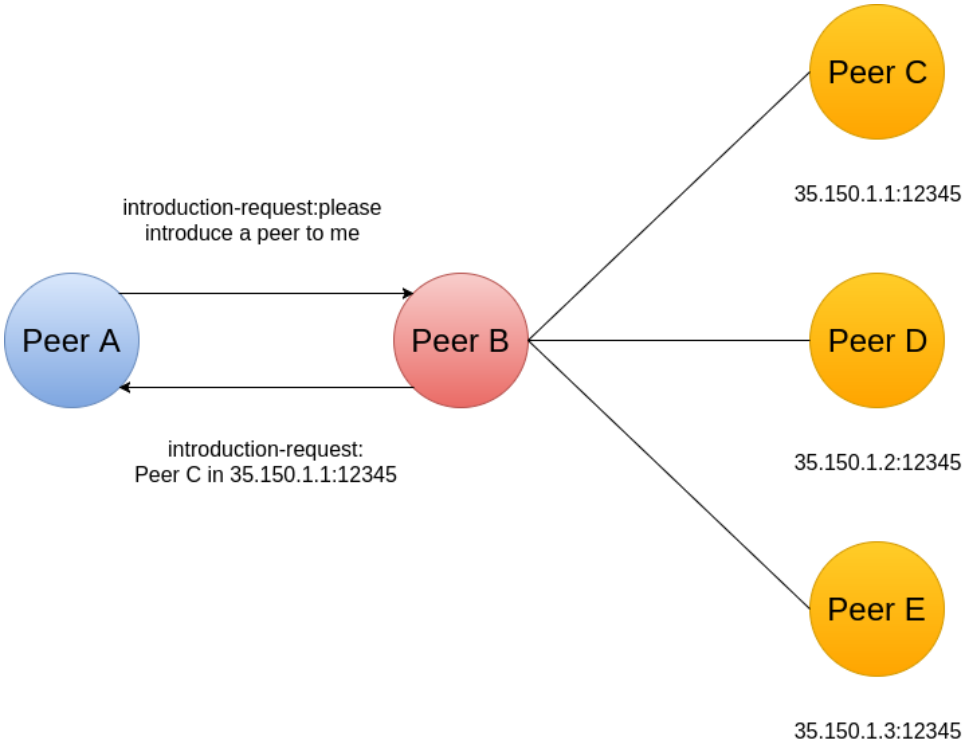

Figure 3.1: Illustration for introduction-request and introduction-response

In an ideal network, the above 4 Messages should be enough for peer discovery task. However, the presence of Network Address Translation (NAT) greatly hinder peer discovery task. A NAT is a module of Internet infrastructure to solve the problems that there are not enough IPV4 address for all devices on the Internet. The presence of NAT enables multiple devices with different local addresses to share one public address. For example, in a network, device D with IP 192.168.1.2 and device E with IP address 192.168.1.3 can share the same public address 35.157.80.100. For the observer outside the local network of devices D and E, they appear to have the same address 35.157.80.100 (but may run on different ports). While NAT offers convenience for sharing public addresses, it brings in obstacles for communication of Peer-to-Peer systems. There are many types of NAT, in this thesis, we only discuss the typical NAT. Such a NAT will block all inbound packets to a specific port unless there are outbound packets coming out from the port recently. The following scenario will demonstrate the obstacle: peer C run with 192.168.1.3, it shares a NAT with other peers, and it is mapped to public address 35.157.80.100 on port 12345. peer A knows peer C is at 35.157.80.100:12345, but it cannot contact peer C because C never send packets to peer A, C's NAT hence blocks all packet from peer A, C will receive no packet. To solve this, A and C needs cooperation. C should be aware of the incoming packet of A beforehand and send a packet to A. That packet will "punch a hole" in its NAT

and allows packets from A coming in. To enable such cooperation between A and C, the current Dispersy Walker uses the following two Messages:

- puncture-request: It is a Message containing the address of peer A which wants to send packet to the receiver of puncture-request (peer C), the receiver of puncture-request should send a puncture Message to the peer A.
- puncture: It is a Message that should be sent after receiving puncture request. The puncture Message may not be received by peer A because the presence of A's NAT, but that does not matter because the goals of puncture Message is to puncture holes in peer C's NAT.

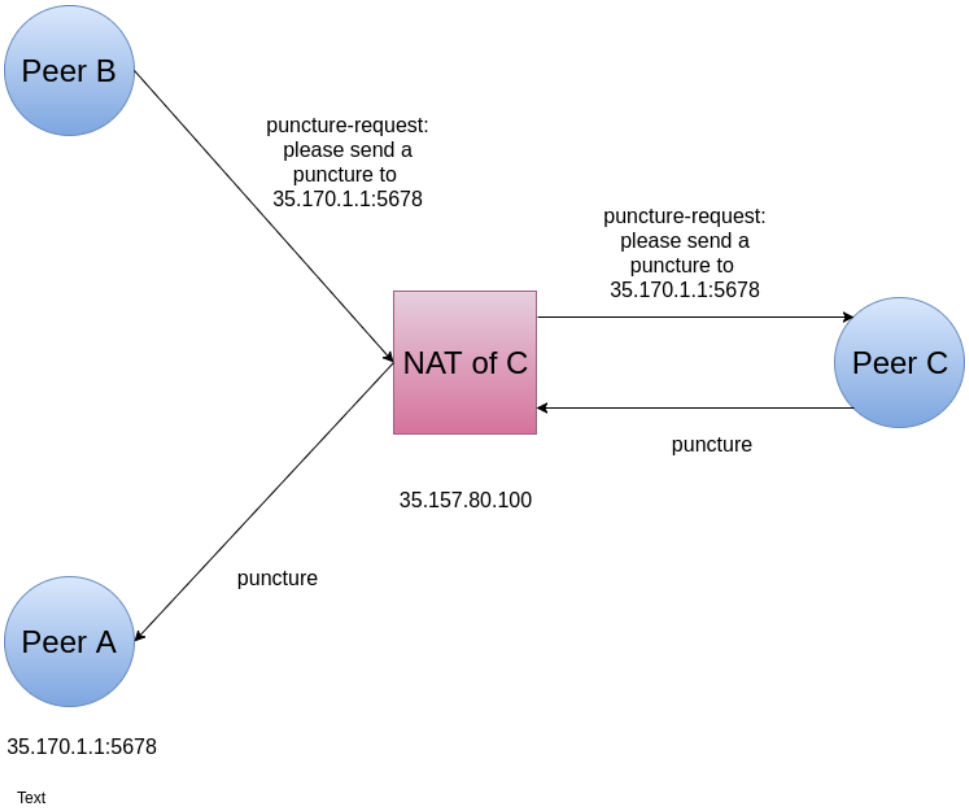

Figure 3.2: NAT puncturing

Now we put the workflow of six mentioned Message together:

- 1.Peer A sends introduction-request to peer B
- 2.If B knows the public key of peer A, skip to 4, else, B will reply with a missing-identity Message to A.
- 3.When receive missing-identity Message from peer B, A will reply it with a dispersy-identity Message containing A's public key.

- 4.Peer B sends introduction-response containing the address of a random peer C that B knows. At the same time, B will send a puncture-request to C, containing address of A.

- 5.C will send a puncture to A, hence open the hole in C's NAT.

With these six Messages, most peers of Dispersy network should be able to discover each other and puncture holes in most of the NAT, but for some NATs which apply very strict policies, the NAT puncturing procedure will not work. But this thesis does not aim to solve the limitation of NAT puncturing mechanism, so we will not further discuss it.

Need to notice: the hole in the NAT will not open forever, the lifespan of hole is determined by the configuration of NAT, but usually less than 60 seconds. Hence the lifespan of a peer is less than 60 seconds. This is an important nature even in the cases there is no NAT. We will discuss it later.

For convenience, in the rest of this thesis, we call the action that Peer A send an introduction to peer B as "taking a step" or "visiting a peer".

3.1.4 Misleading Attack in Dispersy Walker

Recall the Misleading Attack we mentioned in the previous chapter, attackers can mislead introduction requesters to irrelevant peers hence manipulate the peer list of the victim (introduction requesters). Now, with the knowledge of Dispersy Walker, we can now design a Misleading Attack in Dispersy Walker: When a Sybil controlled by attackers receive introduction-request from honest peers, the Sybil will intentionally introduce a peer favored by the attackers (e.g. another Sybil). Though the victim will decide which peer to visit randomly, therefore it may not visit the introduced peer in next step, but if the number of Sybils is high enough, the attackers can fill the peer list of victims with a lot of Sybils hence ensure the victims have high enough chance to visit Sybils in next step.

3.2 Storing Historical Behaviors

To implement a more sophisticated Walker to prevent visiting Sybils, We need a bookkeeping system to store historical behaviors associated with every identity. The first attempt for storing historical behaviors in Tribler is BarterCast [19]. In BarterCast, a peer provides voluntary reports about their interaction with a third party. However, peers have strong motivation to hide the interactions that impair their reputation, they can also provide fake reports, tampering their own history. Hiding or tampering will fundamentally impair the reliability of the bookkeeping functionality. Addressing this flaw, Tribler team created a new, tamper-proof bookkeeping system – MultiChain. Multichain was first introduced by Norberhuis in [20], but now its name change to Trustchain, keep in mind they are different names for the same thing. In this thesis, we will mainly use the term "Trustchain".

The concept of Trustchain is similar to the concept of the famous Blockchain introduced in the original Bitcoin document [21]. Before we discuss TrustTchain, we should first introduce Blockchain.

3.2.1 Blockchain

Blockchain is a data structure help peers in distributed systems to reach a global consensus of the order of the happening time for a series of interactions. The most famous application of Blockchain is supporting the transactions of between peers in Bitcoin. Bitcoin is a cyber currency which aims to provide credit without a central authority. As a kind of currency, Bitcoin should support paying functionality that supports a peer (Alice) to transfer a certain amount of Bitcoins (e.g. 5 BTC) to another peer (Bob). Similar to a fiat currency like Euro, the way to transfer money is making an announcement that Alice transfer a certain amount of money to Bob, and prove that you are Alice. In the case of transferring Euro, the announcement should be sent to the bank which holds the account of Alice, however, in the Bitcoin scenario, there is not a central organization like a bank. The announcement (we will refer it as announcement A1) will be broadcast to all the peers in Bitcoin network. Because of the lagging of Internet, the announcement needs time to reach every peer, and different peers are likely to receive this announcement in different time. This nature provides benefit to launch an attack named "double spend", Alice can sign another announcement (announcement A2) to transfer the Bitcoin, which should be transferred to Bob, to Alice herself instead. Because of the lagging of Internet, it is likely that many peers receive announcement A2 before announcement A1, hence they will validate announcement A2 and invalidate announcement A1. If there are enough peers validate announcement A2, the network as a whole will invalidate announcement A1, as a consequence, Bob will not receive the Bitcoins he ought to receive.

The announcement mentioned above is called "transaction" in Bitcoin system. The root of the double-spend problem is lacking a way to issue time stamp on the transaction, ensuring transaction A1 has an earlier timestamp than transaction A2. It is easy to require the transaction initiator of the transaction (namely, Alice) to issue the time stamp, but Alice has strong motivation to lie on the time stamp. So Bob will not trust the time stamp issued by Alice. The transaction will not happen because of lacking mutual trust. Bitcoin system solves this problem by creating a global consensus on the order of transactions, the consensus is reached by using Blockchain.

Blockchain is namely a "chain of blocks". A Block can be seen as a "box of transactions", a set of transactions are grouped together and put into a Block, and the Blocks will be chained together to form a Blockchain. A Blockchain can be described as $Block0, Block1, Block2, Block3, Block4, Block5...$ where $Block0$ is the first Block; the transactions in $Blocki$ happens before transactions in $Blockj$ if $i < j$. Every peer can construct its own Blocks and Blockchain, but there is only one valid Blockchain in the Bitcoin system – the longest one. To prevent peers

favoring their own Blockchain so that the consensus cannot be reached, the system requires peers to consume a lot of computing power in creating a new Block. Although peers are allowed to create blocks and they have the right to choose which transactions to be put in a specific block, they cannot validate this block easily. To validate a Block, a peer needs to solve a mathematical puzzle which has no other way to solve but randomly guessing numbers. This mechanism prevents peers' own Blockchain to grow too fast, force them to use a common Blockchain, namely, the longest Blockchain. To make sure a peer's own Blockchain outgrow the current, global, Blockchain, the peer need to have greater computation power than the sum of computation power of the rest of peers all over the world, which is not realistic.

Some property of Blockchain prevents peers to intentionally tamper or hide Blocks. To better illustrate it, we assume there is a Blockchain $Block0...Blockn$. Every Block (except $Block0$) should include the hash value of the previous Block, for example, $Block3$ should contain the hash value of $Block2$. The hash value of the previous Block serves as a "hook" to all contents of the previous Block. If a peer tamper $Blockx$, then the hash value of $Blockx$ changes, but the $Block_{x+1}$ still points to the correct $Blockx$ rather than the tampered one, and because the correct $Blockx$ does not exist anymore, $Block_{x+1}$ will points to a non-existing Block, that will make $Block_{x+1}$ invalid, hence the all following Block after $Block_{x+1}$ will be invalid. So, if a peer wants to tamper a Block, it needs to tamper all Blocks following this Block, it is not realistic to do so. The same mechanism can also prevent peers hiding some Blocks.

A Blockchain contains all transactions of all peers around the world, in other words, it keeps the all historical behaviors of all peers. Although this nature shows benefits, it also has major downsides. First, all peers need to have the whole Blockchain to participate in Bitcoin transactions, a newly joined peer needs around one day to collect and analyze the full Blockchain before it can start working, in addition, as the number of transactions grows larger, this warming-up time will also increase; furthermore, to validate a transaction, a peer need to validate the whole history of every single Bitcoin involved in this transaction, that also cost computation power. To be even worse, as the length of Blockchain grows, this problem will be even more serious in the future.

Second, creating block cost too much time, that significantly limits the number of transactions can be validated per second. Currently, the Bitcoin system can validate 7 transactions per secondwhile VisaNet have a peak performance of 56 thousand transactions per second [22][23]. If Tribler directly uses Blockchain, such slow transaction processing speed may cause problems. So, Tribler team develop a variant of Blockchain – Trustchain

3.2.2 Trustchain

The concept of Trustchain is first introduced in [20]. Later, a major upgrade is introduced in [24]. This chapter will introduce the concept of Trustchain basing on the work of [24]

Unlike Blockchain, Trustchain is not designed for currency system, the "transaction" of the Trustchain is about the amount of downloading and uploading between two peers, counted in megabytes. In fact, there is not a data structure named "transaction" in Trustchain: a Block in Trustchain only describe one interaction between two peers, unlike Trustchain where a Block contains multiple transactions. For better sketching the image of Trustchain, we will still use the term transaction".

Notice that, as we mentioned, the Trustchain has another name "Multichain", as its name, Trustchain System has multiple chains of Blocks. Unlike Blockchain who needs a single chain to reach a global consensus concerning the order of interactions, Trustchain system does not require such a strict consensus. Every peer in the Trustchain system is responsible to maintain their own chains that store all historical behaviors (transactions) of their own. A peer does not necessarily store Blocks of others. But one peer can also request transactions of other peers. That means: it is unnecessary to requesting the whole history of the network before joining the network, but a peer can request the history after joining the network if it needs.

A Trustchain Block contains signature from only one involved peer, however, to validate an interaction record, we need the signature of both involved peers, so Trustchain uses two Blocks to depict an interaction. The two Block contains the same uploading and downloading records, but contains different signatures – two signatures of two involved peers, one for each. Similar to Blockchain, all Blocks depicting the historical interactions of a specific peer should be chained together, every Block contains a hash of the previous Block. For the two Blocks involved in same interaction, they contain different "previous hash" field, pointing to the previous Block of the two involved peers, also one for each. The two Blocks are also linked together, in fact, the historical interaction can only be validated with both Blocks presence. Figure 3.1 is the illustration of Trustchain used in Pim Veldhuisen's work [24], the columns represent chains of different peers.

When interactions between peers happen, a new Block can be created. Because of the absence of global consensus, creating Block does not need any proof-of-work effort like solving mathematical puzzles in Blockchain. The procedures to create a new Block are:

- Step 1: For the two peers, the one who uploads more data should initiate the creating of the new Block. We call this peer as "requester".

- Step 2: The requester will create a Block, fill uploading and downloading field based on its own perspective on the interaction; sign the block and add the hash of requester's previous Block to it. Send the newly created Block to the other peer in this interaction, we call this peer as "responder"

- Step 3: The responder will check the receiving Block on uploading and downloading field based on its perspective of the interaction, if they match the perspective of responder, the responder will creating a new Block con-

taining the same uploading and downloading field, sign it, point them to the Block created by requester. Send the new Block to requester.

- Step 4: the requester receives the Block, now requester and responder both have two Blocks depicting this interaction.

3.3 Scoring System and Walking strategies

The Trustchain has been implemented in Tribler, every Dispersy instance has a Multichain Community which is responsible for managing the creating and storage of Trustchain Blocks. Multichain Community also defines two Messages allowing peers to share Blocks in their storage, the details of the two Messages will be discussed in next chapter. With Trustchain in place, a peer can now be aware of the historical behaviors of other peers hence it can now use a peer discovery strategy based on the history of other peers. The peer discovery strategy can be further divided into two part – first, generate a score (trust value) for peers basing on their history; second, determine which peer to visit based on the score. Fortunately, Tribler team has investigated both of them.

3.3.1 Scoring System

We use the term "scoring system" to refer a system which assigns scores to different peers. A peer with high score has a good reputation hence it will be favored by other peers. In this sense, we can also call the scoring system as "reputation system". Previous researchers of the Tribler team have investigated the scoring system and its applications: Pim Otte's research [25] describes two accounting mechanism which can assign scores. The first one is NetFlow. NetFlow is processed basing on a directed graph, where nodes in the graph representing peers and edge representing the upload from one peer to another, the weight of an edge is the amount of uploading data counting on megabytes. NetFlow calculates the reputation score of a peer B in the perspective of peer A, basing on the difference of maxflow in both direction between A and B. For example, see Figure 3.2, it shows the reputation of Peer P, T, Q in the perspective of peer R. According to the graph, R uploads 6 megabytes to P, and 8 megabytes to Q; P upload 9 megabytes to R and 5 to Q; Q uploads 3 to P and 3 to R. T uploads 2 to P and 2 to Q. Then the reputation of P, Q, T can be calculated as follow: P upload 9 megabytes to R through the path (P,R) and 3 through the path (P,Q,R), that is 12 in total. And P consumes 9 megabytes from R: 6 through path (R,P) and 3 through the path (R,Q,P), so the reputation of P (in R's perspective) is 12-9=3. The scores of Q and T can be calculated in a similar way.

The second accounting system is PimRank. The idea of PimRank can be described as follow. See Figure 3.3, it is a directed graph consisting of Multichain Block, a node represents a Multichain Block, the edge between Blocks are "hooks" we mentioned in the previous section, a "hook" is either a "previous hash" field

pointing to the previous Block of a peer, or the "link sequence number" pointing the other Block in the same interaction. A Walker start in a random Block and randomly walks via links to Blocks connected in current position (the Block that the Walker "stand" for now), and every turn a Walker has a chance of β to teleport to a random position of the graph. Similar to Markov graph, after enough walking step, the probability distribution of standing in a certain Block is static, such distribution is called stationary distribution. The probability can then be used as the score for the Block owners.

The problem of the two accounting mechanisms is that they consume too much time. In the scenarios that there are 20000 Blocks, NetFlow will take around 300 seconds and PimRank is much faster, but still takes around 10 seconds [25]. The time consuming is acceptable for uploading management scenarios (the scenarios that the two mechanisms being designed for) where an application decides which peers it should upload files to, but for a Walker, that time consuming is not acceptable: a Walker visits a peer every 5 seconds and may collect Blocks from the visited peer, that will change the graph used in PimRank and NetFlow, when the result of the PimRank or NetFlow come out, they are already out of date. So we should find a faster scoring algorithm for the Walker, the scoring algorithm is not necessary to be as sophisticated as PimRank and NetFlow.

3.3.2 Walking Strategies

For Dispersy Walker, the most important application of a scoring system is supporting walking strategies. Pim Veldhuisen [24] tests the performance of two walking strategies in his researches, he aims to evaluate the Block exploration ability of the two strategies. Although the goal of our research is seeking for proper strategies that resilient for Sybil Attack described in Chapter 2, the research of Pim Veldhuisen [24] still provides valuable information.

The first walking strategy in Pim Veldhuisen's work [24] is Random Walking. Every time a Walker needs to choose a peer to visit, it randomly picks out one of its known peers to visit. The Algorithm 1 shows the random walking strategy in pseudo codes.

Algorithm 1: Pim Veldhuisen's Random Walking strategy

```
1 for each step do
2 |   next_node = pick_random_from(connected_neighbours);
3 |   walk_towards(next_node);
4 end
5 def walk_towards(node):
6       new_peer = node.request_neighbour();
7       connected_neighbours.add(new_peer);
8       new_peer.request_blocks();
```

The second walking strategy is called "Focus Walking", which assigns the probability of a candidate (a peer to be chosen) according to its score. However, Pim's

research [24] does not mention what algorithm he uses to calculate the score. But that is a minor issue for our research. After calculating scores of peers, he ranks all peers, and assign probability basing on their ranks: the top rank peer has a probability of ϕ to be chosen, if it is not chosen, the second peer has a probability of ϕ to be chosen, if the second peer is not chosen, then take same procedures on third peers and move on, until a peer is chosen. The essence of the "Focus Walking" is introducing bias in the probability assigned to each peer: instead of assigning equal probability to every peer, it assigns greater probability to high-rank peers. So, to be scientific, we will call "Focus Walking" as "Bias Walking", or more specifically, "Pim's Bias Walking"

The pseudo codes of this strategy are depicted in Algorithm 2. Notice that, Algorithm 1 and Algorithm 2 are copied from [24]

Algorithm 2: Pim Veldhuisen's Bias Walking strategy

```
1 while uniform_random_variable() > phi do
2 |   index = (index + 1) % len(ranked_live_edges);
3 end
4 next_node = ranked_connected_neighbours[index];
5 walk_towards(next_node);
6 def walk_towards(node):
7       new_peer = node.request_neighbour();
8       connected_neighbours.add(new_peer);
9       new_peer.request_blocks();
```

Because the random walking strategy is the current strategy of Dispersy Walker, we are only interested in focus walking strategy. The strategy assign higher probabilities to high score (trust value) peers. The reasoning for this is that high score peers are more likely to contains relevant Blocks that we need. In this article, we do not care the Blocks collecting very much, but we have reason to favor high score peers too – high score peers are less likely to be a Sybil. Compare with creating an identity in Tribler, boosting its reputation is significantly harder. For creating an identity, attackers only need a few milliseconds, but for boosting its reputation, attackers need two things – enough files to be shared with honest users and the chance to share those files. Though such difficulty cannot completely prevent attackers having any high reputation Sybils, it can limit the portion of high reputation Sybils in a very low level.

Since we have the reason to favor high reputation peers, the performance of Pim's Bias Walking Strategy can provide us valuable sights. Unfortunately, the evaluation shows that the focus walker may cause load balancing issue – the walker visit high reputation peers too frequently hence it may overwhelm those peers. In our opinion, the roots of the load balancing problem of focus walker are:

- The algorithm discriminates the peers according to rank, the difference between top rank and second rank peers are big enough, let alone the difference between top peers and low reputation peers.

- The peers in the experiment have an infinite lifespan, the high reputation peers survive too long, hence receive too many visits.

So, we believe we can mitigate the load balancing problem by:

- Rank the peers by groups, rather than by single peer. We can put all high score peers in a group and low reputation peers to another group. We assign different probabilities to a group, but all group members share the same probability. By this way, we actually shrink the difference of bias in probability
- we enforce finite lifespan on all peers, but allow high score peers to live longer.

Besides load balancing problem, we have another reason to enforce finite lifespan on peers: with infinite lifespan, high reputation Sybils can stay in the peer list forever. Though high reputation Sybils are few, they are still enough in numbers to fill the peer list of a single peer.

3.4 Sybil Attack Defense

Scoring System is not the only way to combat Sybil Attack, the topology of the network can also be used in defense, previous researches have exploited its benefits. For example Yu and Kaminsky [26] identify the characteristic of the network as follow:

- 1.There are two regions in the network, an honest region and a Sybil region The two regions are connected with few edges, called attack edges.
- 2.Compared with the number of Sybils, the attack edges are few in number.
- 3.We (the honest users) are born in honest region.
- 4.Every node knows which nodes it is connected to. In other words, every node in the network is aware of the existence of the edges between other nodes and itself.
- 5.Most nodes are always online, that means whenever you send some request to other nodes, they will respond to you on time. And the network is static, the topology will not change.

Yu and Kaminsky [26] concludes the above characteristic for social networks, where nodes represent an account and edges represent some ”relationship”, for example, in Twitter, the relationship is “follow”.

In Dispersy network, the characteristic 1 to 4 all hold: as we described in chapter 2, we can use a graph to represent Dispersy network where nodes represent identities and edges represent “knowing the address”, while creating fake identities

(Sybils) is easy but introducing such Sybils to honest nodes are hard, the attack edges are few in number compared with the number of Sybils. Because the attack edges are few, the graph can be roughly divided into two distinct regions – honest region and Sybil regions rather than mixing the honest peers and evil peers together. We assume the trackers are not compromised, then a honest peer should start its peer discovery in a honest region.

The characteristic 5 does not hold because the lifespan of Dispersy peers is short, hence peers frequently time out. Therefore we can not use SybilGuard directly in Dispersy network, because SybilGuard requires a peer to store two tables in peers around it. In Dispersy network, because the frequent time out of peers, the two tables need to be transmitted frequently which wastes significant amount of resources, the amount of resources waste depends on the specific configuration, in the original paper of SybilGuard [26], the authors provide the example that for every peer in the network, it needs to store data in 2000 peers, in SybilGuard, such data only needs to be generated and stored once, but in Dispersy, because of the short lifespan (60 seconds), such data needs to be generated, transmitted and stored every 60 seconds; that is a burden for the whole system. But on the other hand, the short lifespan of peers can be used as a weapon against Sybil Attack.

Recall that the Misleading Attack in Dispersy Walker in 3.1.4. The idea behind this kind of Attack is injecting the address of Sybils into the peer lists of victims. In the perspective of the whole network, it can also be described as Sybil region injects "toxic" address to the honest region. The toxic address can only be injected via attack edges. However, the attack edges are few in number. Injecting more Sybil address can increase the probability that honest peers visiting Sybil peers hence creating more attack edges, further increase the injecting velocity of Sybil address. If the lifespans of the peers are infinite, the network will eventually evolve to a fully connected network that all peers have edges with all other peers, including Sybils. However, the lifespan of peers are finite, the Sybils will finally time out in the peer list of honest peers, causing the disappearing of attack edges. We can, therefore, get a positive loop: by preventing the probability of a peer to visiting Sybils, we can reduce the number of attack edges, hence further decrease the probability of visiting Sybils for all peers. It is a "delaying tactics" that we do not need to identify whether a specific peer is Sybil, we only need to reduce the probability of visiting Sybils so that we can delay the invasion of Sybils and kill them by using the time out mechanism. And a simple way to reduce the probability of visiting Sybils is: visiting high reputation peers more frequently, visiting low reputation peers less frequently. The details of this strategy will be discuss in the next chapter.

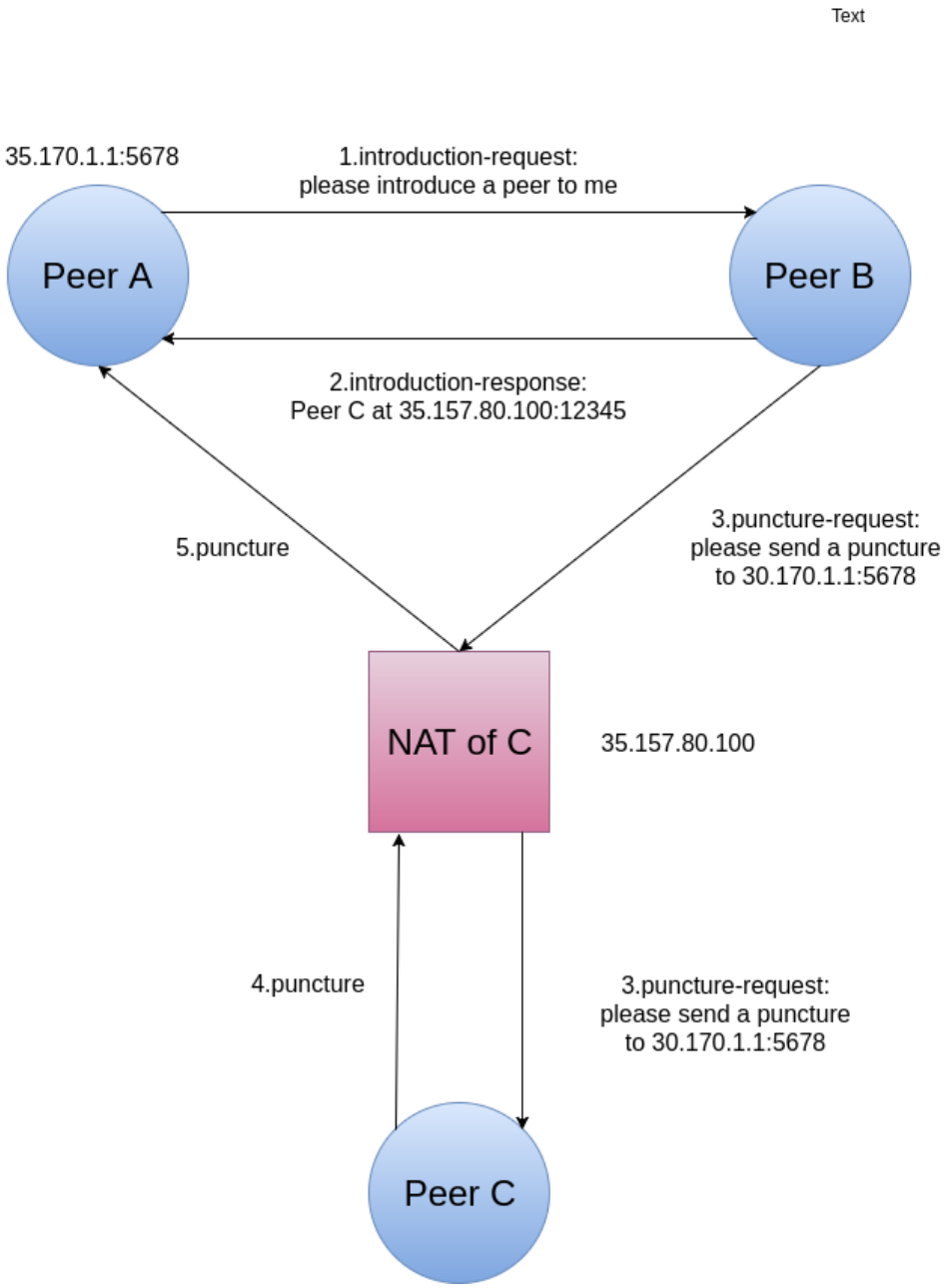

Figure 3.3: the whole work flow

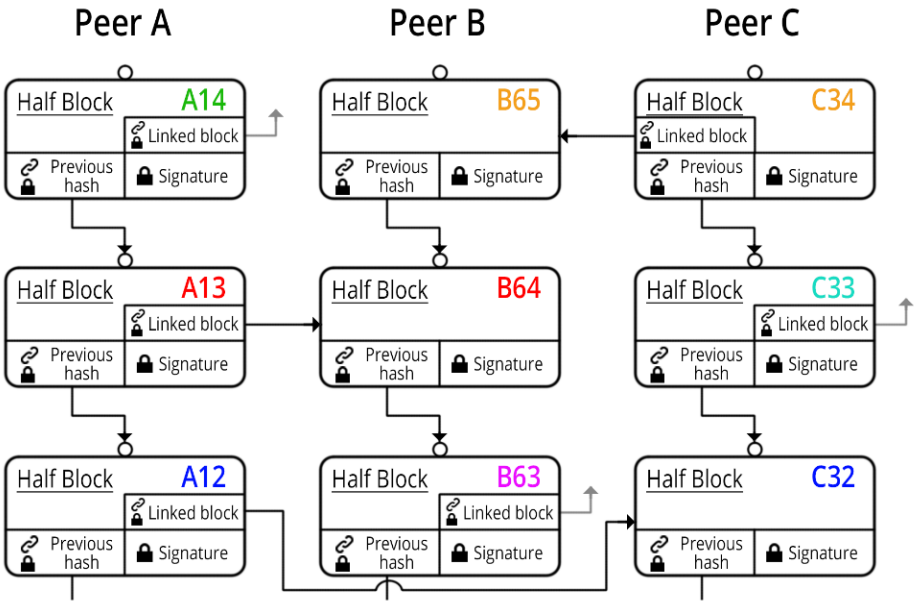

Figure 3.4: TrustChain

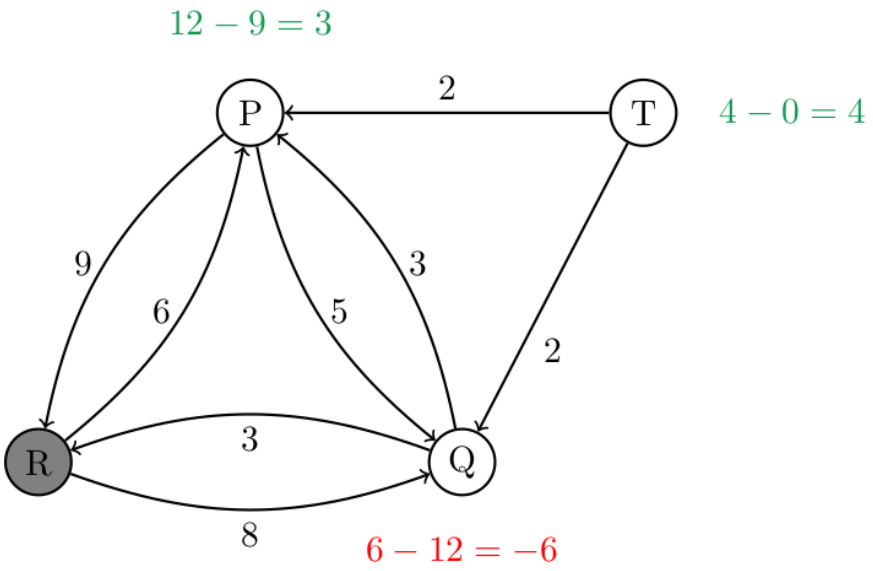

Figure 3.5: NetFlow Algorithm

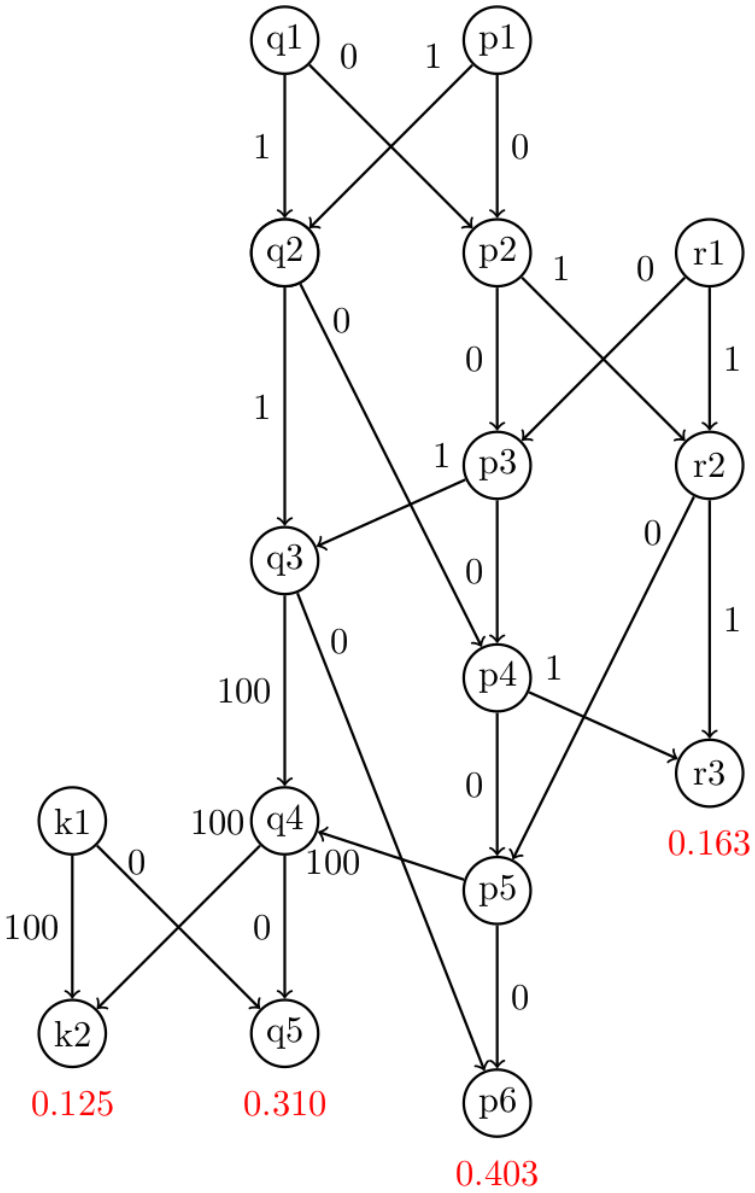

Figure 3.6: Temporal Page Rank Algorithm

Chapter 4

Design of Transitive-Trust Walker

In this chapter, we describe the design of the new Walker. We will first list the design requirements for the new Walker. Then we will discuss the design corresponding to the each requirement.

4.1 Design Requirement

We have following design requirements:

- The new Walker can collect Trustchain Block from other peers
- The new Walker should be able to calculate trust value of peers basing on the Blocks it has
- The new Walker should be able to prevent visiting sybils in some degree.
- The new Walker should be able to cover all the peers in the network with enough steps.
- The new Walker should not create serious load balancing problems

For calculating the trust for other peers, a Walker should be aware of the historical behaviors associated with other Walkers. Our Walker chooses TrustChain as the bookkeeping system for historical behaviors, but unlike Blockchain where peers have global consensus of the whole history, TrustChain does not require peers to be aware of the whole history. Hence Walkers are usually aware of their own history only. So, before calculating the trust, a Walker must collect Blocks which store the historical behaviors of other peers.

With knowledge about historical behaviors in place, the Walker should be able to calculate the trust value. The trust value will be used in peer discovery strategy.

The peer discovery strategy should be resilient to Sybil Attack, more specifically, Misleading Attack with Sybils.

Our goal is to use trust based peer discovery strategy to defend the Walker against Sybil Attack, more specifically, Misleading Attack. However, on the other hand, the new Walker is still a Walker hence it has to meet the basic requirements of Walker. As a Walker, it should be able to discover all peers in the network with enough steps, that is to say, when the number of the steps taken approaches to infinite, the Walker should be able to discover every peer in the network instead of a small subset of them. In addition, the Walker should not cause serious load balancing problems: if all Walkers in the network favors a few high reputation peers, those high reputation peers will receive too many introduction requests and end with overloaded. Our new Walker should prevent such situations happen. But slight load balance problem is tolerable.

4.2 Walker Architecture

Dispersy provides many supports for the current Dispersy Walkers.

- Port Listening: Dispersy listens on specific ports, handles incoming packets and notify Walkers of different Community instances.
- Message Conversion: The Community instances in Dispersy are responsible for conversion between Message instances and binary strings that can be transferred via Internet.
- Peers Management: The Community instances in Dispersy can store, retrieve and remove peers discovered by Walkers.
- Persistent Storage: Dispersy is responsible for storing and retrieving identities of known peers, collected Blocks.

Because the new Walker should not rely on any modules of Dispersy, we need to create modules which provide the same functionalities mention above, the architecture of the new Walker is shown in Figure 4.1

Incoming packets from the Internet will be received by the Network Endpoint, it will check the format of the incoming packets. If a packet has a correct format, it will be passed to Message Parser, the Message Parser will convert it to a Message instance that can be recognized by Walker Core. The Walker Core will handle the incoming Messages; depending on the type of the Message (introduction-request, introduction-response etc.), the Walker core may need the support from Peers Manager, Block Discovery Module, or Persistent Storage Manager.

Peer Manager allows Walker Core to store and retrieve information of peers, including the IP address, port, identity. The Peer Manager will categorize peers as trusted peers, outgoing peers, incoming peers and introduced peers. The trusted peers will be discussed later, the outgoing peers are peers that the Walker already

visited. The incoming peers are those who send us an introduction-request, the introduced peers are those introduced to us via introduction-response. The Peer Manager will also automatically clean up peers that have timed out.

The Block Discovery Module will automatically collect Blocks from other peers, the received Blocks will be pass to Persistent Storage Manager.

Persistent Storage Manager manages the database which stores the identity of peers, the Blocks collected from other peers. It also provides support for Reputation System module which will be discussed later.

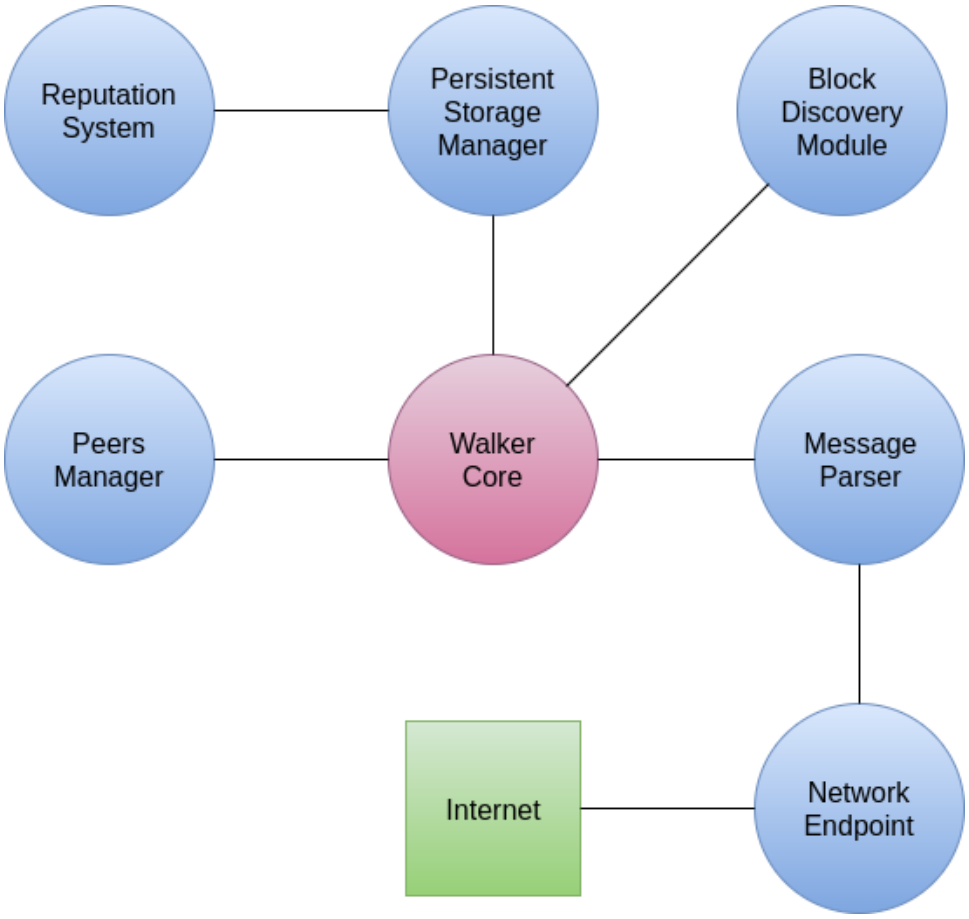

Figure 4.1: Walker Architecture

The source codes of the Walker can be found in the github repository:
`https://github.com/YourDaddyIsHere/even-cleaner-neighbor-discovery`

4.3 Block Collecting

Our reputation system and walking strategy rely on the Trustchain Blocks describing the historical interactions of other peers. As we mentioned in previous chapter,

Trustchain system does not enforce a global consensus on all peers, hence the peers do not have the global interactions records, however, our reputation system and walking strategies need the interaction records of other peers, so our Walker need to collect Blocks describing such interactions while conducting peer discovery task.

Fortunately, Dispersy allows peers to request existing Blocks from other peers, there are two Messages involved in this process: crawl-request and crawl-response.

- crawl-request: It is a Message instance created when a peer wants to collect Blocks from other peers. It contains the public key of the creator of Blocks.

- crawl-response: It is a Message instance created when a peer receives crawl-request Message from other peers. It is the response of a crawl-request and it contains a Trustchain Block.

The Blocks request/response procedure is: peer A sends a crawl-request to peer B, the crawl-request contains the public key of peer C. When peer B receives the crawl-request from peer A, it retrieves Blocks which are created by peer C in its database, if there is at least one such Block, B then sends a crawl-response contains that Block to peer A, if there are multiple such Blocks, B will send multiple crawl-response to peer A, each contains a single Block.

The following example main help you better understand the procedures: Peer B has Blocks created by three different peers. Among them Blocks C1,C2 and C3 are created by Peer C; D1 and D2 are created by Peer D; E1,E2,E3 and E4 are created by Peer E. Now, Peer A sends a crawl-request to Peer B. The crawl-request contains the public key of Peer C. When Peer B receives this crawl-request, it will reply 3 crawl-response Messages to Peer A, the 3 crawl-response Messages contain the Blocks C1,C2 and C3, one for each.

The Blocks can be collected at the moment when our Walker needs to evaluate the reputation of certain peer or collected regularly while the Walker is exploring peers. The first approaches may cause a traffic jam in the network because of the enormous number of relevant Blocks, For example, when Peer A needs to evaluate the reputation of Peer F, it sends a crawl-request to peer B requesting Blocks of Peer C. However, there are 100 thousand Blocks created by F. For calculating the reputation of Peer F, Peer A has to send a lot of crawl-request to exhaust all those Blocks, that will cause a burst in the traffics in the network. It may cause a traffic jam for the network, hinder the communication between Peer C and other peers. To prevent the traffic jam, we choose the second approach – collecting Blocks regularly from other peers beforehand. In other words, even the Walker currently do not need such Blocks, it should still collect it for future use.

Our new Walker will send a crawl-request whenever we receive an dispersy-identity message or a introduction-response from the peers. That means whenever it discovers a new peer, it will collect the Blocks that the peer has. The Blocks will be stored persistently, hence as time goes by, the Blocks will be accumulated. With

more and more Blocks, the evaluation of reputation will be more and more close to the ground truth.

4.4 Reputation System

We build our reputation System basing on a graph similar to "Interaction Graph" of Pim Otte's work[25], but our graph differs with "Interaction Graph" on some aspects, we call our graph as Trust Graph.

Definition 6 (Trust Graph) *A directed graph GT = (V, E) is called Trust Graph if: V is a set of nodes representing Dispersy peers and E is a set of edges representing upload interaction between Graph. An edge (i, j) indicates there is at least one interaction that peer i upload data to peer j.*

Basing on Trust Graph, we define the term Directed Trust and Transitive Trust:

Definition 7 (Directed Trust) *In a Trust Graph, peer i has Directed Trust on peer j if and only if there is an edge (j,i)*

Definition 8 (Transitive Trust) *In a Trust Graph, peer i has k-hop Transitive Trust on peer j if there is a path $j, x_1, x_2...i$, the length of the path is k.*

Definition 9 (K-Hop Trust) *In a Trust Graph, peer i has k-hop Trust on peer j if peer i has Directed Trust, or a t-hop Transitive Trust on peer j, where $t \leq k$*

Given a value k, the reputation system divides all peers into two class: the peers that we have k-hop transitive trust are put in the trusted group, and the other peers are put in the untrusted group. We treat the two groups different but treat any members in the same group equally.

Choosing a proper k is not a minor issue. Basically, a small k will make the trust rare and lead to the situation that our Walker has no one to trust and have to work like a fully random Walker. However, a large k means the Walker will have a high chance to trust a sybil. Image that there is only a single sybil S who has a directed trust with an honest peer H; assume that the Walker has knows the whole topology of Trust Graph, then: if we choose $k = 2$, the Walker has limited chance to trust sybil S, unless our Walker has Directed Trust on peer H. if We choose $k = \infty$, the Walker will have very high chance to trust S, as long as it trusts any peer that has transitive trust on H or S.

We can not analytically determine the optimal value of k. We can, of course, take different k to conduct multiple experiments and compare which k is better. However, because the number of natural number is infinite, we can never exclude the possibility that there is a better k that we do not know.

Therefore, in this thesis, we take a conservative attitude and choose $k = 2$, without trying to seek for a optimal k

4.5 Walking Strategies

By using the concept of K-Hop Trust, we are now able to design trust based peer discover strategies, we actually create two such strategies:

4.5.1 Bias Random Walking

Compared with creating Sybils, boosting the reputation of Sybils are expensive and hard, that implies the fact that most Sybils have low reputation. In other words, a peer which has a relatively high reputation (trust value) is not likely to be a Sybil. Therefore, compared with visiting a random peer, visiting a high-reputation peer is less likely to encounter a Sybil. Hence high reputation peers are also less likely to intentionally introduce us with a Sybil. On the other hand, if the Walker only visit high reputation peers, it is likely to be limited in some small regions consisting of only high-reputation peers, leaving the vast area of network unexplored. While such strategy may provide the highest level of security, it compromises the basic functionality of a Walker – Peer Discovery. Hence it is necessary to make a trade-off between security and functionality. To address this issue, we choose to use a random Walker which has a bias in probability – it does not treat all peers equally in probability, but assign a higher probability to trusted peers, where trusted peers are peers that we trust. The bias walking strategy is described in Algorithm 3.

Algorithm 3: Bias Random Walking

```
for each step do
    #random number between 0 and 1;
    random_number = get_random();
    if random_number ≥ 0.995 then
        next_node = a random tracker ;
    else if random_number ≥ 0.5 then
        next_node = a random trusted peer ;
    else if random_number ≥ 0.3 then
        return a random outgoing peer ;
    else if random_number ≥ 0.15 then
        next_node = a random incoming peer ;
    else
        next_node = a random introduced peer ;
    end
    walk_towards(next_node);
end
def walk_towards(node):
      new_peer = node.request_neighbour();
      connected_neighbours.add(new_peer);
      new_peer.request_blocks();
```

4.5.2 Teleport Walking

The bias walking strategy,though introducing bias in probability, is still a random walking strategy. It is random nature may make it too frequently visit peers that it already visited recently, causing a degrading in peer discovery speed. To prevent such case, we should force the Walker to visit unvisited peers: we should force the Walker to visit the newly introduced peer rather than the peers it already visited. With this strategy, the Walker will walk through a random path in the network, visit the peer along the path one by one. In this fashion, peers are less likely to visit already visited peers hence it discover new peers in a high speed. But once the peer visits a Sybil controlled by attackers, the Sybil can introduce the Walker with another Sybil, as a result, the Walker will walk deep into the Sybil region and has no chance to get back. To avoid that, we require the Walker to "teleport home": teleporting back to the starting point of the path with a probability α, once the Walker teleport back, it will randomly pick one of its trusted peers and take it as the starting point of a new random path. If there is not a trusted peer, take a random peer in Walker's peer list. The details of this strategy is depicted in Algorithm 4

Algorithm 4: Teleport Walking

```
for each step do
    #random number between 0 and 1;
    random_number = get_random();
    random_number2 = get_random();
    if random_number ≥ α then
        next_node = current_node.introduce() ;
    else
        if there is at least one trusted peer then
            next_node = random_trusted_peer ;
        else
            next_node = random_peer
        end
    end
end
```

An illustration of Teleport Walkings strategy is shown in Figure 4.2, where blue node represents the working Walker, red node represents the peer which the Walker is currently visiting, yellow peers represent the nodes the Walker visited along this random path.In addition, solid lines represent the fact that the Walker trust those nodes and know their address, and the dashed lines represent the fact that the Walker know the address of those peers but not trust them. In (e), the Walker teleports home and in (f) the Walker take a new step in a new random path

```
def walk_towards(node):
    new_peer = node.request_neighbour();
    connected_neighbours.add(new_peer);
    new_peer.request_blocks();
```

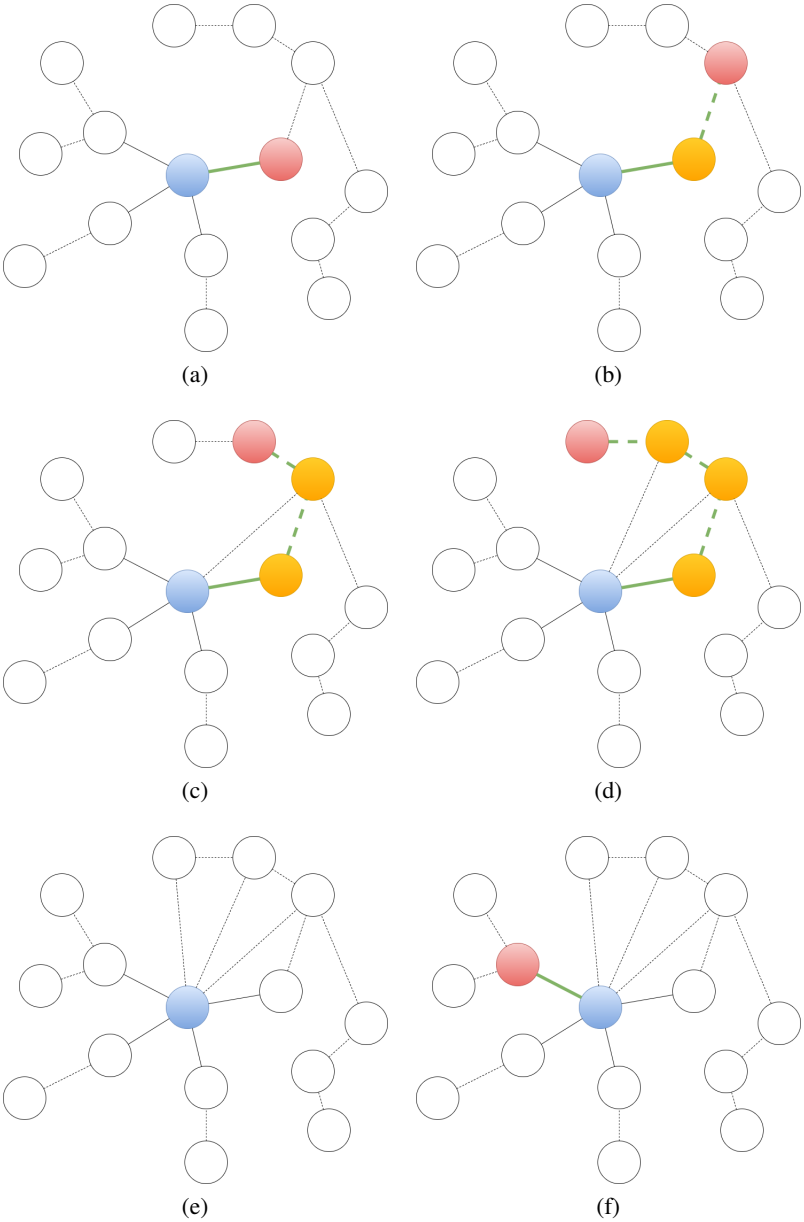

Figure 4.2: four steps and teleport home then start a new random path

Chapter 5

Validation

5.1 Validation List

In Section 4.1 in the previous chapter, we have some requirements for our Walker. Some of the requirements will be immediately fulfilled when the Walker is implemented, but the others need to be validated via experiments. The requirements need to be validated are:

- **Requirement 1**: Being able to discover all peers in the network after sufficient steps
- **Requirement 2**: Being able to prevent load balancing problem.
- **Requirement 3**: Being able to prevent visiting Sybils in some degree. It might not be able to completely prevent visiting Sybils, but it should be able to reduce the probability it encounters Sybils.

In this chapter, we will validate the three aspects above experimentally. Before describing the details about the experiments setting, we briefly describe the philosophy behind each experiment:

For Requirement 1, we put the Waker in a randomly generated network, allowing the Walker freely discover peers in the network, after a certain number of steps, we measure the peer discovery ability of the Walker according to the number of peers it discovered.

For Requirement 2, We put the Walker in a randomly generated network, allowing the Walker freely discover peers in the network, after a certain number of steps, we measure the load balance level according to the distribution of received introduction request from this Walker in all peers in the network.

For Requirement 3, we put the Walker in a randomly generated network where there are both honest peers and evil peers (Sybils), the Walker decide which peers to visit according to a predefined peer discovery strategy. After a certain number of steps, we measure the performance basing on the number of honest peers and evil peers discovered.

5.2 Simulated Network

All of the validation experiments will be conducted in simulated networks. Compared with a real network, a simulated network is easier to monitor and less resource intense. Therefore, by using a simulated network, it will be easier to conduct experiments with a large network. The simulated network consists of two parts – a simulated Network Endpoint and a set of Simulated Peers. A Simulated Peer is a basic unit in the simulated network, it emulates the activity of a real peer, i.e. sending introduction-request, introduction-response etc. Simulated Peers will not actively send Messages, they only silently wait for incoming Messages and respond to them like a real peer. However, the communication between Simulated Peers is not based on Dispersy Message, so, when a Simulated Peer need to communicate with a real Walker, it needs the Simulated Endpoint to convert the data to the format which can be recognized by a real Walker.

In all validation experiments, there is only one real Walker which is fully functional. As a real Walker, it will send introduction-request to other peers periodically to explore the simulated network. Because of the existence of the Simulated Endpoint, the Walker is not aware that the peers it discovers are not real peers. The Walker believes it is walking in a real Dispersy network.

The architecture of the simulated network is shown in Figure 5.1

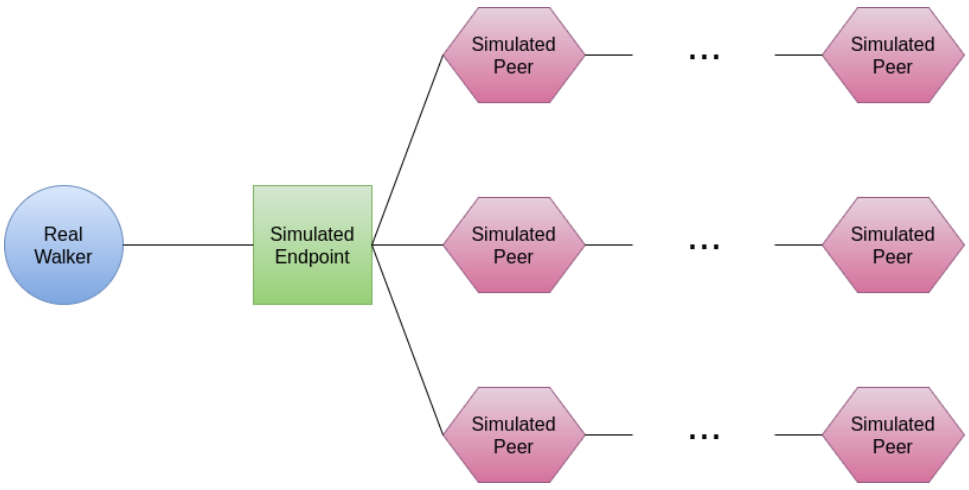

Figure 5.1: Sybil Prevention Validation

The source code of the simulated netowkr can be found in the Github repository:

`https://github.com/YourDaddyIsHere/Local_Virtual_Walker_Network_without_port_limit`

5.3 Coverage Validation

A basic requirement for a Walker is being able to explore the whole network after taking enough steps, in other words, the Walker should be able to cover the whole network. However, in the previous chapter, the two walking strategies favor a subset of peers (trusted peers) over other peers, that brings in a possibility that the Walkers will be limited in a small area around trusted peers, being "pinned" in a small region hence will not explore other peers outside the region. Here we need to assume that, in the network, there is not a "isolated subgraph" where there is no node in this subgraph has at least an edge between itself and a node outside the subgraph. Because if there is an isolated subgraph, no Walker can explore the whole network.

In the case there is no isolated subgraph, by enforcing a finite lifespan for all peers, we eliminate the possibility to be "pinned" in theory, because of the finite lifespan, no peer can limit the Walker forever. But we still need to validate the theory experimentally. In this section, we will use experiment to validate it.

The experiment is conducted in a simulated network. Our goal is to test the coverage ability of the Walker. We believe that if the Walker can completely cover a small network, it will also be able to cover a larger network. We can briefly prove this as following: if there the Walker cannot explore the whole network with infinite steps, it must be "trapped" in a subgraph of the network and the Walker can explore every node in the subgraph otherwise we can find another smaller subgraph which can meet this requirement. To ensure that the Walker cannot escape this subgraph even with infinite steps, this subgraph must be an isolated subgraph otherwise there must be at least one node which has at least one edge between itself and a node outside the subgraph. When the Walker visits such node, it will have a non zero chance to be introduced with a node outside the network hence has a non zero chance to escape the subgraph. However, we assume that there is no isolated subgraph in the network, hence the case above will not happen, so that the with infinite steps, the Walker will not be trapped in a subgraph. So, if we can experimentally prove that the Walker can explore a small network with infinite steps, we can prove that the Walker can explore a larger network.

Given the deduction above, we do not need a very big network to test the coverage of the Walker. The larger the network, the more steps we need to cover the whole network, hence we need more time for the experiment. To restrict the time consumption in a reasonable scale, we choose to use a relatively small network where there are only 2500 simulated peers in it. The real Walker will take 50 thousand steps in the network, the number of discovered peers over time will be recorded, the result for the Random Walker is shown in Figure 5.2

Figure 5.2 shows the result for the Random Walker, which is the current Dispersy Walker. In Figure 5.2, as time goes by, the curve converge to 2500, that means with enough time, the Random Walker will be able to visit all peers in the network. That is a desirable property for a Walker because the basic requirement for a Walker is being able to cover the whole network. In fact, discovering the last few peers is

time-consuming, for a network with 2500 peers, we might need to take hundreds of thousand steps to ensure exploring the last peer, so, in the experiment, we only require walkers to explore 95% of the peers within 50 thousand steps. The coverage validation for all types of Walkers in shown in Figure 5.3

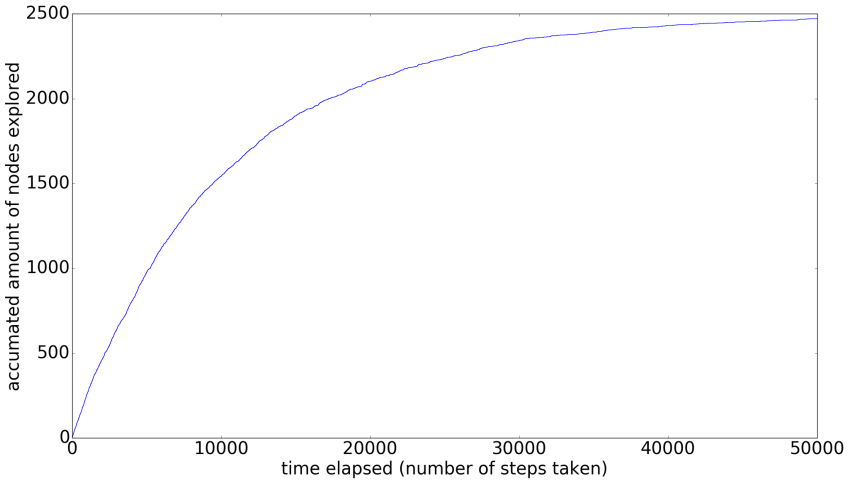

Figure 5.2: Random Walker Peer Discovery

In Figure 5.3, the x-axis represents the steps a Walker already took, and the y-axis represents the number of peers the Walker already visited. For every type of Walker, we repeat the experiment 5 times with 5 different random seed, for each node (a node is a (time, number of discovered peers) pair), we actually have five (time, number of discovered peers) pair, and we only choose the median of the number of discovered peers and demonstrate the corresponding pair in Figure 5.3 as a node.

The four curves in the graphs converge to 2500 after some steps, recall that there are 2500 peers in total, reaching 2500 means explores all peers. As the number of explored peers increases, the chance to encounter a unseen peer becomes more and more slim, as we mentioned, exploring the last few peers needs some luck, so we do not require Walkers to explore the last few peers, we require them to explore 95% of peers, in this sense, In this sense, all Walkers meet the requirement. But they differ in the discovering speed: the Teleport Walker with 0.2 probability is the fastest, the Teleport Walker with 0.5 probability is the second, the Bias Random Walker is the slowest one. Compared with Random Walker, the Bias Random Walker needs obviously more steps to discover the same number of peers.

The reason of the slowing down of discovery speed may be “revisiting discovered peers”: for discovery efficiency consideration only, A Walker better not visits the peers which are already visited recently, we can define the term “recently” as “within 60 seconds”, notice that 60 seconds is the lifespan of a normal peer. So, for discovery efficiency consideration, we should prevent revisiting (visit-

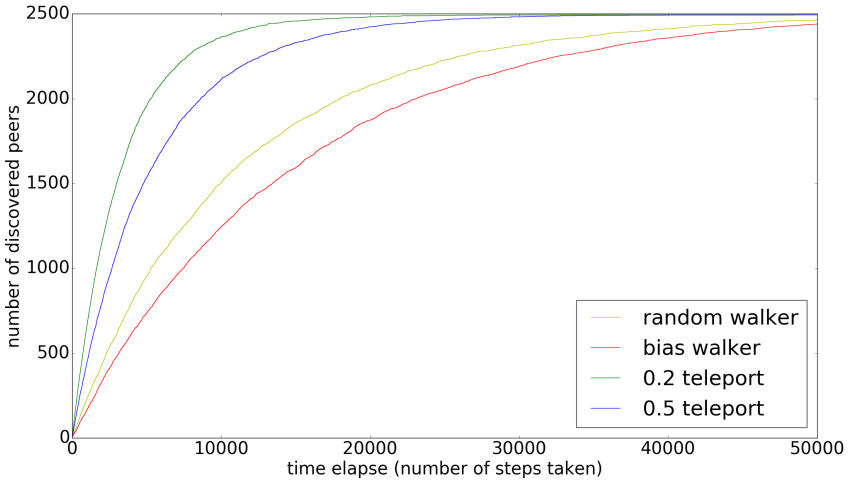

Figure 5.3: coverage of 4 types of Walker

ing a peer twice within 60 seconds). However, discovery efficiency is not the only thing we need to concern, hence revisiting cannot be completely prevented. For Random Walker and Bias Random Walker, they are likely to visit a peer which is recently visited, especially for Bias Random Walker which has a larger probability to visit trusted peers than others. For Teleport Home Walkers, they are less likely to revisit peers. To illustrate this, please image a Teleport Home Walker which has 0 probability to teleport Home, it starts from a random peer and goes along a random path in the network, it will never revisit any peers unless the random path forms a loop and the loop intersect at a recently visited peer. Such probability is slim. Because the lifespan of a peer is only 60 seconds and a Walker takes a step every 5 seconds, a peer has only 12 "recently visited peers". To form a loop and intersect with recently visited peers, the latest visited peer must introduce one of the 12 recently visited peers. If there are 2500 peers in the network, the probability is $12 \div 2500 = 0.0048$. On the other hand, a Bias Random Walker has around 0.5 chance to visit a visited trust peers, 0.0048 is much smaller.

Our two Teleport Home Walkers – 0.2 Teleport Home Walker and 0.5 Teleport Home Walker – because of their nonzero probability to teleport home, they may have a larger probability (larger than 0.0048) to encounter revisiting, but they should still suffer revisiting problem less than Random Walker and Bias Random Walker. In fact, in the experiment, for all peers, we store the time stamp that it is visited by our Walkers and calculate the interval between every two visiting to count the happening times of revisiting. The result is shown in Table 5.1

As Table 5.1 shows Walkers all have revisiting problems in different levels, but the result supports our theory: Bias Random Walker suffer revisiting problem most, then Random Walker, then 0.5 Teleport Home Walker, then 0.2 Teleport Home Walker. Notice that the revisiting count shown in Table 5.1 is also the median for

the results of 5 different experiments.

Walker	revisiting count
Random Walker	18882
0.2 teleport Walker	7211
0.5 teleport Walker	18160
Bias Random Walker	36086

Table 5.1: median of revisiting count for different walker

As we can see, Bias Random Walker has the largest revisiting count, and 0.2 teleport Walker has the smallest revisiting count, combine Table 5.1 and Figure 5.3, the larger the revisiting count, the slower the number of discovered peers converges to 2500. Hence the difference in the convergence speed can be explained by revisiting count.

Though the Bias Walker has a slower convergence speed compared with Random Walker, hence it needs about 30% more steps to achieve the same discovery goal, it is still acceptable.

5.4 Load Balance Validation

The Random Bias Walking strategy and the Teleport Walking strategy both have the danger to visit trusted peers too frequently and cause load balancing issue. A trusted peer have a much longer lifespan than a normal peer, but the lifespan is still finite, that will mitigate the load balancing problem in some degree. But we still need to measure the load imbalance level experimentally.

The experiment setting in Load Balance Validation is completely the same with the experiment in Coverage Validation – 2500 peers in the network, four types of Walkers are tested and every Walker takes 50 thousand steps. We record the number of received introduction-request of every simulated peer and draw the curve to show the distribution of the number of the received introduction-request. The result is shown in Figure 5.4 and Figure 5.5

In Figure 5.4, we show the distribution of received introduction-request of every peer. In this experiment, we use Random Walker. The peers are rank according to the received introduction-request count, that is to say, the peer whose node ID is 0 receives least introduction-request and the greater the node ID, the more introduction-request received. The Random Walker is the current Dispersy Walker, so we can treat this result as the baseline. The load balance data of other peers are shown in Figure 5.5.

In Figure 5.5, the x-axis represents the Node ID, the y-axis represents the number of introduction-request received by this peer. Every type of Walker takes 50 thousand steps and the network has 2500 simulated peers. So a peer receives 20 introduction-request on average. We can measure the severity of load balance problem by a simple metric "balance ratio"– the number of received introduction of the

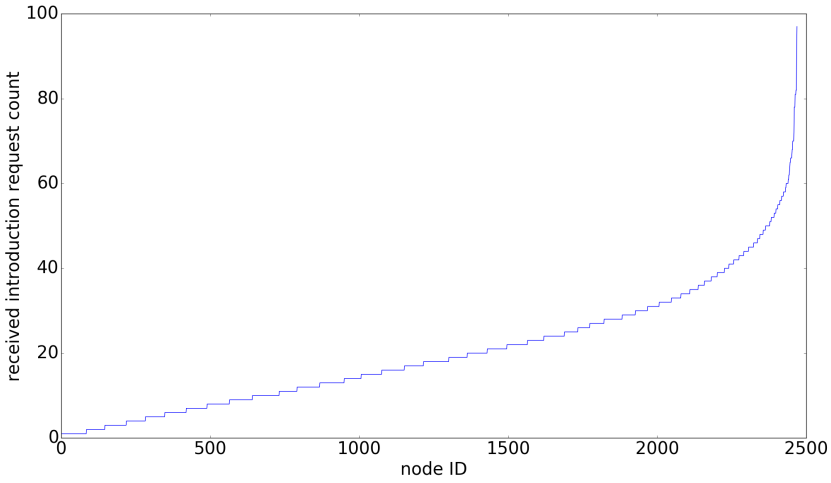

Figure 5.4: Random Walker Load Balance

most overwhelmed peer, divided by the average number of received introduction for all peers. The larger the balance ratio, the more serious the load balance problem

For the Bias Walker, the most overwhelmed node receives around 100 introduction-request, that is 5 times the average level, so the balance ratio is 5. We take it as a baseline. For Teleport Walker with 0.2 probability, the top 1 node receive 64 introduction-request, that is 3.2 times the average level. For Teleport Walker with 0.5 probability, it is around 5 times the average. For Bias Random Walker, that is 5.5 times the average, that is the worst among the 4 Walker.

To better measure the severity of the load balance problem, for every Walker, we run the experiment 5 times (just like what we do in the previous section), and take the median of the results and show it in Table 5.2. Notice that for all experiments, we take 50 thousand steps and we have 2500 peers in the network, so the average number of received introduction request is 20.

Walker	top 1 peer visited count (median)	balance ratio
Random Walker	97	4.85
0.2 teleport Walker	49	2.45
0.5 teleport Walker	87	4.35
Bias Random Walker	101	5.05

Table 5.2: number (median) of received introduction-request for the top 1 peer

As shown in Table 5.2, Bias Random Walker has worst load balance problem, but it is not serious – it is 5.05, only slightly larger than the value of Random Walker (4.85). Since the Random Walker is the current Dispersy Walker, and it does not

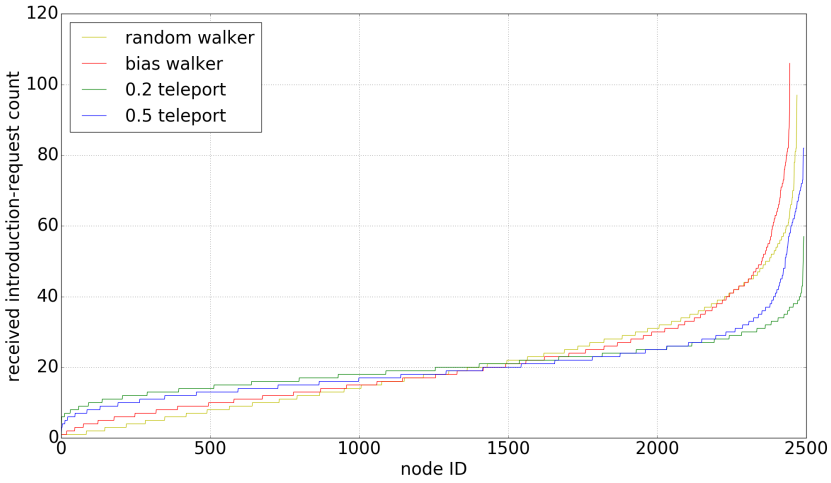

Figure 5.5: Random Walker Load Balance

encounter serious load balance problem, we can take it as the baseline. Therefore, the severity of Bias Random Walker is only slightly greater than the baseline hence it meets our design requirement – does not have serious load balance problem. Given the Bias Random Walker meets the requirement, the rest of the Walker also meets the requirement.

5.5 Sybil Prevention Validation

The validation experiment of Sybil prevention performance is also conducted in a simulated network. For this experiment, we use a much larger network to emulate the situation Walker in the real network. The real network is large and the Walker will never be able to exhaust the whole network.

The network has 1 million simulated peers and a single real Walker. The network contains two regions: a single honest region with 300 thousand honest simulated peers and a Sybil region with 700 thousand simulated peers. A peer has 20 edges with other peers in the same region. The two regions are connected with some attack edges, the number of attack edges varies over different scenarios. The performance of the Walkers in all those scenarios will be recorded and compared.

The real Walker will start with no knowledge about the network except the address of a tracker. The real Walker start with 0 Trustchain Block, it will collect Blocks while it is visiting other peers. There are still four types of Walker to be tested in the experiment: a Random Walker, a Teleport Home Walker with 0.2 probability to teleport home in each step. A Teleport Home Walker with 0.5 probability, and the Walker with Bias Random Walking strategy. The Random Walker will serve as a baseline, the strategies of the rest three Walkers have been described

in the previous Chapter. For every Walker, it will conduct Peer Discovery task using 10 thousand steps, we will record the number of honest peers and evil peers it discovers and then calculates the evil ratio, where

$$evil\_ratio = number\_of\_evil\_peers \div number\_of\_honest\_peers$$

The result of the experiment is shown in Figure 5.6 to Figure 5.8.

Figure 5.6 tries to give you an impression about what is happening. It shows the number of discovered honest peers and evil peers over time, in the case that there are 200 thousand attack edges between the honest region and the evil region. The x-axis is the time elapsed and the y-axis is the number of peers. As time goes by, the number of discovered honest peers and discovers evil peers (sybils) both increase, for a certain moment, we can calculate the evil ratio. The evil ratio overtime is shown in Figure 5.7

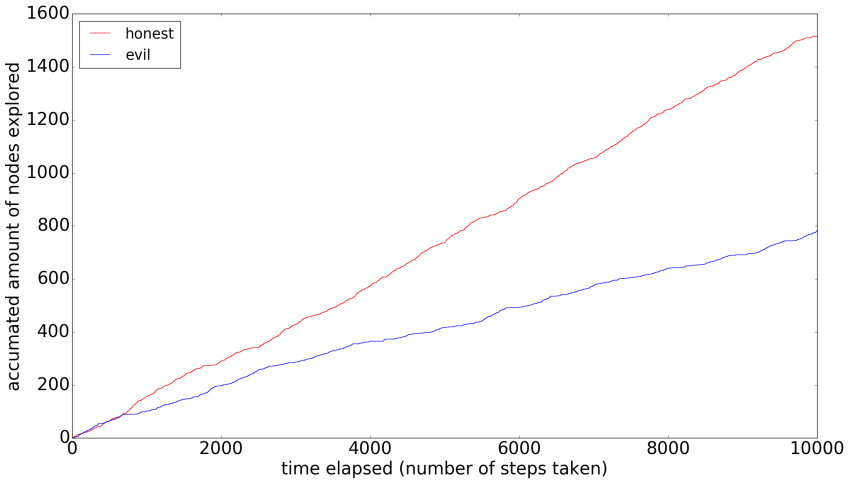

Figure 5.6: Discovered honest peers and evil peers over time

Figure 5.7 depicts the evil ratio over time in the same experiment with Figure 5.6. The x-axis is the time elapse and the y-axis is the evil ratio. The evil ratio is 0 at the beginning and then fluctuate fiercely; at last, as the time goes by, it finally becomes stable around 0.6.

Given a certain moment, by comparing the evil ratios of different Walkers, we can measure their ability to combat against sybil attacks. However, our intuition tells us when the number of attack edges changes, the stable evil ratio will also change. The larger the number, the greater the attacking strength hence the greater the evil ratio. So, to measure the performance of different Walker with different attack strength, we conduct the experiment in the same network with different numbers of attack edges, from 50 thousand to 400 thousand. The result is shown in Figure 5.8

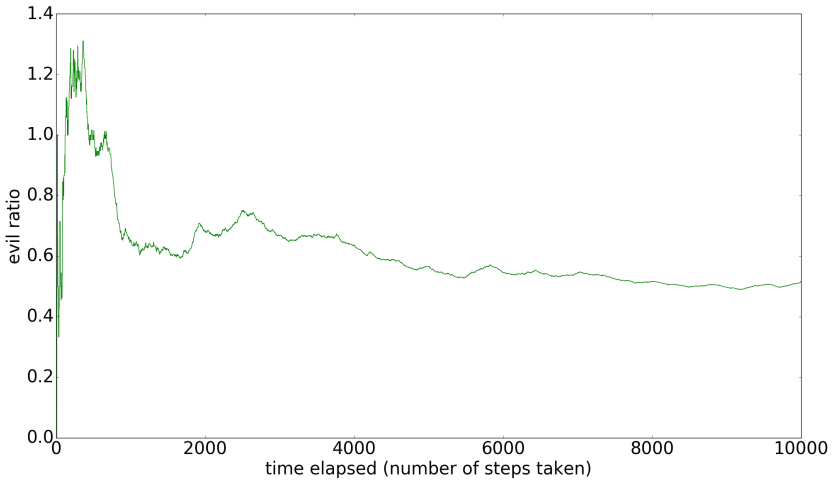

Figure 5.7: The evil ratio peers over time

Similar to Figure 5.3, for every Walker, we run 5 experiments with 5 different random seeds and take the median of the results to show in the figure.

According to the result, Bias Random Walker is the best one on Sybil Prevention aspect, the Teleport Walker with 0.5 probability is the second, the Teleport Walker with 0.2 probability is even worse than the Random Walker. The result is an evidence showing the effectiveness of the Bias Walking strategy and Teleport Strategy (with probability 0.5).

5.6 Discussion

The performance of Bias Random Walker on Sybil Prevention aspect is only slightly better than Teleport Walker with 0.5 probability in many cases (e.g. in the case there are 200 thousand, 250 thousand and 400 thousand attack edges), but it suffers more serious load balance problem than Teleport Walker with 0.5 probability. In addition, compared with other Walkers, Bias Random Walker has worst performance in Coverage Validation. So, there is not a "best Walker". Because our research goal is to implement and test the peer discovery strategy based on Trust, we do not have to figure out which is the best Walker. Since the Bias Random Walker and the Teleport Home Walker with probability 0.5 both use Trust based peer discovery strategy and compared with current Dispersy Walker, they both demonstrate greater resilience to Sybil Attack, we can say our research goal is achieved.

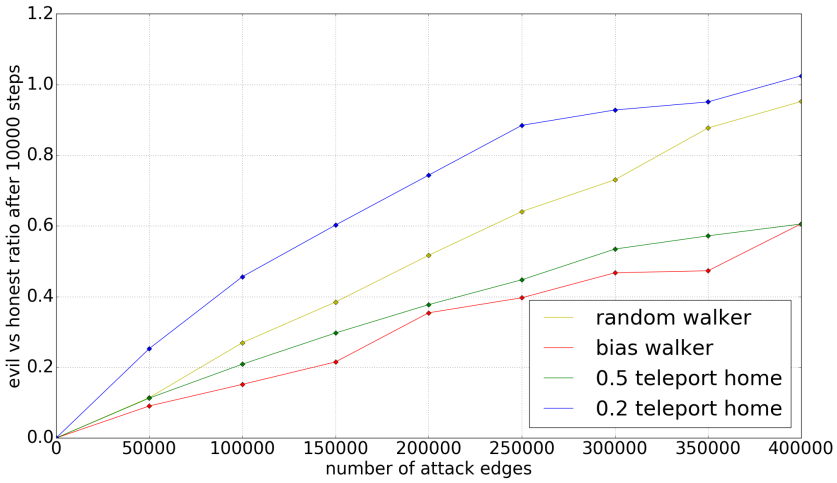

Figure 5.8: The evil ratio of different walker in the network with different attack strength

5.7 Unsolved Problem

According to the experiment results, our new Walker has demonstrated its ability to counter Sybil Attack, or more specifically, the Misleading Attack mentioned in Chapter 2. Recall that we mentioned another attack in Chapter 2 – the DDoS attack. We assert that such attack cannot be completely defended, but it might not be economical for the attacker: it may inflict more resources loss for the attacker rather than the victim.

It is true in the context of current Dispersy Walker (Random Walker): DDoS using introduction-request message in Dispersy will cost the attackers more resources than the victim. We can use an experiment to validate this assert:

For fairness, we should provide two machines with same amount resources for the attacker and the victim, one for each. To achieve this goal, we use Amazon AWS t2 large instances. The resources details for t2 large instances can be found in amazon website [27].We provide the attacker and the victim with two t2 large instances, one for each. The victim runs one Walker instance on one t2 large instance. The attacker runs the attacking scripts in another t2 large instance, the scripts will create 1000 introduction request per second and send to the victim's machine. We record the CPU usage for both attacker and victim every second for the whole duration of the experiment. Notice that because the attacker and the victim have exactly same amount of all kinds of resources, we can compare the CPU usage in percentage rather than the absolute value. The result is shown in Figure 5.9.

In the Figure 5.9 x-axis is the time elapse and the y-axis is the CPU usage in percentage. The attack begins around time = 40. We can observe the increase of the

CPU usage on both attacker and victim. However, the CPU usage of the attacker is much greater than that of the victim. That means, for wasting 1 unit of the resource of the victim, the attacker needs at least 3 unit of resources. For creating a introduction request, the attacker need to generate a large random number as an unique identifier but for the victim, it just need to copy that identifier. That may be the reason for the difference of the workload between creating introduction-request and introduction-response.

Though for an attacker who has sufficient resource, such attack is also feasible it is not economical for the attacker, that is the reason we think this attack is less attractive to the attacker, compared with Misleading Attack.

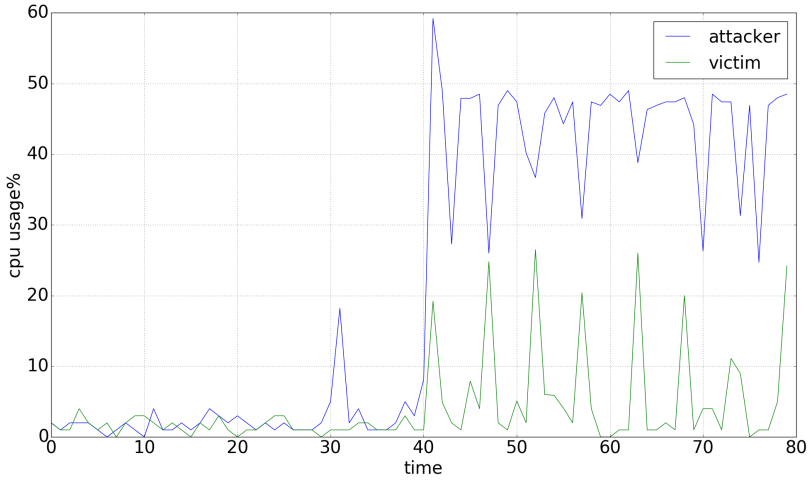

Figure 5.9: CPU usage: victim vs attacker

But after we introduce crawl-request message, the situation is now completely different. Using crawl-request message in DDoS is very economical for the attacker. For the default setting of Dispersy, when receiving a single crawl-request message, the peer should return up to 100 crawl-response message, each contains one blocks. One might easily come up with the TCP SYN flood attack [28] where the TCP SYN message enforce more burden on the server side than client side, hence by flooding TCP SYN to the server, one client can waste a lot of resources of the server at the cost of minor loss on itself. The similar situation happens in our new Walker now. To mitigate the crawl-response flooding, we can reduce the number of crawl-response to reply. However, even we reduce the number to 1, the resources cost in the responder is still far greater than the requester. We use the same experiment setting in introduction-request flooding attack, the result is shown in Figure 5.10:

As Figure 5.10 shows, the attack begins at the moment around time = 30, the attacker easily wastes around 40 units of the resource of victim at the cost of around 1 unit on itself. The root of such imbalance is that: the attacker can store crawl-

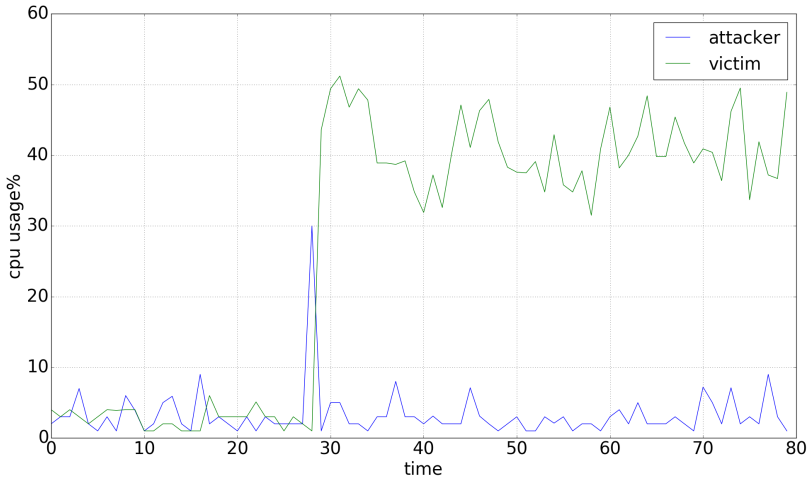

Figure 5.10: victim vs attacker CPU usage

request as binary string format in the memory and reuse it whenever it needs, but the victim cannot reuse the crawl-response because the crawl-response messages contain different blocks every time.The victim can, of course, create one crawl-response for every block and store all of those crawl-responses in its memory, but given the number of blocks in the database, this method will then cause very high memory usage, if the victim does so, it is actually attacking itself.

On the other hand, reducing the number of crawl-response to reply has a side effect: it will slow down the blocks collection process for honest requesters, and also impose a heavy burden on their bandwidth and CPU resources.

We do not have an idea to defend the crawl-request flooding attack for now.

Chapter 6

Conclusions and Future Work

In previous chapters, we discuss the peer discovery problems we encounter, the strategy to solve it, the implementation of the strategy in new Dispersy Walkers and the validations of such Walkers. In this chapter, we will have conclusions for the previous chapter and discuss the limitation of the methods we use in the previous chapter and the future work.

6.1 Conclusions

In this thesis, our goals include

- Improve the current Dispersy Walker, adding Trustchain Blocks collection ability to it.
- Improve the current Dispersy Walker, enabling it to calculate the trust value for other peers according to the historical behaviors of other peers.
- Improve the current Dispersy Walker, implementing a trust-based peer discovery strategy to it, the peer discovery strategy should be able to reduce the probability of the Walker to visit sybils

To achieve the first goal, we employed the Trustchain system which can record all historical behaviors in the form of Blocks. To solve the problem that peers are not aware of the global history of the network, the new Walker will keep collecting Blocks from any peers it visits.

For the second goal, we create a graph basing on the interaction among peers, where the nodes represent peers and edge represent upload or download interactions among peers. We define two trust level – “trusted” and “untrusted”.

For the third goal, we believe that a peer with high reputation is less likely to be a Sybil. In our new Walker,”high-reputation peers” refers to ”trusted” peers. Following this belief, we design two walking strategies. The first one is Bias Random Walking strategy, where the Walker randomly decide which peer to visit in next step, but instead of assigning equal probability to all peers, it assigns a higher

probability to trusted peers. The second one is Teleport Walking strategy, where the Walker is deterministic to choose the newly introduced peer as the next peer to visit, but with a certain probability α, the Walker will teleport to the starting point, and randomly choose a trusted peer to start a new exploration path. If there is not a trusted peer, the Walker will choose a random peer.

We validate our new Walker experimentally to test their performance. The first thing to be validated is the exploration ability. A good walking strategy should allow the Walker to explore all peers in the whole network. According to the result of this experiment, all tested walking strategies are able to explore the whole network while they differ in exploration speed. Bias Random Walking strategy has the slowest speed, Teleport Walking strategy demonstrate the highest speed with a low α, but in the case where α is relatively high, the exploration speed slows down.

The second to be validated is the load balance. Walkers are not required to distribute their introduction-request to all other peers equally, but they should prevent concentrate their introduction-request on a small portion of peers hence produce serious load balance problem. The experiment shows load balance problem in all walking strategies, but the imbalance levels are acceptable.

The third thing to validate is their abilities to prevent visiting sybil peers. According to the result, the Bias Random Walking strategy and Teleport Walking strategy both prove their effectiveness, while Teleport Walker only shows superiority over the current Dispersy Walker in the case α is relatively high (0.5).

So, we can now draw the conclusion: the experiments in the previous chapter shows that the two walking strategies are both effective against Sybil Attack. They also meet the coverage and load balance requirements. Therefore, our goal to implement and test the usefulness of trust based peer discovery strategy is achieved.

6.2 Limitation

In our design, the Walker supports k hop trust, k can be any value. However, we cannot determine the optimal k analytically. In our experiment, we set $k = 2$, but there may be other k which can lead to better performance. To be even worse, there may not be a global optimal k, it is likely that optimal k is sensitive to the average degree of the nodes in the graph. For example, for a graph that the average degree of nodes is small, $k = 20$ may be the optimal value, but for a "small world" graph [29], $k = 20$ will be too large. Therefore, given a specific graph, it is necessary to fully investigate the graph before setting k, that is a major limitation

All experiments in this thesis are conducted in simulated networks. The simulated networks are design to be close to the real network, but they cannot 100% emulate the real network because we do not have a thorough investigation on the real Dispersy network, we have no idea on:

- The number of Blocks a peer has on average in a real network.
- The number of peers a single peer knows in a real network.

- The number of Sybils and high-reputation Sybils in the real network.
- The number of attack edges in a real network.

In addition, we do not consider the online and offline issue in the simulation. All peers are always online in the whole duration of the experiment. However, in real life, the users of Dispersy will continuously go online or offline. Peers going online or offline will change the topology of the network, going offline will possibly make some Blocks unavailable hence hinder the calculating the reputation of Peers.

Furthermore, in the simulated network, we do not take the NAT into account (but we still enforce finite lifespan to the simulated peers), we assume that the Walker can contact any other peers after NAT puncturing. However, in a real network, the NATs of two peers may both adopt very strict policies hence they cannot contact each other even after the standard NAT puncturing.

6.3 Future Work

Basing on the Unsolved problem mentioned in Chapter 5 and the Limitation mention in this chapter, there are some works for the future:

We should fully investigate the real Dispersy network, we can invite users to voluntarily report the number of Blocks they have over time and the number of peers they know over time. We can also use this method to investigate the average length of online duration. With such knowledge, we can make our simulated network better emulate the real network.

We should improve the crawl-request and crawl-response message, make the burden of requester and responder more balance. We might be able to achieve this goal by employing proof of work mechanism. We can require the requester to put a random number or timestamp in crawl-request, hence the crawl-request messages are different every time hence cannot be reused. But the feasibility and the usefulness of such approaches need to be investigated in future works.

Bibliography

[1] A. S. Tanenbaum and M. Van Steen, *Distributed systems: Principles and paradigms*. Prentice-Hall, 2007.

[2] R. Schollmeier, 'A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications', in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, IEEE, 2001, pp. 101–102.

[3] ERNESTO. (2015). Bittorrent still dominates internet upstream traffic, [Online]. Available: `https://torrentfreak.com/bittorrent-still-dominates-internets-upstream-traffic-151208/`.

[4] B. Cohen. (2017). Bitorrent protocol specification, [Online]. Available: `http://www.bittorrent.org/beps/bep_0003.html`.

[5] D. Evans, 'The internet of things: How the next evolution of the internet is changing everything', *CISCO white paper*, vol. 1, no. 2011, pp. 1–11, 2011.

[6] Wikipedia. (2017). List of network protocols (osi model) — wikipedia, the free encyclopedia. [Online; accessed 24-September-2017], [Online]. Available: `https://en.wikipedia.org/w/index.php?title=List_of_network_protocols_(OSI_model)&oldid=794023617`.

[7] P. Maymounkov and D. Mazieres, 'Kademlia: A peer-to-peer information system based on the xor metric', in *International Workshop on Peer-to-Peer Systems*, Springer, 2002, pp. 53–65.

[8] G. Fodor, E. Dahlman, G. Mildh, S. Parkvall, N. Reider, G. Miklós and Z. Turányi, 'Design aspects of network assisted device-to-device communications', *IEEE Communications Magazine*, vol. 50, no. 3, 2012.

[9] D. Clip, *Gnutella protocol specification v0. 4*, 2007.

[10] M. Ripeanu, 'Peer-to-peer architecture case study: Gnutella network', in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, IEEE, 2001, pp. 99–100.

[11] S. Helal, N. Desai, V. Verma and C. Lee, 'Konark-a service discovery and delivery protocol for ad-hoc networks', in *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, IEEE, vol. 3, 2003, pp. 2107–2113.

[12] L. Wang and J. Kangasharju, 'Real-world sybil attacks in bittorrent mainline dht', in *Global Communications Conference (GLOBECOM), 2012 IEEE*, IEEE, 2012, pp. 826–832.

[13] Wikipedia. (Sep. 2017). Denial of service attack, [Online]. Available: https://en.wikipedia.org/wiki/Denial-of-service_attack.

[14] J. Mirkovic and P. Reiher, 'A taxonomy of ddos attack and ddos defense mechanisms', *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[15] D. Senie and P. Ferguson, 'Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing', *Network*, 1998.

[16] J. R. Douceur, 'The sybil attack', in *International Workshop on Peer-to-Peer Systems*, Springer, 2002, pp. 251–260.

[17] Wikipedia. (Aug. 2017). Donald trump twitter following might include more than 4 million bots, [Online]. Available: https://en.wikipedia.org/wiki/SYN_flood.

[18] L. Lamport, 'Time, clocks, and the ordering of events in a distributed system', *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[19] M. Meulpolder, J. A. Pouwelse, D. H. Epema and H. J. Sips, 'Bartercast: A practical approach to prevent lazy freeriding in p2p networks', in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, IEEE, 2009, pp. 1–8.

[20] D. N. Steffan, 'Multichain: A cybercurrency for cooperation', 2015.

[21] S. Nakamoto, 'Bitcoin: A peer-to-peer electronic cash system', 2008.

[22] wikipedia. (2015). Scalability-bitcoin, [Online]. Available: https://en.bitcoin.it/wiki/Scalability.

[23] V. inc. (2014). Visa inc. at a glance, [Online]. Available: https://usa.visa.com/dam/VCOM/download/corporate/media/visa-fact-sheet-Jun2015.pdf.

[24] P. Veldhuisen, 'Leveraging blockchains to establish cooperation', MSc thesis, Delft University of Technology, May 2017.

[25] P. Otte, 'Sybil-resistant trust mechanisms in distributed systems', MSc thesis, Delft University of Technology, Dec. 2016.

[26] H. Yu, M. Kaminsky, P. B. Gibbons and A. Flaxman, 'Sybilguard: Defending against sybil attacks via social networks', in *ACM SIGCOMM Computer Communication Review*, ACM, vol. 36, 2006, pp. 267–278.

[27] Amazon. (Sep. 2017). Amazon ec2 instance types, [Online]. Available: https://aws.amazon.com/ec2/instance-types/.

[28] Wikipedia. (2017). Syn flood, [Online]. Available: https://en.wikipedia.org/wiki/SYN_flood.

[29] S. Milgram, 'The small-world problem', *Psychology Today*, vol. 1, no. 1, 1967.