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

3rd October 2017

Author
Changliang Luo

Title
Peer Discovery With Transitive Trust in Distributed System

MSc presentation
TODO GRADUATION DATE

Graduation Committee
TODO GRADUATION COMMITTEE Delft University of Technology

Abstract

It is a common case that resources belong to different people and people distribute across the world. So, for making good use of resources, people have to cooperate with each other. Cooperation is import not only in physical world but also in cyber space. Computers holding different resources need to cooperate with others. The problems that people cooperate with people, 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 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 historic behaviors of other peers. We believe by using the established trust, a Peer to Peer system will be more resilient to Sybil Attack.Our trust system will be implemented and tested in the peer discovery system of Tribler, which is a distributed system helping people sharing 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.But the core of the onion never change. Now, I know what is the core – cooperation. In stone age, hunters cooperate to hunter 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 trust 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 am also appreciate the help from Martijn de Vos and Quinten Stokkink, they kindly help me warm up with Tribler project. I also grateful to Kelong Cong, he offers invaluable experience and advice on my thesis.

Changliang Luo

Delft, The Netherlands
3rd October 2017

Contents

Chapter 1

Introduction

The topic of this thesis might sounds 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 "peer discovery"

Image that you live in 1970s and you are seeking for a old friend that you have not seen for 10 years. You have no idea what job he is working for, which country he stays, what city he lives in, what telephone number he is using, and there is no Facebook or Twitter at 1970s. The only thing you know is he was your classmate in B university 10 years ago.How do 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 centimeter thick), it is better to call it "address tower". We all know it is not realistic.

The more realistic way is that you visit the B university, visiting people and requesting about 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 get to the introduced people, request information from them and move on. By doing this, you get closer and closer to your old friend, and finally "discover" him in a corner of this vast world. And there is another story, you accidentally visit some liars, they lie to you about the contact information of you 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 system, the task to find another peer is called "peer discovery".Just like the two story above, in peer discovery, you need to make decision 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 is any system follow the definition provided by [1]:

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

In this sense,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 he/her "as a single coherent system".

Distributed system can be further categorized as different sub class according to their architecture, a "traditional" one is Client/Server Architecture, a network consisting of multiple client and a web server is a typical type of Client/Server architecture. To be more accurate, Client/Server architecture can be defined as following[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 emphasize 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.

Besides the Server/Client architecture, there is a more "modern" type of distributed system – 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/Service architecture and Peer to Peer architecture: the peers can share part of there resources, so the amount of resources among peers does not necessarily exist;all peers play same role in Peer to Peer architecture, unlike Server/Client architecture where Server and Client 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.Which is a super popular file sharing application released in 2001. By the time of 2015, it still dominated the upstream traffic of Internet – has a upstream share of 28.56%[3]

BitTorrent is not a specific software, it is a protocol described in [4].Softwares that support the BitTorrent protocol can join in the network and share or download files from other users.The downloading and uploading happen between clients without passing through intermediary entity. That raise a problem – how to locate the clients you want to contact. There are billions of machines running on the Internet, only part of them are running bitTorrent clients, and only part of the BitTorrent clients have the file you want to download, hence the first step to initiate a download or upload 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 face this problem, in fact, for all Peer to Peer system, the foundation of cooperation among peers are peer discovery. For all those systems, peer discovery can be described as "given some clues,find out the address of other peers", but since different Peer to Peer systems vary on many aspect, they may choose different peer discovery approaches. To better explain the idea of different peer discovery approaches, we will describe three typical applications which use different peer discovery approaches – ARP request in Ethernet,old fashion BitTorrent, Mainline Distributed Hash Table (MLDHT) based BitTorrent.

First, APR request. As we know, Internet is consist of different layer of protocols,according to OSI model[5],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, 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 a IP address 35.150.1.2 and a MAC address 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 need 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 known 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 a ARP request to all machines in this Ethernet saying "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 do they have IP address 35.150.1.2, if not, ignore it. So, everyone in the network will ignore this ARP request, except Machine B.Once receive this APR request, B will reply a 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.

Second, old fashion BitTorrent client.Though BitTorrent is a typical Peer to Peer system, in its old version,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 a same file is grouped together, and such group is called "swarm". For each swarm, there is a file 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 swarm. To start downloading a file, one needs to join the swarm; to join the swarm, one needs a torrent file. 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.

Third, Main Line DHT based BitTorrent client.For legal consideration, and also other concerns like robustness, BitTorrent start to use 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[6].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 a ID, which is a 160 bits hash, to it;this hash uniquely identifies the peer. Kademlia Protocol introduce a concept of "distance" between two ID, hence we can calculate the distance between every two peers. For a peer, it only stores a (IP address,UDP port,Peer ID) for peers close to it. The concept "close to it" means the distance is less than a certain value d, d can be specified in 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.

In the three example above, we see three different strategies for peer discovery, ARP request uses broadcast strategy, old version BitTorrent uses central service,MLDHT BitTorrent uses a introduction based peer discovery strategy. Though all these strategies work well in their case, we can still see the pros and cons for them:

Broadcast strategies 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^2 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 a Ethernet is usually small,broadcast will not cause serious traffic jam, but for applications with a large network, broadcast is not a ideal strategy

Introducing a centralized entity will easily solve the peer discovery problem, but will introduce new problem: the centralized entity will be the performance bottleneck, it is always easier to double the number of Bittorrent clients in the swarm than double the resources in the tracker. Besides the performance issue, a centralized tracker will also perform as the single point of failure, once it is taken down by the attackers, the whole swarm will break down.

The strategy of DHT Bittorent client is a desirable one. Idea that peers rely on other peers to discover new peers ,rather than relying on a centralized tracker, shows significant potential.There is not a single point of failure, and 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, previous research[7] demonstrates feasible attacks on Main Line DHT Bittorrent Clients. 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.

Fortuantely, Tribler Project provides a chance to investigate similar peer discovery strategies. Dispersy is a module of Tribler which is responsible for peer discovery, packets handling and persistent storage, the peer discovery functionality concentrates in a module of Dispersy, named Walker. Walker conducts peer discovery rely 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 similar to [7]. 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 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,investigate the probability to defend attackers in the network where the majority of the peers are controlled by attackers.

This thesis is organized as follow: Chapter 2 will describe the problem of peer discovery in a high level way, investigate the problem we are facing with a abstract model. In Chapter 3, the high level problem in previous chapter will be specified.Details of Tribler and Dispersy will be introduced, and the relevant 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 2

Problem Description

Modern technology like 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. In other words, it allows cooperation among strangers. However, the foundation for cooperation is trust among people but the stranger have not trust among each other; Working in an environment without trust is dangerous, attacks may happen in anytime anywhere.But establishing trust among strangers is hard, especially in the large scale cooperation in the Internet.

The most easy way to establish trust is introducing a central party that 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 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, but 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 defend attacks using the established trust.

In this chapter, we will give the formal definition of peer discovery, 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 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 other people, who might know the address, to you. 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 edges (i, j) indicates 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.*

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 j will then reply address of node j or other relevant nodes;knowing a new node will result in adding a new edges 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 a (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 responsibility to help other peers, that is to say,as a peer, when other peers ask for help, you have to help them–either gives 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 obligate to respond to any introduction request, the attacker can attack the peer by sending 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,so that they can make the resources of the target machine unavailable to legitimate users[8].For DDoS attack, there are already many researches, Mirkovic create a taxonomy for DDoS attack and defense [9]. For example,a peer as a victim can protect himself by filtering strategy [10], where it can partially abandon its introduction responsibility to some other peers. But there is no strategy which can perfectly protect this peer from such attack unless it completely abandon his introduction responsibility, that will greatly undermine the cooperation protocol.However,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 machines to send messages to the victim;but mobilizing is never free.Sometimes the DDoS attacker have to pay more than the victims,that will limit the usefulness of DDoS attack.In later chapter, we will show the fact that in Tribler arsenal, introduction request based DDoS attack inflicting more resources lost 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 a irrelevant peers to them. A peer may spend a few minutes or even hours to make a progress in discover the target peer, but an attacker can ruin all those effort by introducing it with a far away peer. The attacker can even introduce the honest peer with one of its accomplice, 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 few seconds 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 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 cyber space, the difficulty can be overcome. In physical world, the interaction between you and others are done through your body, you body is actually an agent of your mind. In this world, everyone has only one body.The mind and body has a one to one mapping. However, in cyber space, that is not the case. People in cyber space interact with others via identities like Facebook account, email account or forum account 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 nature person,organization or anything living stuff who has a mind.
- **identity**: an agent of entity, which is used for interaction.

Though people in cyber space can in theory have multiple identities in the same domain like twitter, many systems still have a 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 “sybils”. The term “sybil” is introduced by Douceur in 2002[11].A famous example for sybil is Trumph’s bot event [12]. People find out that one of the trumph’s fan account is actually a bot controlled by some organization, to be even worse, people doubt that there are more bots in Trumph's fans account. That is to say, some organizations create some fake identity and use them for certain goals.They may want to create a illusion that Trumph'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 cyber space. 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 its identity, because it is hard to reveal the entity behind this identity, we cannot tell the difference between a normal identity and a sybil. Therefore, attackers can create a lot of sybils to create a a illusions that there are a lot of normal identities 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 a sybil controlled by the attacker, the probability is high enough to make Misleading Attack feasible.Remember that the resource needed for creating sybils in cyber space is usually few, that makes the cost of Misleading Attack acceptable.

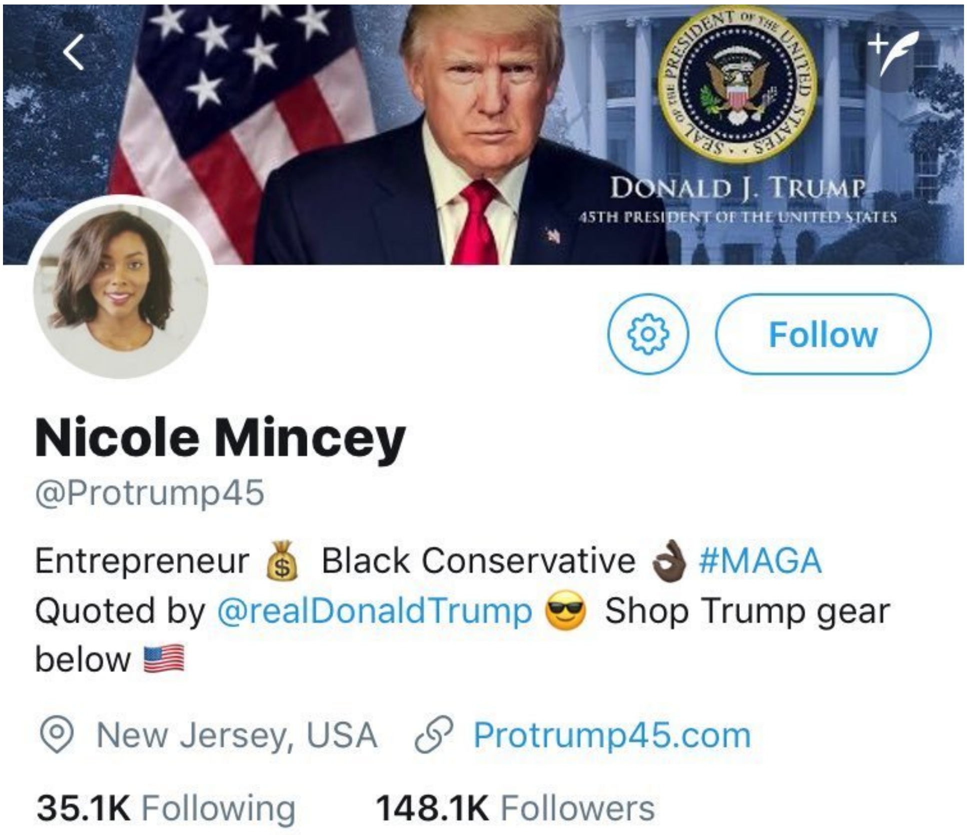
Compared the cost and gain of attacks in the two scenarios, we believe the misleading attack in second scenario is more critical.So, this thesis will focus on developing a defense strategy against such attack. Now we formally define Misleading Attack

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.*

2.3 Defense mechanism

In our daily life, people already develop defense mechanism to misleading attack, people will not buy a flight ticket to the other side of this planet basing on the introduction from a stranger; but if the introduction comes from your parents or very 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 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 trust has been proved to be successful, with the trust among people, we can bravely embrace various kinds of cooperation with significant 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 cyber space. Now we define the trust and the trust based peer discovery strategy in our model:

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



Nicole Mincey
@Protrump45

Entrepreneur 💰 Black Conservative 👌🏿 #MAGA
Quoted by @realDonaldTrump 😎 Shop Trump gear below 🇺🇸

New Jersey, USA Protrump45.com

35.1K Following 148.1K Followers

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

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 Strategy basing on trust in combating the misleading attack with sybils.

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

Fortunately, 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 for the misleading attack in Definition 3. But for now, Dispersy Walker does not have a book keeping system for historic behaviors of other peers.So there is not a Trust calculation system 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 book keeping 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 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 system. It provides functionality 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 system can put attention on designing the protocols in high level without worrying about details in lower level.

By installing Dispersy on all machines in a distributed system, developer can enforce protocols on all those machines. Dispersy provides support on many aspect of distributed system development. However, many of its functionalities are irrelevant to this article. In this chapter, we will only introduce the relevant functionalities.

We have introduce some terminologies in previous chapter,but in Dispersy context, they have more specific meanings.In 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. Dispersy supports multiple kinds of Authentication, but only two kinds are relevant with this article – MemberAuthentication and NoAuthentication. In MemberAuthentication, the Authentication sections contains the identity of the Message sender. If a Message adopt 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 article, 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[13]

- Resolution" In this article 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 network, it contains its own Message 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 instance of a specific Community type share a 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 its own Community instance 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 type 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 modules, it is a set of functions scattering in a Community class, hence it is hard to maintain and further develop. Tribler Team has a plan to take out the Walker and form an independent module. It provides a good chance to adding new features to the Walker.

3.1.3 Walker and Peer Discovery

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

Similar with old-fashion of BitTorrent peer discovery, there are trackers in Tribler network for bootstrapping use. Every peer in the network start with an 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 peers 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, puncture.The details of those Messages are listed below:

- introduction-request: introduction-request Message 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 a introduction-request to peer B.

- introduction-response: introduction-response 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 their messages, instead of 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: dispersy-identity 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 upgrade all Community to version 2, the dispersy-identity Message will be removed.

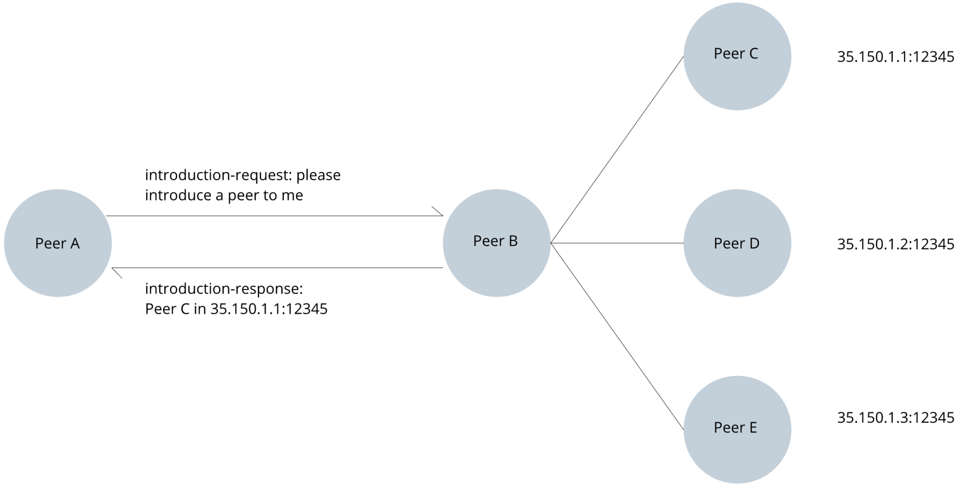

Peer A
Peer B
Peer C
Peer D
Peer E
introduction-request: please introduce a peer to me
introduction-response: Peer C in 35.150.1.1:12345
35.150.1.1:12345
35.150.1.2:12345
35.150.1.3:12345

Figure 3.1: introduction-request and introduction-response

In a 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 address 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 appears to have the same address 35.157.80.100 (but may run in different ports). While NAT offers convenience for sharing public address, it brings in obstacles for communication of Peer to Peer system. There are many types of NAT, in this article, we only discuss the typical NAT that typical users use. A typical 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 in port 12345. peer A knows peer C is at 35.157.80.100:12345, but it cannot contact peer C, because C never send packet 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 come in. To enable such cooperation between A and C, the current Dispersy Walker uses the following two Messages:

- puncture-request: it contains 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: puncture Message 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.

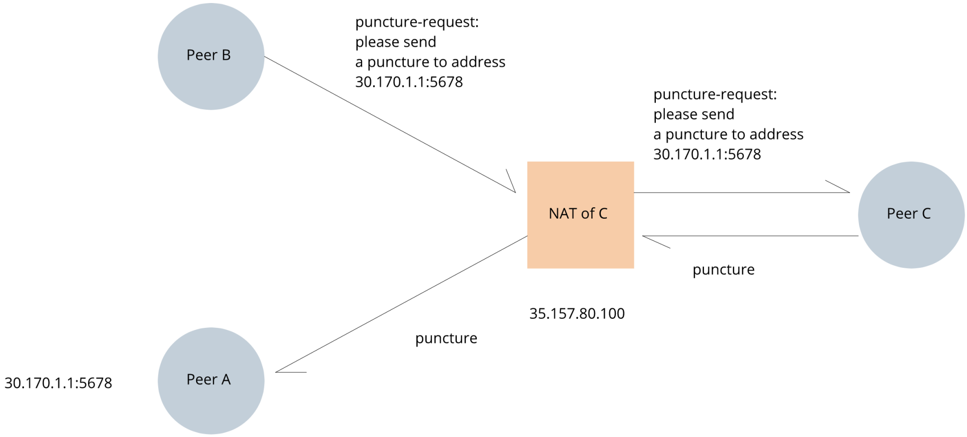


Figure 3.2: NAT puncturing

Now we put the workflow of six mentioned Message together:

- 1.peer A send introduction-request to peer B
- 2.if B knows the public key of peer A, skip to 4,else, B will reply with an 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 Message, 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 adopt very strict policies, the NAT puncturing procedure will not work. But this article 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 life span of hole is determined by the configuration of NAT, but usually less than 60 seconds. Hence the life span of a peer is less than 60 seconds. This is a important nature we can utilize. We will discuss it later.

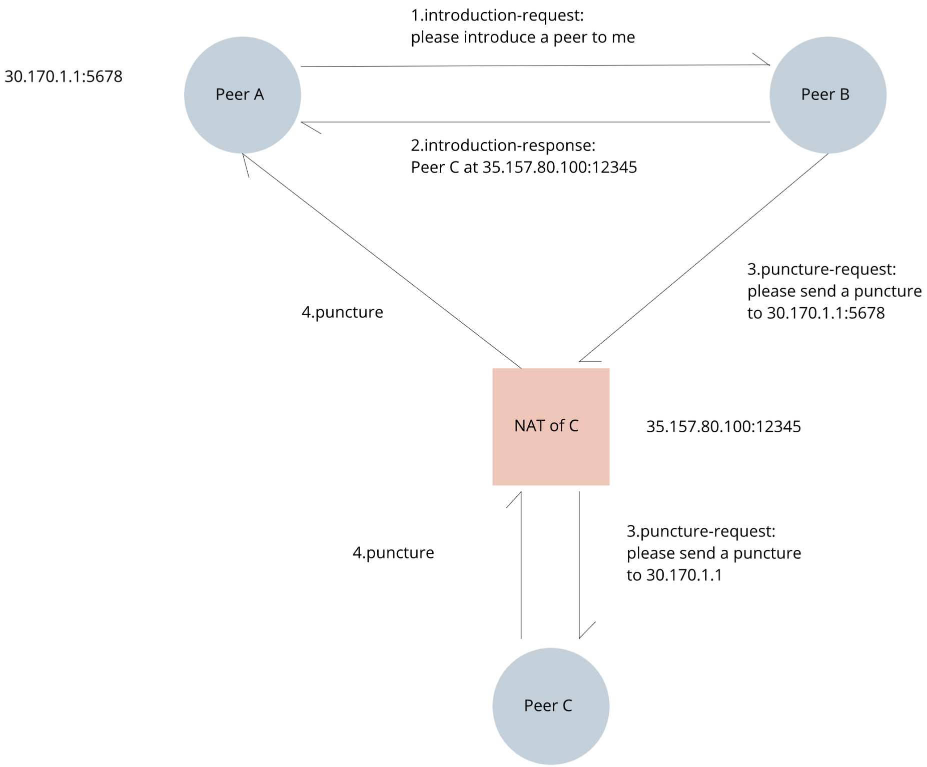


Figure 3.3: the whole work flow

3.2 Storing Historic Behaviors

To implement a more sophisticated Walker to prevent visiting sybils, We need a book keeping system to store historic behaviors associated with every identity. The first attempt for storing historic behaviors in Tribler is BarterCast[14]. In BarterCast, a peer provides voluntary report 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 report,tamper their own history. Hiding or tampering will fundamentally impair the reliability of the book keeping functionality. Addressing this flaw, Tribler team created a new, tamper-proof book keeping system – MultiChain. Multichain was first introduced by Norberhuis in [15],but now its name change to TrustChain,we will use the two terms interchangeably according to the context,but keep in mind they are different names for the same thing. The concept of Multichain is similar to the concept of the famous Blockchain introduced in [16]. Before we discuss Multichain, we should first introduce Blockchain before discussing Multichain.

3.3 Blockchain

Blockchain is a data structure supports the transaction of Bitcoin between peers. 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 a announcement that Alice transfer a certain amount of money to Bob, and prove that you are Alice. In the case of transfering 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 A) 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 a attack named "double spend", Alice can sign another announcement (announcement B) to transfer the Bitcoin, which should be transfer to Bob, to Alice herself instead. Because of the lagging of Internet, it is likely that many peers receive announcement B before announcement A, hence they will validate announcement B and invalidate announcement A. If there are enough peers validate announcement B, the network as a whole will invalidate announcement A, 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 of a way to issue time stamp on the transaction, make sure transaction A has a earlier timestamp than transaction B. 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 solve 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 in to 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 valid Blockchain in the Bitcoin system – the longest one. To prevent peers favoring their own Blockchain so that the consensus can not be reached, the system require peers to consume a lot of computing power in creating a block. Although peers are allow to create blocks, they have the right to choose which transactions to be put in a specific block, they can not validate this block easily. To validate a Block, a peer need to solve a mathematical puzzle which have no other way to solve but randomly guessing number.This mechanism prevent peers' own Blockchain to grow 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 prevent peers to intentionally tamper or hide Blocks. To better illustrate it, we assume there is a Blockchain $Block0$,$Block1...Blockn$. Every Block (except Block0) should include the hash value of the previous Block, for example, $Block3$ should contains the hash value of Block2. The hash value of previous Block serves as a "hook" to all contents of 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 follow 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 keep the all historic behaviors of all peers. Although this nature shows benefits, it also have 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 grow larger; in addition, 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 validate per second. Currently, the Bitcoin system can validate 7 transactions per second [17]while VisaNet have a peak performance of 56 thousand transactions per second [18].If Tribler directly uses Blockchain, such slow transaction processing speed may cause problems. So, Tribler team develop a variant of Blockchain – Multichain

3.4 Multichain

The concept of Multichain is first introduced in [15]. Later, a major upgrade is introduced in [19]. This chapter will introduce the concept of Multichain basing on the work of [19]

Unlike Blockchain, Multichain is not design for currency system, the "transaction" of the Multichain 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 Multichain: a Block in Multichain only describe one interaction between two peers, unlike Blockchain where a Block contains multiple transactions. For better sketching the image of Multichain, we will still use the term "transaction".

Unsurprisingly,Multichain System has multiple chains of Blocks. Unlike Blockchain who need a single chain to reach an global consensus concerning the order of interactions, Multichain system does not require such a strict consensus. Every peer in the Multichain system is responsible to maintain their own chains that store all historic behaviors (transactions) of their own. But one peer can also requesting transactions of other peers but it is obligatory to collect chains from others. In other words, it is unnecessary to requesting whole history of the network before joining in the network.

A Multichain Block contains signature from only one involved peer,however, to validate a interaction record, we need the signature of both involved peers, so Multichain 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 Block chain, all Blocks depicting the historic interaction of a specific peer should be chained together, every Block contains a the hash of previous Block, for the two Blocks involved in same interaction, they contains 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 historic interaction can only be validated with both Blocks presence.Figure 3.1 is the illustration of Multichain used in [19],the columns represent chains of different peers.

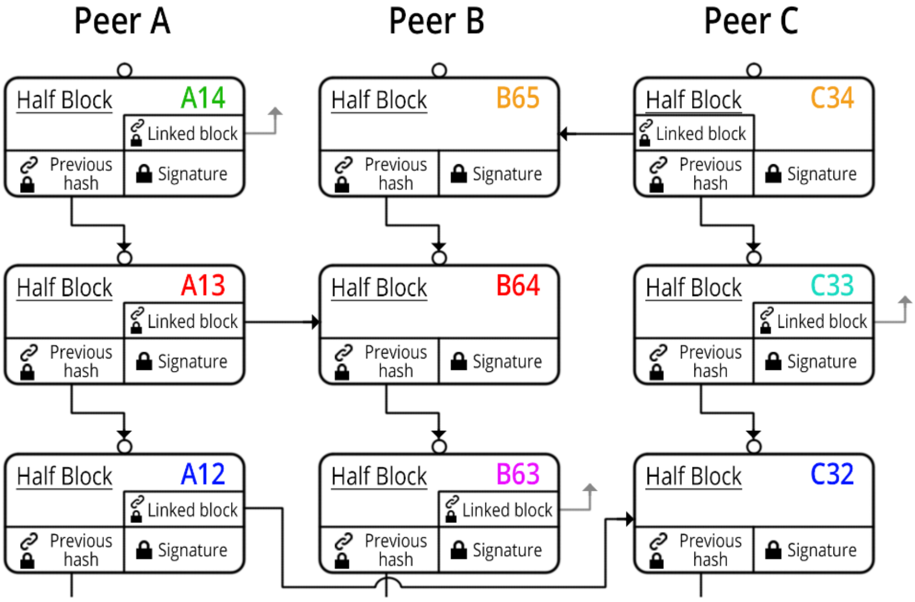


Figure 3.4: Multichain

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: between the two peers, the one who upload 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 basing 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”he
- step 3: the responder will check the receiving Block on uploading and downloading field basing on its perspective of the interaction, if they match the perspective of responder, the responder will creating a new Block containing the same uploading and downloading field, sign it, point the 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.

3.5 Scoring System and Walk strategies

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

3.6 Scoring System

Pim Otte’s research[20] describes two accounting mechanism which can assign scores. The first one is NetFlow. NetFlow is processed basing on a directed graph, where nodes in graph representing peers and edge representing the upload from one peer to another,the weight is the amount of uploading data counting on megabytes.NetFlow calculates the reputation score of a peer B in the perspective of peer A is based on the 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 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

(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 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 previous section, a "hook" is either a "previous hash" field pointing to previous Block of a peer, or the "link sequence number" pointing the other Block in a 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 Block certain Block is static, 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 more faster, but still takes around 10 seconds. 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.

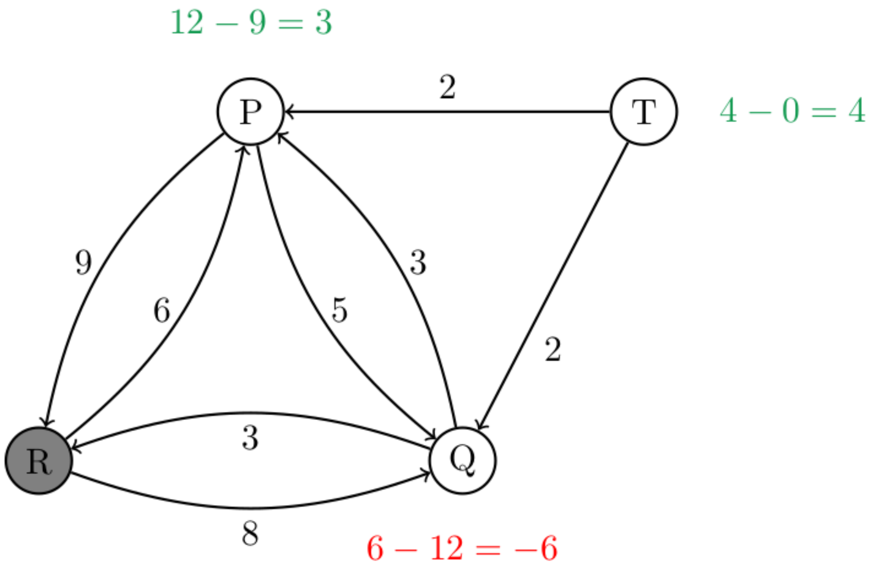


Figure 3.5: NetFlow algorithm

3.7 Walk strategies

[19] test the performance of two walking strategies in his researches, he aims to evaluate the Block exploration ability of the two strategies. Although the goals of our research is seeking for a proper strategies that resilient for Sybil Attack described in Chapter 2, the research of [19] still provides valuable information.

The first walking strategy in [19] is random walking. Every time a Walker needs to choose a peer to visit, it randomly pick out one of its known peers to walk. The Algorithm 1 shows the random walking strategy in pseudo codes.

Algorithm 1: Pim Veldhuisen's Random Walking strategy

```
for each step do
    next_node = pick_random_from(connected_neighbours);
    walk_towards(next_node);
end
def walk_towards(node):
    new_peer = node.request_neighbour();
    connected_neighbours.add(new_peer);
    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, [19] does not mention what algorithm he uses to calculate score. But that is 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 introduce bias in the probability assigned to each peer, instead of assign equal probability to every peer, it assign 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 is depicted in Algorithm 2. Notice that, Algorithm 1 and Algorithm 2 are copied from [19]

Algorithm 2: Pim Veldhuisen's Bias Walking strategy

```
while $uniform_random_variable() > phi$ do
    index = (index + 1) % len(ranked_live_edges);
end
next_node = ranked_connected_neighbours[index];
walk_towards(next_node);
def walk_towards(node):
    new_peer = node.request_neighbour();
    connected_neighbours.add(new_peer);
    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 probability to high score (reputation) 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 – 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 needs 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 limited the portion of high reputation sybils in a very low level – it is likely that less than 1% sybils have high reputation.

Since we have the reason to favor high reputation peers, the performance of focus walking in [19] 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 discriminate the peers according to rank, the difference of 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 life span, 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 group, rather than by single peer. We can put all high reputation peers in a group and low reputation peers to another group. We assign different probability to a group, and all group members share the same probability.

- we enforce finite life span on all peers, but allow high reputation peers to live longer.

Besides load balancing problem, we have another reason to enforce finite life span on peers: with infinite life span, 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 single peer.

3.8 Sybil Attack Defense

Reputation 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[21] identifies the characteristic of the network as follow:

- 1.There are two regions in the network, a 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

- 3.We (the honest users) are born in honest region

- 4.Every node which nodes it is connected to

- 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.

[21] concludes the above characteristic for social network, 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 the number of sybils. Hence the graph can be roughly divided into two regions – honest region and sybil regions. We assume the trackers are not compromised, then the a honest peer should start its peer discovery in a honest region.

The characteristic 5 does not hold, hence 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 a lot of resources. But on the other hand, the time out of peers can be used as a weapon against Sybil Attack.

Recall the Sybil Attack we discussed in Chapter 2. The idea behind this kind of Sybil Attack is injecting the address into the peer lists of honest peers. In the perspective of the whole network, it can also be described as sybil region injects ”toxic” address to honest region. The sybil 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 new attack edges, further increase the injecting velocity of sybil address. If the life span of the peers are infinite, the network will eventually evolve to a full connected network that all peers have edges with all other peers, including sybils. However, the life span of peers are infinite, 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 (as a whole) to counter such Sybil Attack.And a simple way to reduce the probability of visiting sybils are: visiting high reputation peers more frequently, visiting low reputation peers less frequently

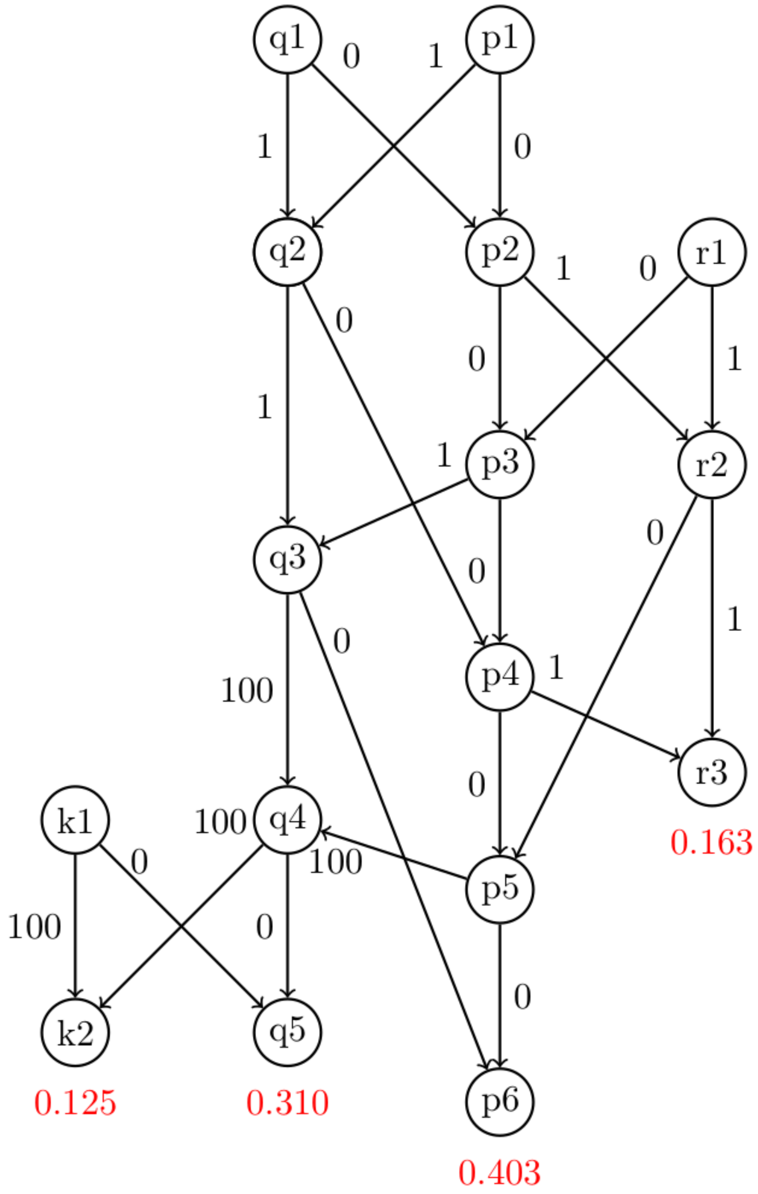


Figure 3.6: NetFlow algorithm

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 requirements.

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 and
- 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 historic behaviors associated with other Walkers. Our Walker choose TrustChain as the book keeping system for historic behaviors, but unlike Block Chain where peers have global consensus of the whole history, Trust Chain does not require peers to be aware of the whole history, hence Walkers are usually aware of their own history. So, before calculating the trust, a Walker must collect Blocks which store the historic behaviors of other peers.

With historic 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 base peer discovery strategy to defend the Walker against Sybil 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 approach to infinite, the Walker should be able to discover every peers in the network instead of a small subset of them. In addition, the Walker should not cause load balancing problems: if all Walkers in the network favors a few high reputation peers, those high reputation peers will receive too many introduction request and end with overloaded. Our new Walker should prevent such situations happen.

4.2 Walker Architecture

Dispersy provides many supports for the current Dispersy Walkers.

- Port Listening: Dispersy listens on a 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 string that can be transfer via Internet.
- Peers Management: The Community instances in Dispersy can store, retrieve, 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 provides the same functionalities mention above, the architecture of the new Walker is shown in Figure 4.1

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

Peer Manager allow 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.

Crawler will automatically collect Blocks from other peers, the received Blocks will be pass to Persistent Storage Manager.

Persistent Storage Manager manage 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.

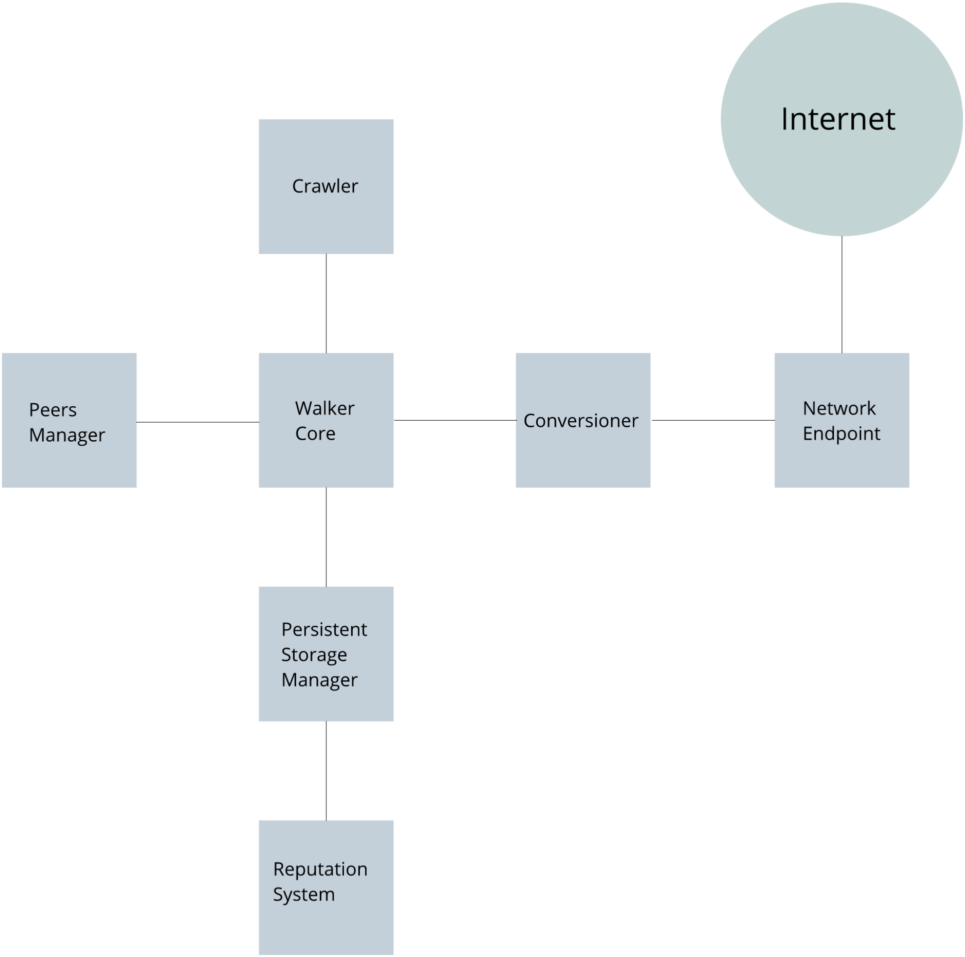


Figure 4.1: Walker Architecture

4.3 Block Collecting

Our reputation system and walking strategy relies on the Multichain Blocks describing the historic interactions of other peers. As we mentioned in previous chapter, Multichain 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 needs 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 peer, there are two Messages involved in this process: crawl-request and crawl-response.

- crawl-request: contains the public key of the creator of Blocks.
- crawl-response: the response of a crawl-request, contains a 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 retrieve Blocks which are created by peer C in its database, if there is at least one such Block, B then send 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 Blocks can be collected at the moment when our Walker needs to evaluate the reputation of certain peer or collected regularly when the Walker is exploring peers. The first approaches may cause a traffic jam in the network because the of the enormous number of relevant Blocks, so, we choose the second approaches.

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 discover 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 [20], 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 an set of edge representing upload interaction between Graph. A edge (i,j) indicates there is at least an 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 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 divide all peers into two class: the peers that we have k-hop transitive trust are put in trusted group, and the other peers are put in 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 have 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 $k = \infty$, 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, therefore, in this article, we take a conservative attitude and choose $k = 2$

4.5 Walking 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 relatively have reputation 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.On the other hand, if the Walker only visit high reputation peers, it is likely to be limited in some small regions consisting of high-reputation peers, leaving the vast area of network unexplored. While such strategy may provide highest level of security, it compromise 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 bias in probability – it does not treat all peers equally in probability, but assign higher probability to trusted peers, where trusted peers are peers that we trust. The bias walking strategy is described in Algorithm 3.

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 to 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.

Algorithm 3: Bias Random Walking

```
for each step do
    #random number between 0 and 1;
    random_number = get_random();
    if $random\_number \geq 0.995$ then
        next_node = a random tracker ;
    else if $random\_number \geq 0.5$ then
        next_node = a trusted peer ;
    else if $random\_number \geq 0.3$ then
        return a outgoing peer ;
    else if $random\_number \geq 0.15$ then
        next_node = a outgoing peer ;
    else
        next_node = an 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();
```

With this strategy, the Walker will walk through a certain 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 peer in a high speed. Busrt once the peer visit a sybil controlled by attackers, the sybil can introduce the Walker with another sybil, as a result, the Walker, we walk deep into the sybil region and has no chance to get back. To avoid that, we require the Walker to ”teleport” 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 peer and take it as the starting point of the new 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 $random\_number \geq \alpha$ 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 “you”, red node represents the peer you are currently
  visiting, yellow peers represent the nodes you visited along this random
  path.In addition, solid lines represent the fact that “you” trust those nodes
  and know their address, and the dashed lines represent the fact that “you”
  know the address of those peers but not trust them.In (e),“you” teleport
  home and in (f) you take the 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();
```

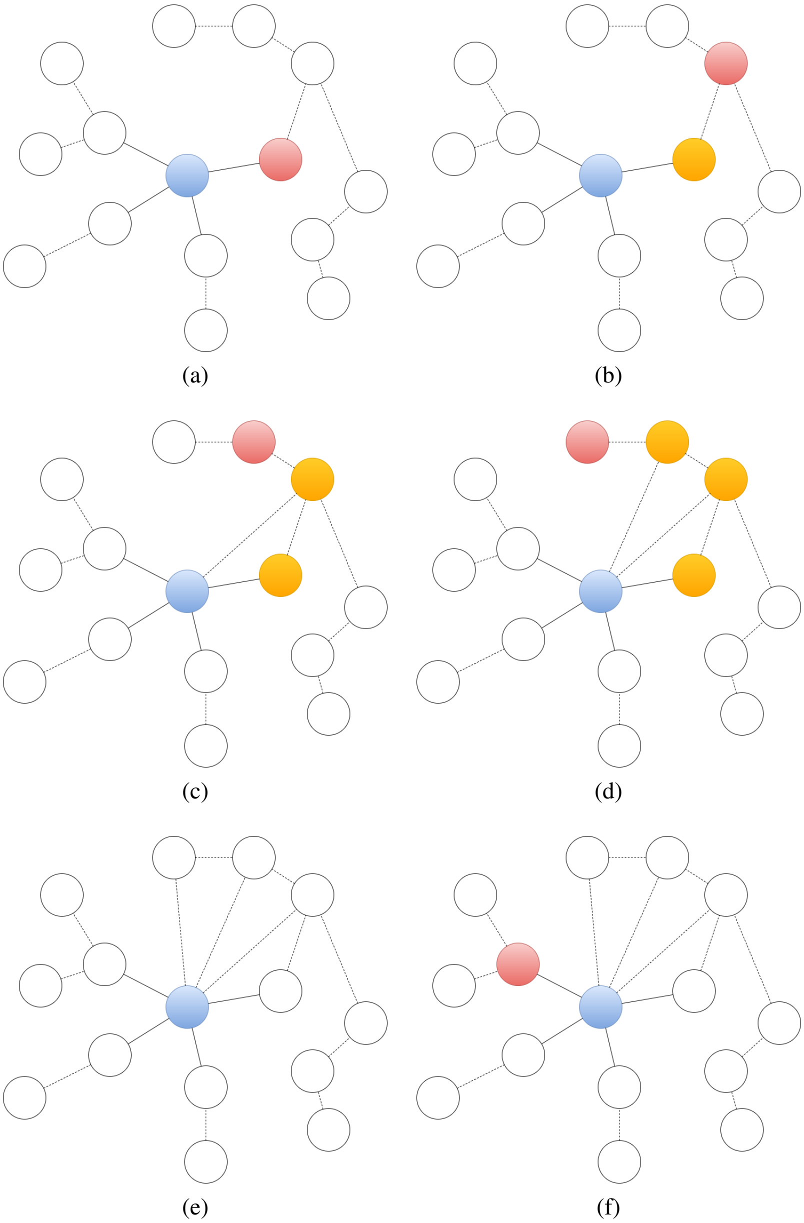

(a) (b) (c) (d) (e) (f)

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 previous chapter, we have some requirements for our Walker, and some of the requirements will 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 enough steps
- **Requirement 2** :Being able to prevent load balancing problem.
- **Requirement 3**: Being able to prevent visiting sybils in some degree.

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

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

For Requirement 3, We put the Walker in a randomly generated network, allowing the Walker freely discover peers in the network, after 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 simulated network, it will be easier to conduct experiments with large network.The simulated network consists of two parts – a simulated Network Endpoint and a set of Simulated network. A Simulated Peer is the 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 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 meets are not real peers. The Walker believes it is exploring a real Dispersy network.

The architecture of the simulated network is shown in Figure 5.1

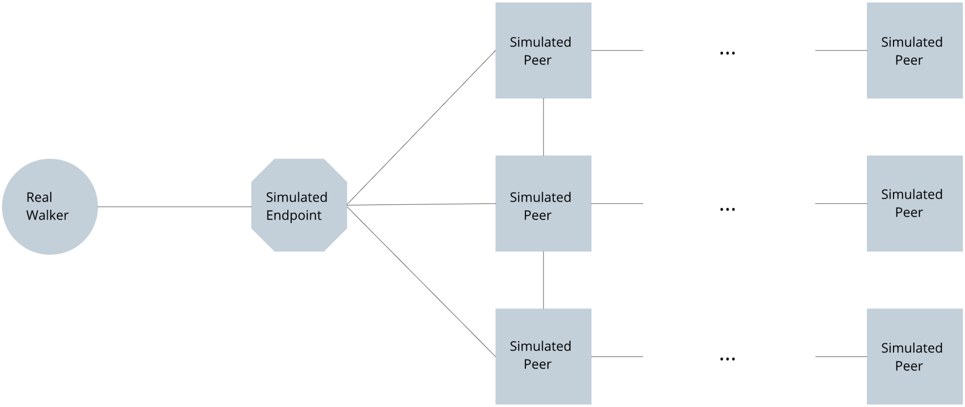


Figure 5.1: Sybil Prevention Validation

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 previous chapter, the two walking strategies favor some kind of peers over other peers, that brings in possibility that the Walkers using those strategies will be limited in the area around a set of peers, being ”pin” in a small region hence will not explore other peers outside the region. By enforcing a finite life span for all peers, we eliminate such possibility in theory, because of the finite life span, no peer can limit the Walker forever. But we should still to validate the theory experimentally.

The experiment is conducted in a simulated network.Because of goal is to test the coverage ability of the Walker, we do not need a very big network. The larger the network, the more steps we need to cover the whole network,since we only want to see the trend, using large network is actually a waste of time. So we use a small network where there are only 2500 simulated peers in it. The real Walker will takes 50 thousand steps in the network, the number of discovered peers over time will be recored, 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, visiting every 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, 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

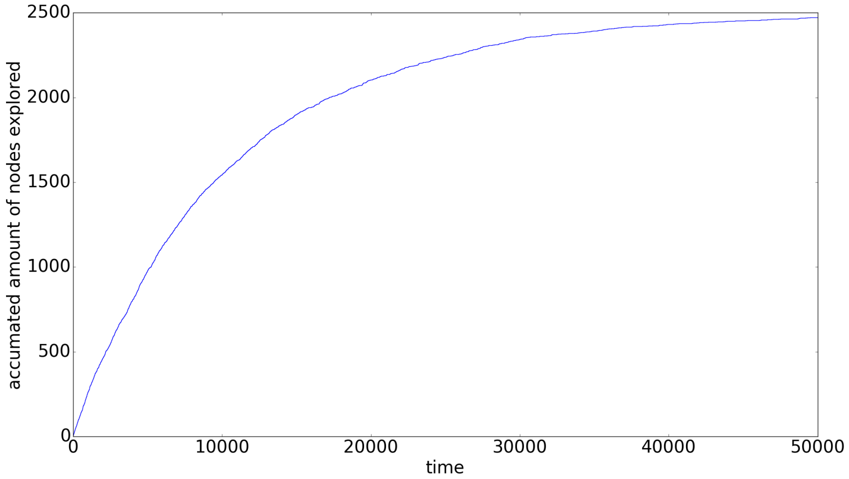


Figure 5.2: Random Walker Peer Discovery

In Figure 5.3, x-axis represents the steps a Walker already take, and the y-axis represents the number of peers the Walker already visited. For all types 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 for (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 curves in the four graphs converge to 2500 after some steps, recall that there

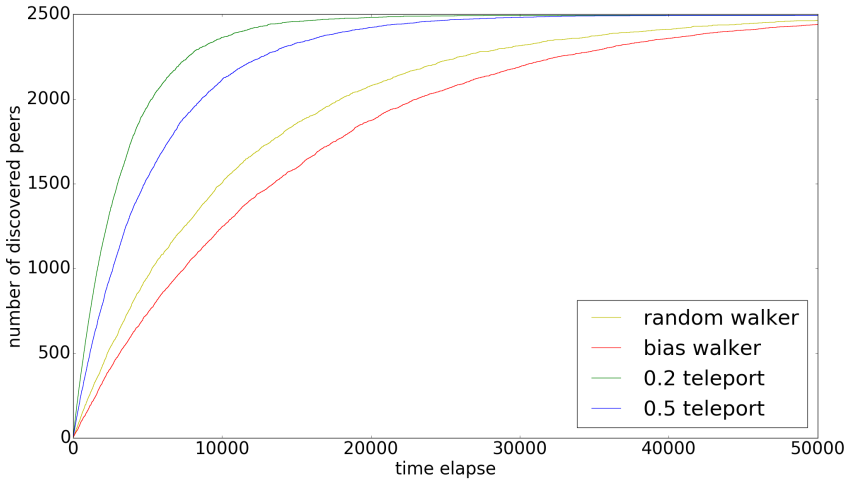


Figure 5.3: coverage of 4 types of Walker

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 differs in the exploring 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 about 30% 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, the 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 are the life span of a normal peer. So,for discovery efficiency consideration, we should prevent revisiting (visiting a peer twice within 60 seconds). However, discovery efficiency is not the only thing we need to concern, and revisiting cannot be completely prevented. For Random Walker and Bias Random Walker, they are also likely to visit a peer which is recently visited, especially for Bias Random Walker which has 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 go along a random path in the network, it will never revisit any peers unless the random path forms a loop, the probability that a random path form a loop is slim. So, in this theory, the smaller the probability to teleport home, the less likely to encounter a revisiting. 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 still have chance to encounter revisiting, but they should 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 level,but the result support 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 among 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 issues. A trusted peer have a much longer life span than a normal peer, but the life span 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 Exploration experiment – 2500 peers in network, four types of Walkers are tested and every Walker takes 50 thousand step. 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 peers. In this experiment we use Random Walker. The peers are rank according to the received introduction-request count, that is to say, 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 base line. The load balance data of other peers are shown in Figure 5.5

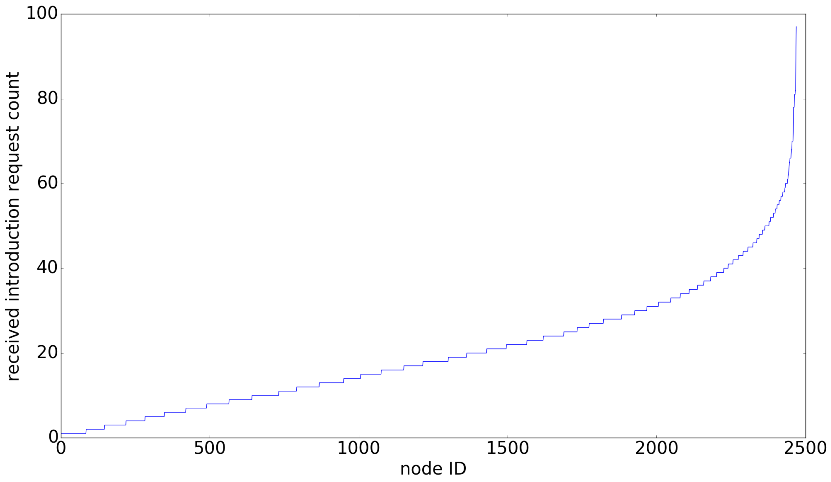


Figure 5.4: Random Walker Load Balance

In Figure 5.5,the x-axis represents the number of introduction-request a simulated peer receives, the y-axis represents the number of peers receiving such amount of introduction-request.Every 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 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. We take it as a base line. 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 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.

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

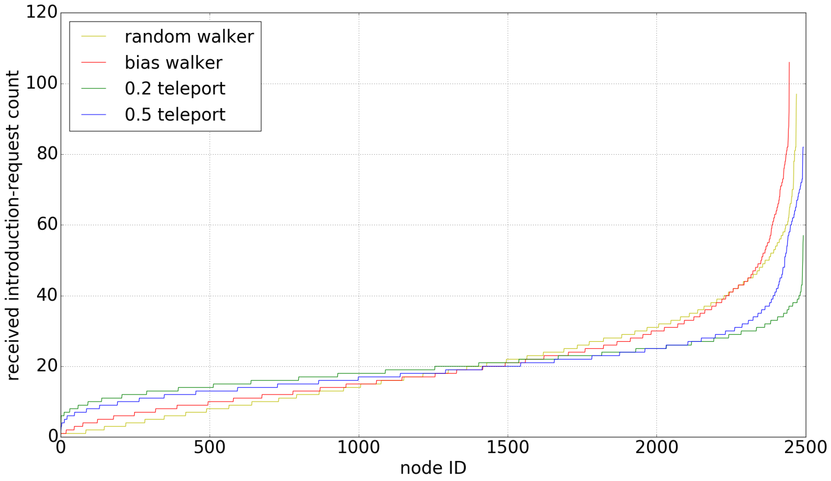


Figure 5.5: Random Walker Load Balance

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

balance problem, we can take it as the base line. Therefore, the severity of Bias Random Walker is only slightly greater than the base line hence it meets our design requirement – do 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 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 region: 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 Multichain Block, it will collect Blocks while it is visiting other peers. There are four types of Walker will be tested in the experiment: a random Walker, a Walker teleport Walker with 0.2 probability in each step, to Teleport back to starting point.a Walker teleport Walker with 0.5 probability in each step, to Teleport back to starting point, and the Walker with bias random walking strategy. The random Walker will serve as a base line, the strategies of the rest three Walkers have been described in 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 might give you a 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 honest region and evil region. The x-axis is the time elapse 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 over time is shown in Figure 5.7

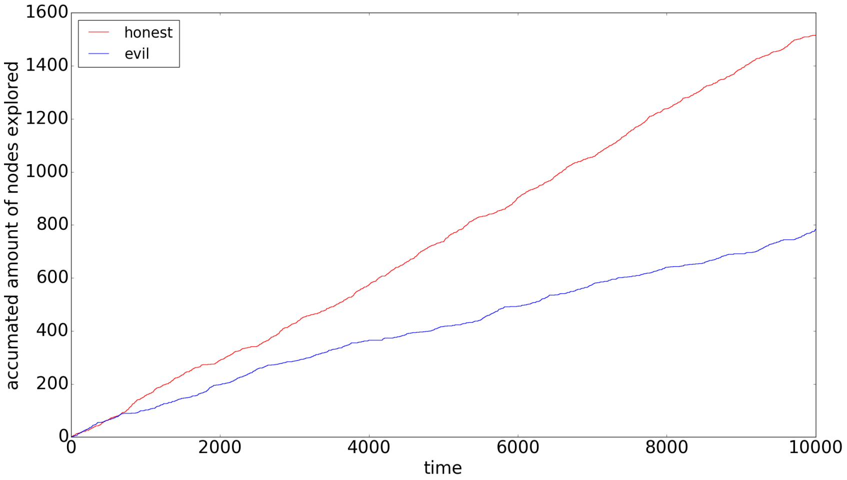


Figure 5.6: discovered honest peers and evil peers over time

Figure 5.7 depict the evil ratio over time in the same experiment with Figure 5.3. 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 become stable around 0.3.

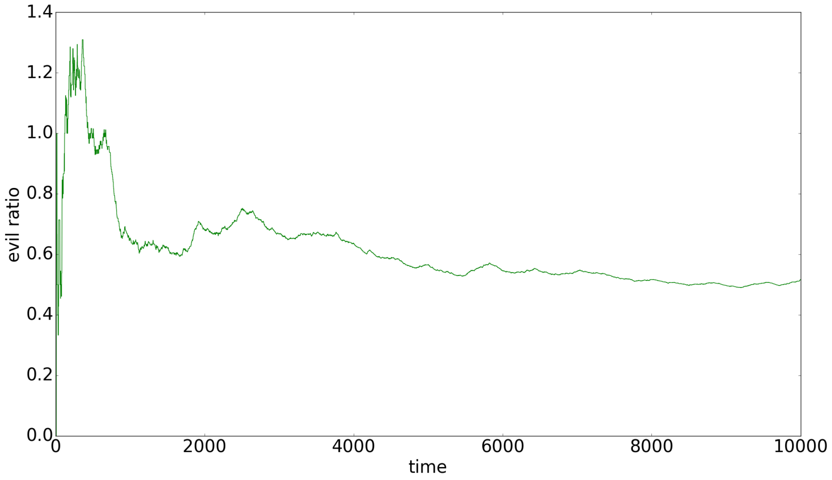


Figure 5.7: evil ratio peers over time

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 changes. The larger the number, the greater the attacking strength. So, to measure the performance of different Walker with different attack strength, we conduct 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

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 suffer 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 implement and test the peer discovery strategy basing on Trust, we do not have to figure out which is the best Walker. Since the Bias Random Walker and the

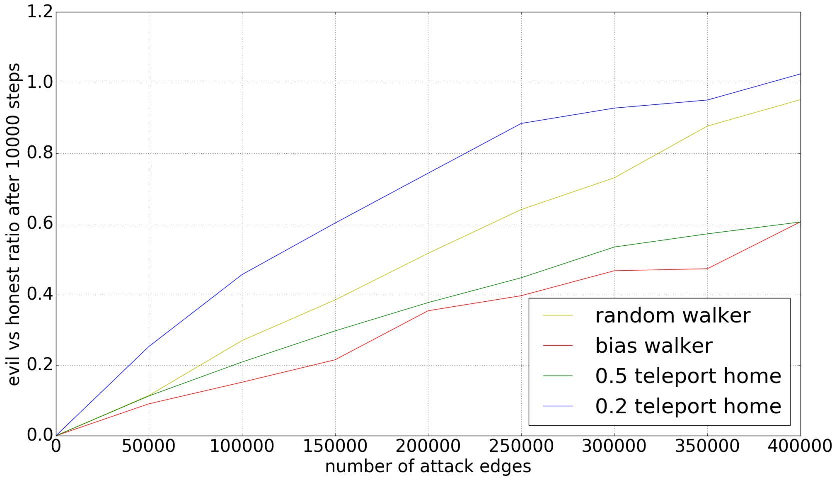


Figure 5.8: evil ratio of different walker in the network with different attack strength

Teleport 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, our research goal is achieved.

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 inflicts more resources loss for the attacker rather than the victim.

It is true in Dispersy context: 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 resources for the attacker and the victim.To achieve this goal, we use Amazon AWS t2 large instances. The resources details for t2 large instances can be found in amazon website [22].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.The result is shown in Figure 6.1.

In the Figure 6.1 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 resource of victim, the attacker needs at least 3 unit of resources. 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.

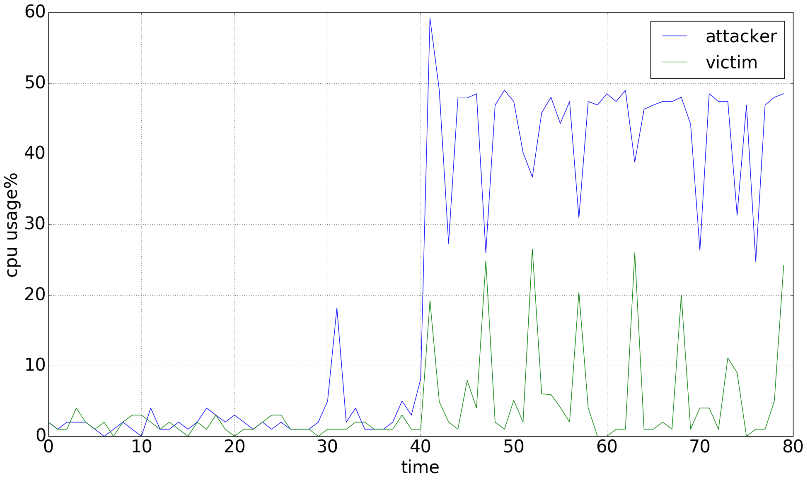


Figure 5.9: victim vs attacker cpu usage

But after we introduce crawl-request message, the thing is different now.Using crawl-request message in DDoS is very resource efficient 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 [23] 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 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 6.2:

As Figure 6.2 shows, the attack begins at the moment around time=30, the attacker easily wastes around 40 units of victim at the cost of around 1 unit on itself. The root of such imbalance is that: the attacker can store crawl-request as binary string format in the memory and reuse it whenever it needs; but victim cannot reuse the crawl-response, because the crawl-response messages contains different blocks every time.The victim can of course create one crawl-response for every block and store all of those crawl-response in its memory, but given the number of blocks in

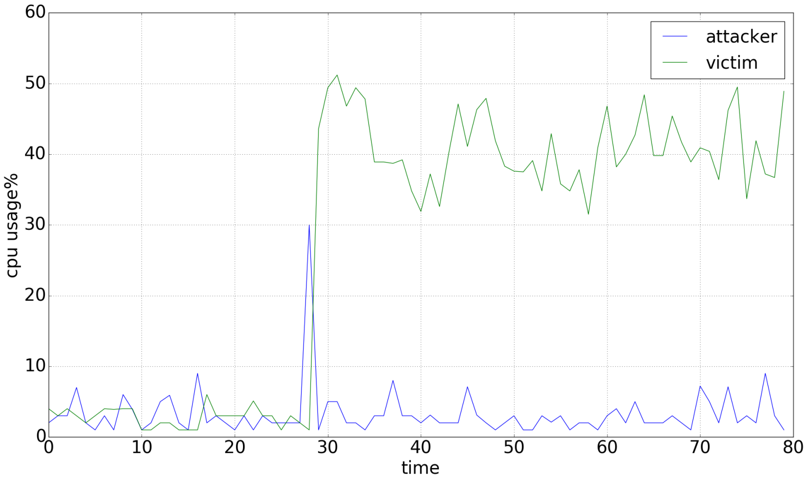


Figure 5.10: victim vs attacker cpu usage

database, this method will then cause very high memory usage.

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 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

6.1 Conclusions

In this article, our goals include

- Improve the current Dispersy Walker, adding Blocks collection ability to it.
- Improve the current Dispersy Walker, enabling it to calculate the trust value for other peers according to the historic 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 Multichain 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. We also build a reputation system to assign a score to known peers according to their historic behaviors. The score is binary: trusted or not trusted.

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 design which peer to visit, but assign 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 α is relatively high, the exploration speed slows down.

The second to be validated is the load balance. Walkers are not require to distribute their introduction-request to all other peers equally, but they should prevent concentrate their introduction-request on a small portion of peers. The experiment shows load imbalance 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 proves its effectiveness, while Teleport Walker only shows superiority over the current Dispersy Walker in the case α is relatively high.

Basing on the three experiment. The two walking strategies are both effective against Sybil Attack. They also meet the basic requirement of a Walker on exploration aspect and load balancing aspect. Using the Bias Random Walking will maximize the Sybil Prevention ability at the cost of exploration speed and a certain degree of load imbalance; using Teleport Walking strategy will cause less lose in exploration speed but it has relatively weaker Sybil Prevention ability compared with Bias Walking strategy.

With the independent nature, the reputation system and the two walking strategies in our new Walker, we achieve all three research goals.

6.2 Limitation

In our design, the Walker supports k hop trust, k can be any value. However, we can not find out a 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 an 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[24], $k = 20$ 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 article are conducted in simulated network. The simulated network is design to be close to the real network, but it cannot 100% emulate the real network. Without a thorough investigation on the real Dispersy network, we have no idea on:

- the number of Blocks a peer has on average
- the number of peers a single peer knows

- the number of sybils in the network
- the number of attack edges.

Without those knowledge, the simulated network we use in experiments cannot 100% emulate the real network.

In addition,we do not consider about the online and offline issue in 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.

Further more, in the simulated network, we do not take the NAT into account, we assume that the Walker can contact any other peers after NAT puncturing. However, in 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

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 overtime. 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 time stamp 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] 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`.

[6] 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.

[7] 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.

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

[9] 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.

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

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

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

[13] 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.

[14] 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.

[15] S. D. Norberhuis, 'Multichain: A cybercurrency for cooperation', 2015.

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

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

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

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

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

[21] 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.

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

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

[24] J. Travers and S. Milgram, 'The small world problem',