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

5th September 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 article 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.As time goes by, the onion grows bigger and bigger, people make break through in technology, culture and art, make the peel of the onion thicker and thicker.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 large scale cooperation ,for example people cooperate in Peer to Peer system like BitTorrent; we can also see the trouble caused by trusting wrong people, like 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
5th September 2017

Contents

Chapter 1

Introduction

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 is, what city he lives in, what telephone number he is using, and there is not 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 a 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 over 10 kilometers 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 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. 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 [14]:

> *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[12]:

> *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 [12]:

> *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 [1].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.

A similar problem happens in Ethernet: when Machine A wants to send a Data Link Layer packet to Machine B which holds certain IP address, A need to locate the machine B in the sense of MAC address.The problems of BitTorrent and Machine A, though differs on many aspects, still share some common features, these two problems can be abstracted into a high level one: given a network where every nodes inside has a unique address, and given a characteristic of a subset S of the nodes as a clue, how to find out the address of the nodes in S.In Bittorent scenarios, the "characteristic" means having the files you are interested in, in Ethernet scenarios, it means the having a specific IP address.In this article, the task to find out the address of such nodes is called "peer discovery"

The machine A, the old version BitTorrent clients and modern BitTorrent clients use three different strategies for peer discovery:Machine A broadcast the ARP request to all machines in its Ethernet, requesting Machine B's reply. old version BitTorrent clients directly ask the centralized server (tracker) for the address of all other clients which are downloading or uploading the interested file,a group of clients which are uploading or downloading a same file is called "swarm". A new version BitTorrent client using Kademlia Distributed Hash Table (e.g. Main Line DHT)[6] will use gossiping strategy: Client A send the address request (requesting the address of a client with certain ID) to A's k-neighbors, if at least one of them knows the address, it (they) will report it to Client A, otherwise they will reply with address of other neighbors who may know the target address. By doing so, Client A will get closer and closer to the target and finally discover it.

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.

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, [17] demonstrate 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 [17]. 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 article is organized as follow: Chapter 2 will describe the current Dispersy Walker, investigate its weakness and formally describe the research problem. Chapter 3 will introduce the related works to this article, which provide inspiration and motivation to this research.Chapter 4 will demonstrate the design of the improved Walker – transitive trust Walker. Chapter 5 will use validate the Walker Experimentally

Chapter 2

Problem Description

While the current Tribler Walker already has basic functionalities in place, it is not fully satisfying. First of all, the Walker is embed in a Community class, instead of being a independent modules, the Walker is actually a set of functions and attributes scattering across Dispersy, that makes it hard to maintain and develop. Tribler Team decides to take the Walker out of Community class and forms an independent module. It is a good chance to add additional functionalities by the way.Second, while the Multichain, which is a book keeping system of peers' historic behaviors, has been implemented, the Walker does not make use of it at all,hence the Walker is now fully unaware of the history of its known peers.Third, because the Walker is now ignorant about the historic behaviors of other peers, the Walker cannot defend Sybil Attack, because in the perspective of a Walker, all peers appear to be the same, no matter they are honest peers or evil peers.However, before embarking on solving the problems of current Walker, we need to make the problems clear first.

This chapter will start with introducing Dispersy, a module of Tribler where the codes of current Walker lies. The functionalities of Peer Discovery will be introduced while the irrelevant functionalities will be skipped. While introducing Dispersy Walker, we will also point out its flaws. Addressing those flaws, we will propose some potential attacks which can utilize the flaws to do harm to Dispersy network.

2.1 Dispersy and Walker

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 as-

pect of distributed system development. However, many of its functionalities are irrelevant to this article. In this chapter, we will only introduce the relevant functionalities.

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.

2.1.1 Message

Message is the basic unit sent between peers for communication,as it is name, a Message instance is a message. 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[5]
- 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.

2.1.2 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.

2.1.3 Walker and Peer Discovery

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.

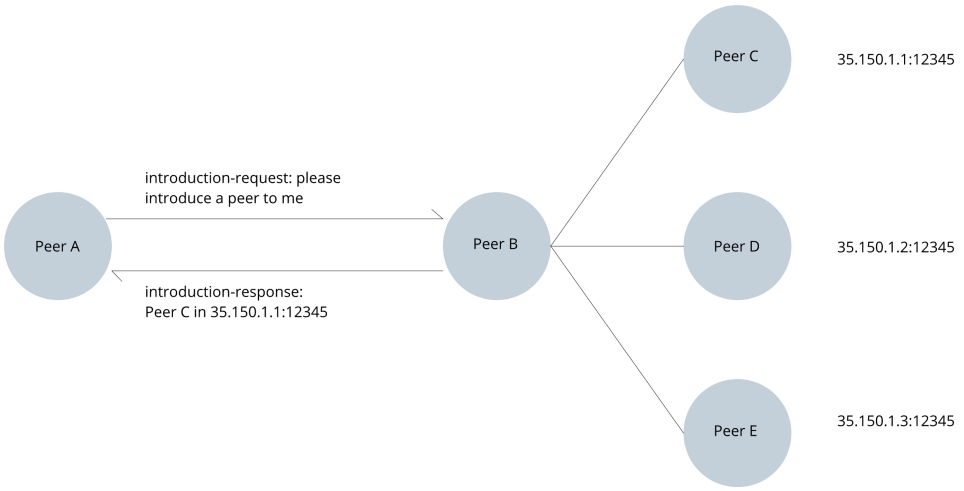

Figure 2.1: introduction-request and introduction-response

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.

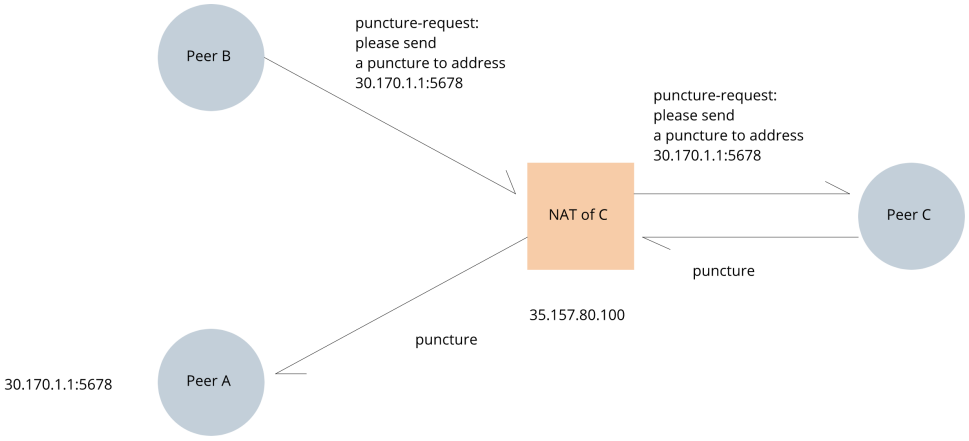

Figure 2.2: NAT puncturing

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.

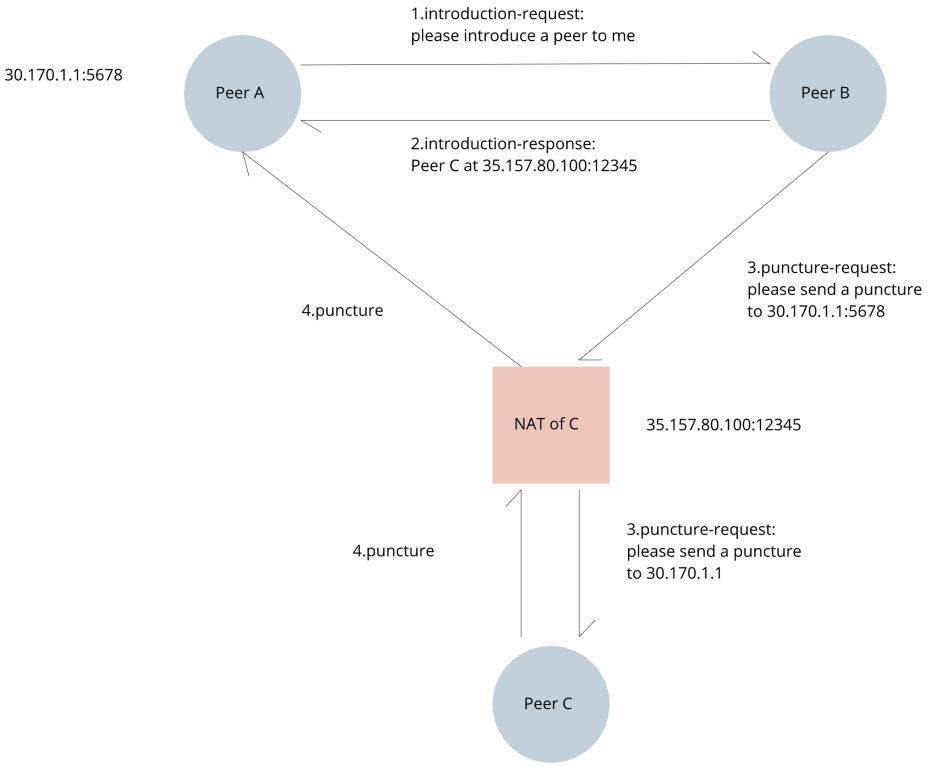

Figure 2.3: the whole work flow

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.

2.2 Potential Attacks

The current Walker in Dispersy does not have good defense mechanism, that makes it vulnerable to attacks.There are many ways to perpetrating an attack on Tribler

network, includes but not limited in:

First,active attack like DDoS. The most direct way is overwhelming the victim with introduction-request Message, that will exhaust the bandwidth and CPU resources. To evade defense mechanism like address filtering, which is described in [13],attackers can create many fake peer and launching it in different address. Dispersy identify a peer according to the self reporting public key from the peer, so creating countless peers are cheap – the attackers only need to randomly create public keys, and Dispersy will identify each public key as a real peer.And because of the presence of NAT, by running every single fake peer in a unique port, attackers can use a single IP address to support tens of thousand fake peers, that means the attackers do not need many IP address as well. By launching attack from different peers in different address, and send introduction request in a fluctuating rate and dynamic fake peers set[8], it poses difficulty for filtering defense.But as we mentioned, the NAT is a big trouble, the trouble is not only for honest peers, but also for attackers. To perpetrate a DDoS attack to a specific victim via evil peers, attackers need other peers to introduce the victim to all of his peers,and send puncture request before the incoming attack, it is a hard task, can only be finished by luck rather by skills.In fact, all attempts to actively attack a specific victim will likely fail because of NAT, but attackers can change their target to the whole network rather than a few peers. For example, the attackers can attack whatever peers introduced to them, or take down the trackers directly. Denying such attacks needs to enforce a defense mechanism on whole network, it is not only the effort of Walker, so we will not further discuss it.

Second,passive attack,which will not actively take actions but wait for other peers actions and react to it like the Sybil Attack describe in [17]. Sybil Attack was first described in [2].It is a class of attack where attackers create many fake identities (called ”sybil”), and launch attack using those sybils Many systems have a implicit assumption that an entity, which is a individual user or a organization user, should only have one or a few identities, hence a single entity can only impose limited influence on the whole system. However, attackers creating a lot of sybils will break those assumption, hence the attacker(s),though few in numbers, can still impose great influence, which they should not have, on the system. Sybil Attack is a broad concept, an example of potential Sybil Attack is described in [11]. This Sybil Attack is using sybils to boost reputation hence the attacker can obtain more contributions from the network. [11] develop an accounting system which can help a honest peer to determine how many megabytes it should contributed to another peer, that system can limit the amount of data to contribute to the attackers, reduce the benefit of launching this kind of Sybil attack. However,there is another kind of Sybil Attack that can not be prevented by such accounting mechanism: the attacker can target the peer discovery process and hinder honest peers to discover other honest peers. The attack is similar to the one described in [17].In [17], the attacker launch Sybil Attack with following strategies :

- case 1:When receiving a PING message from honest peer, return a random

ID

- case 2:when receiving a FIND NODE message, reply as the owner of the target ID
- case 3:otherwise:keep silent and collect information

The PING is a message similar to missing-identity message in Dispersy, FIND NODE is a message to request reply with the owner of certain ID, it is similar to ARP request that requesting the owner of a MAC address to reply.In case 1, the attacker pretend to own a lot of ID, which is the identity of Main Line DHT based BitTorrent client.In case 2, the attacker pretends to be the peer B that another peer A is seeking for,hence the attacker can start interaction with the peer A in the name of peer B. To be even worse, because peer A falsely treating the attacker as peer B, it will report this incorrect mapping (from peer B ID to attacker's address) to other peers who send A a FIND NODE.The the incorrect mapping will spread to all peers in the network. As a consequence, attacker will finally replace B to interact with all peers in the network.

As we mentioned in previous section, the peer discovery in Dispersy also heavily rely on the introduction from other peers.Peer discovery in Dispersy differs with peer discovery in MLDHT Bittorrent protocol on the aspect of the number of peers to be introduced: in MLDHT, a peer sends FIND NODE message (similar to introduction-request) to k adjacent peers and hence likely to be introduced with multiple peers; while in Dispersy, the introduction-request is only sent to a single peer.

Similarly, we can develop a Sybil Attack for Dispersy Walker:

- step 1: creating many sybils, run each sybil in a single port.
- step 2: all sybils visit trackers, inject their address into the peer list of trackers.When a new peer joining the network, it will visit trackers and probably get the address of a sybil.
- step 3: wait for incoming introduction-request, and reply with the address of one of the sybil

By step 3, the sybils will inject the address of sybils into the peer list of honest peers, and to be worse, a honest peer may also introduce the sybil it knows to another honest peer, by doing so, the attackers will fill the peer list of honest peers with sybil address so that the peer may have nearly 100% probability to visit sybils, or at least increase the probability of visiting sybils to a high level.If such attack succeed,the attacker will be able to use those sybils to impair the network: the attacker may try to introduce delay in network, cause degrading in performance, violate privacy by profiling user behaviors, hinder a peer by keeping redirecting it to sybils so that the peer will only know the address of sybil hence has no chance to contact with other honest peers etc.

Sybil Attack now shows the ability to attack not only Main Line DHT based BitTorrent network, but also Dispersy network. It is very likely to have variants that work in other similar Peer to Peer network. Although those variants may differs on some minor aspects, the ideas behind this kind of Sybil Attack are the same: creating sybils and introduce them to honest peers. So, developing a defense mechanism for one variant will offer inspirations for defending other variants.In this article, we will focus on defending such Sybil Attack in Dispersy Scenarios, be fore discussing the defense mechanism, we should first formalize problem:

Definition 1 (Network Model) *In a directed graph $G = (V, E)$ where V is the set of nodes representing peer and E is a set of edge (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 sybils.*

Definition 2 (Peer Discovery) *A node i in the Network in Definition 1 can freely visit any node j if there is an edge (i,j), node j will then reply address of unknown nodes;knowing a new node will result in adding a new edges in the directed graph*

To prevent visiting sybils, we need a strategy for the new Walker to determine which peer to visit every turn. The strategy should be based on the knowledge of the Walker, the knowledge include the knowledge about the historic behaviors of other peers and the topology of the network.

Definition 3 (Peer Exploration Strategy) *A Exploration Strategy can be defined as $v_{result} = S(G_{walker}, H_{walker})$, where G_{walker} is the network topology in the Walker's perspective, and the H_{walker} is the historic behaviors of the historic behaviors of other nodes, in Walker's perspective.*

The current Walker in Dispersy use a random walking strategy which randomly choose a known peer to visit.It does not need to be aware of the historic behaviors of other peers.To explore a more sophisticated Exploration Strategy, we need to obtain knowledge about historic behaviors.However, current Dispersy Walker lacks of a way to record historic behaviors. So, before developing new strategy, we should implement a system to record historic behaviors of other peers.

2.3 Research Goals

Summarize this chapter, we have the goals of this research: First, we need to implement an independent Walker. As we mentioned, the current Dispersy Walker is actually a set of codes scattering across the Community class, that makes it hard to maintain and develop. So, before we adding any additional functionalities to it, we should first take it out of the Dispersy and form an independent module.

Second, the Walker should have a book keeping system to record the historic behaviors of other peers. The current Walker knows other peers only by name, it is

unaware of the historic behaviors associated by those peers. As a consequence,in the perspective of the Walker, the honest peers and sybils appear to be the same. To obtain the ability to discriminate honest peers and sybils, the Walker need to know more information – the historic behaviors of peers.

Third,the new Walker should have a proper peer discovery strategy – a strategy which can prevent the Walker visiting sybils in some degree.That is the core functionality we want to implement.

Fourth, after the implementation of the new Walker, we need to validate it: testing its performance experimentally.

Chapter 3

Related Work

The input of a exploration strategy is the knowledge about the historic behaviors and the topology of network. This chapter will first discuss related works about store and utilize the historic behaviors, then discuss the works use topology of the network to combat sybils

3.1 Storing Historic Behaviors

To implement a more sophisticated Walker to prevent visiting sybils, We need a book keeping system to store historic behaviors associated by every identity. The first attempt for storing historic behaviors in Tribler is BarterCast[7]. 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 [10]. The concept of Multichain is similar to the concept of the famous Blockchain introduced in [9]. Before we discuss Multichain, we should first introduce Blockchain before discussing Multichain.

3.2 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 [18]while VisaNet have a peak performance of 56 thousand transactions per second [4].If Tribler directly uses Blockchain, such slow transaction processing speed may cause problems. So, Tribler team develop a variant of Blockchain – Multichain

3.3 Multichain

The concept of Multichain is first introduced in [10]. Later, a major upgrade is introduced in [16]. This chapter will introduce the concept of Multichain basing on the work of [16]

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 [16],the columns represent chains of different peers.

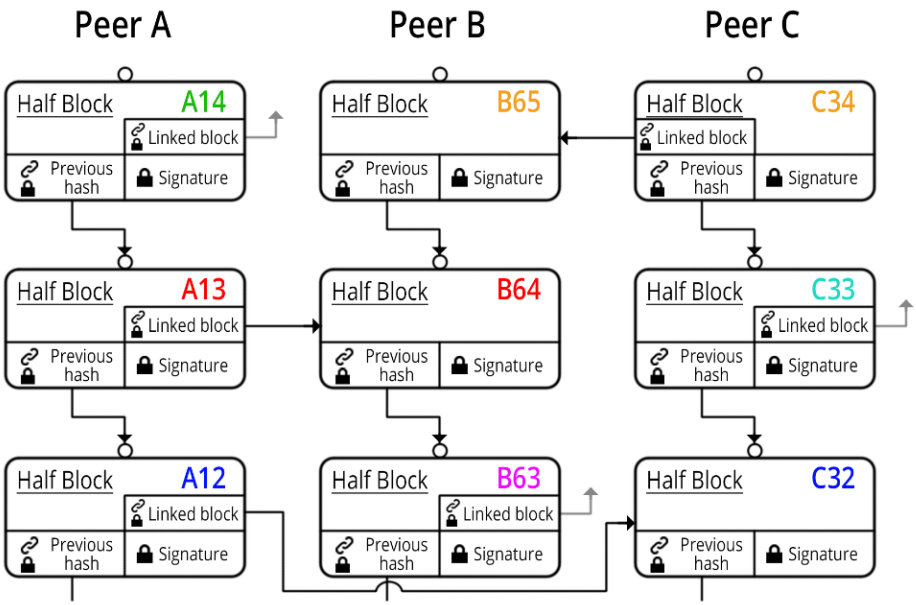

Figure 3.1: Multichain

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.

3.4 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.

3.5 Scoring System

[11] 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.

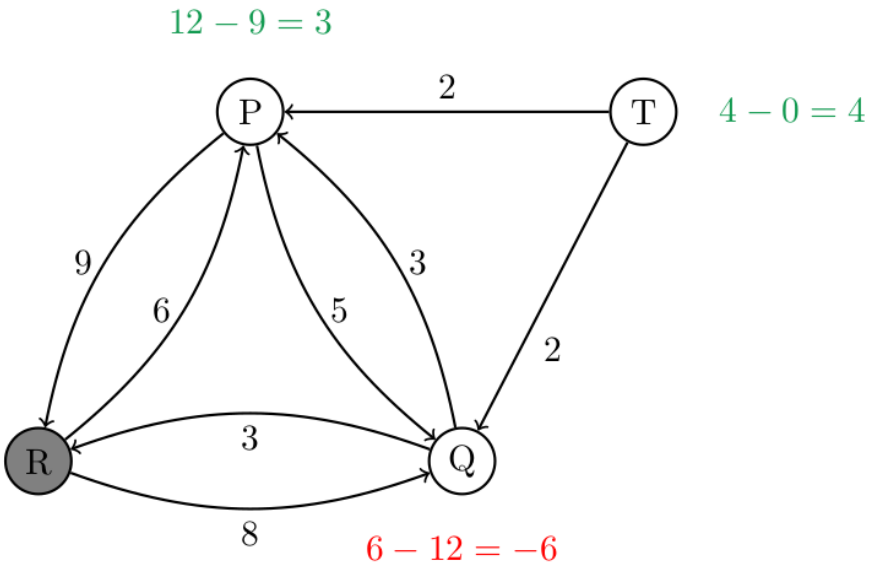

Figure 3.2: NetFlow algorithm

3.6 Walk strategies

[16] 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 [16] still provides valuable information.

The first walking strategy in [16] 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

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

The second walking strategy is focus walking, which assign the probability of a candidate (a peer to be chosen) according to its score. However, [16] 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 pseudo codes of this strategy is depicted in Algorithm 2. Notice that, Algorithm 1 and Algorithm 2 are copied from [16]

Algorithm 2: Pim Veldhuisen's Focus Walking strategy

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

Because the random walking strategy is the current strategy of Dispersy Walker, we are only interested in focus walking strategy. The strategy assign higher 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 [16] 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.

3.7 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[19] 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.

[19] 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

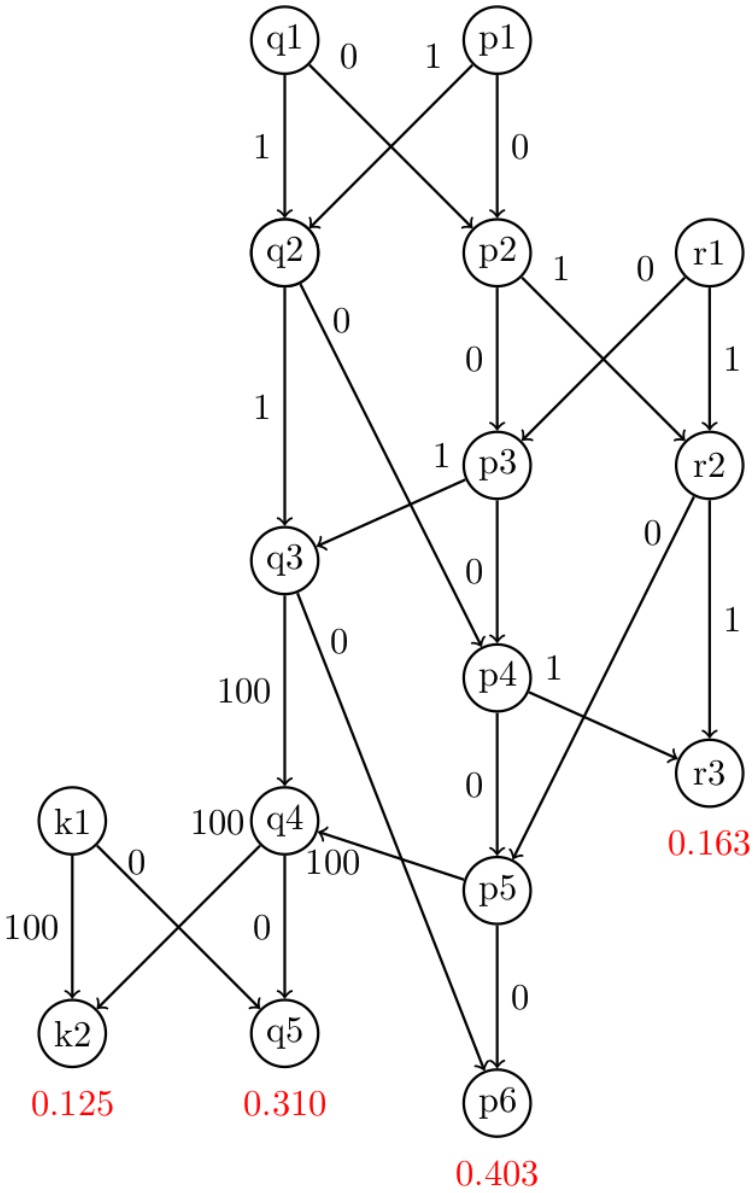

Figure 3.3: NetFlow algorithm

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 should not rely on any modules of Dispersy
- The new Walker can collect Multichain Block from other peers
- The new Walker should be able to calculate reputations of peers basing on the Blocks it has.
- The new Walker should be able to prevent visiting sybils in some degree.

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 aboved, 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.

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

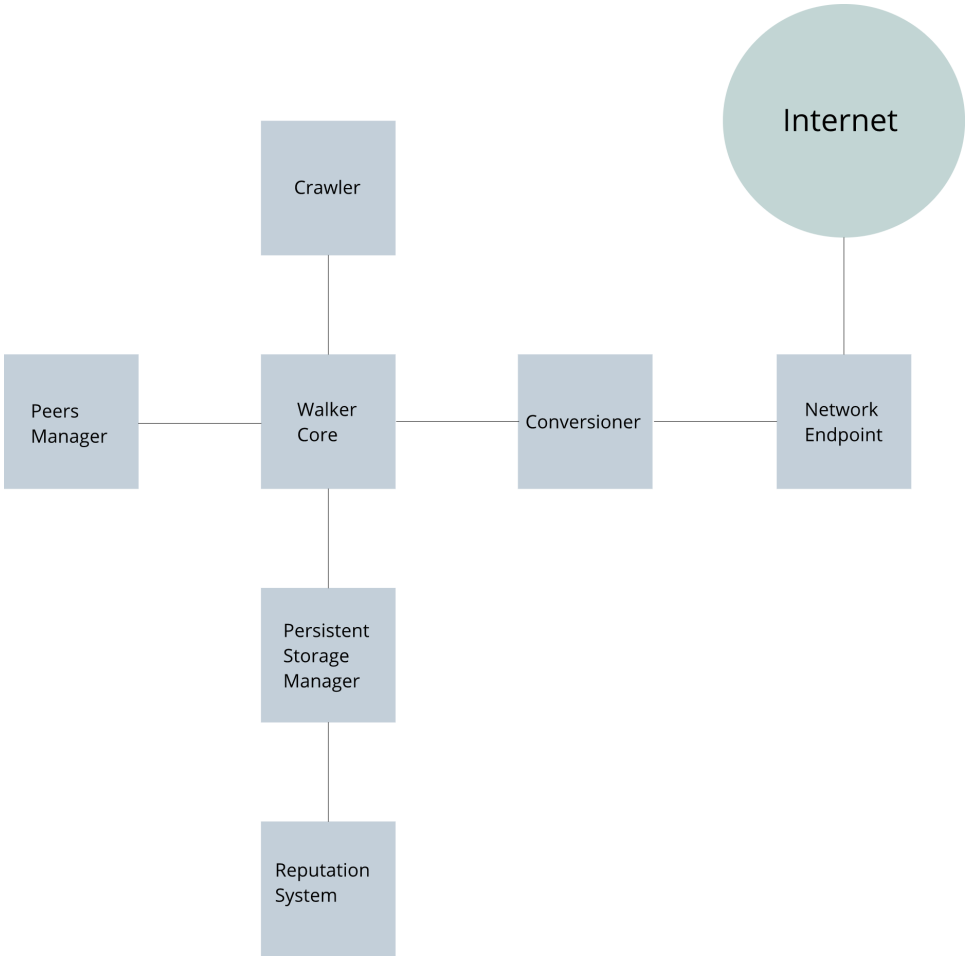

Figure 4.1: Walker Architecture

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 [11], but our graph differs with "Interaction Graph" on some aspects, we call our graph as Trust Graph.

Definition 4 (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 5 (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 6 (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 7 (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.

Algorithm 3: Bias Random Walking

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

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. 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 ≥ α then
        next_node = current_node.introduce() ;
    else
        if there is at least one trusted peer then
            next_node = random_trusted_peer ;
        else
            next_node = random_peer
        end
    end
end
def walk_towards(node):
      new_peer = node.request_neighbour();
      connected_neighbours.add(new_peer);
      new_peer.request_blocks();
```

Chapter 5

Validation

In theory, the two new walking strategy can reduce the probability to visit sybils, but we need a series of validation to confirm their effects. On the other hand, the two new algorithms are similar to the Focus Walking mentioned in [16] on the aspect that they favor some of the peers rather than treat all peers equally. Therefore, the two new strategies may also have a load balancing problem. In this chapter, we will validate the Walker experimentally concerning the issues mentioned above.

5.1 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

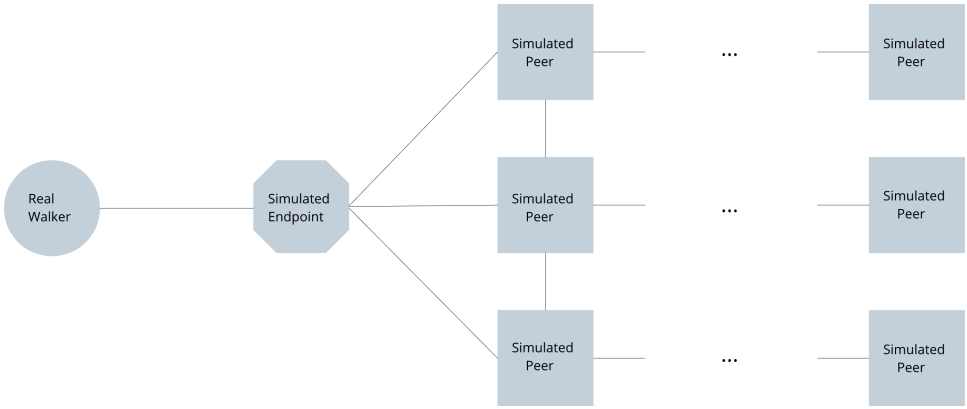

Figure 5.1: Sybil Prevention Validation

5.2 Sybil Prevention Validation

The validation experiment of sybil prevention performance is conducted in a simulated 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 5.2. The x-axis indicates the number of attack edges in the experiment scenarios and the y-axis indicates the evil ratio in this after 10 thousand steps

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 the second is even worse than the Random Walker. The

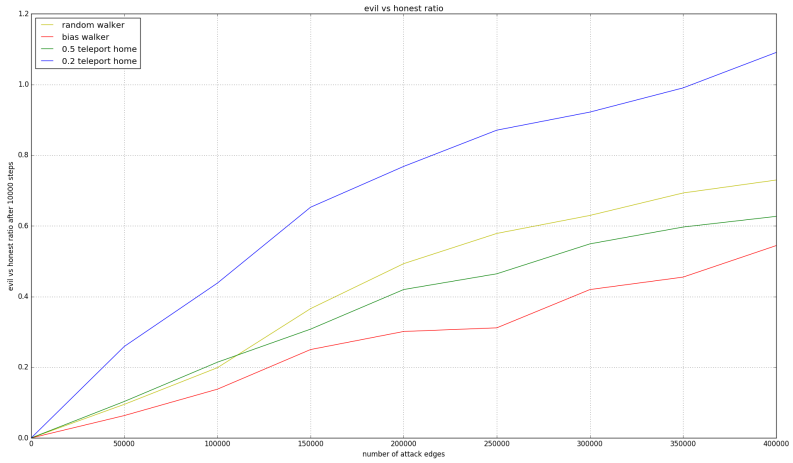

Figure 5.2: Sybil Prevention Validation

result is an evidence showing the effectiveness of the Bias Walking strategy and Teleport Strategy (with probability 0.5).

5.3 Exploration Validation

A basic requirement for a Walker is being able to explore the whole network after taking enough steps. 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 still conducted in a simulated network like the one in Sybil Prevention validation. However, we use a much smaller network with 2500 simulated peers. Because our goal is to test the exploration ability of the Walker rather than Sybil Prevention, we are not adding sybils in this network. We still use the four types of Walker mentioned in Sybil Prevention validation experiment, but in this experiment, all Walkers will take 50 thousand steps in the nework, the number of discovered peers over time will be recorded, the Result is shown in Figure 5.3 to Figure 5.6

In Figure 5.3 to Figure 5.6, x-axis represents the steps a Walker already take, and the y-axis represents the number of peers the Walker already visited. Notice that, visiting a peer twice will only count one. The curves in the four graphs converge to 2500 after some steps, recall that there are 2500 peers in total, reach 2500

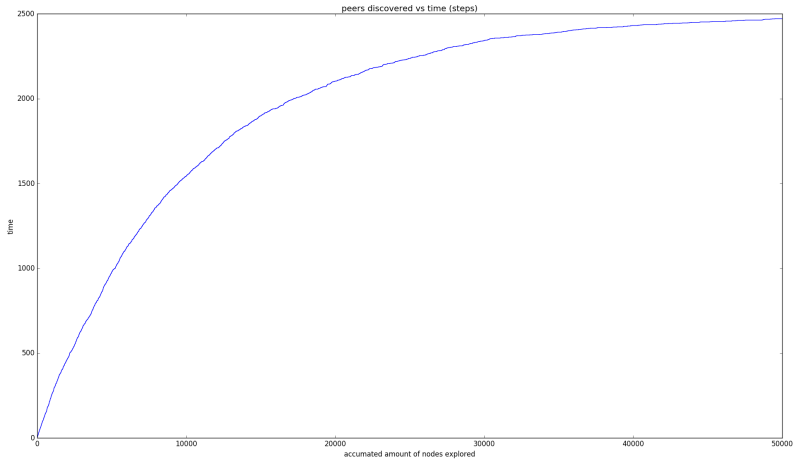

Figure 5.3: Random Walker Peer Discovery

means explores all peers. As the number of explored peers increases, it is harder and harder to explore unseen peers, 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, it is a downside but it is 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 histogram to show the distribution of the number of the received introduction-request. The result is shown in Figure 5.7 to Figure 5.10

In Figure 5.7 to Figure 5.10. The x-axis represents the number of introduction-request a simulated peer receives, the y-axis represents the number of peers receiv-

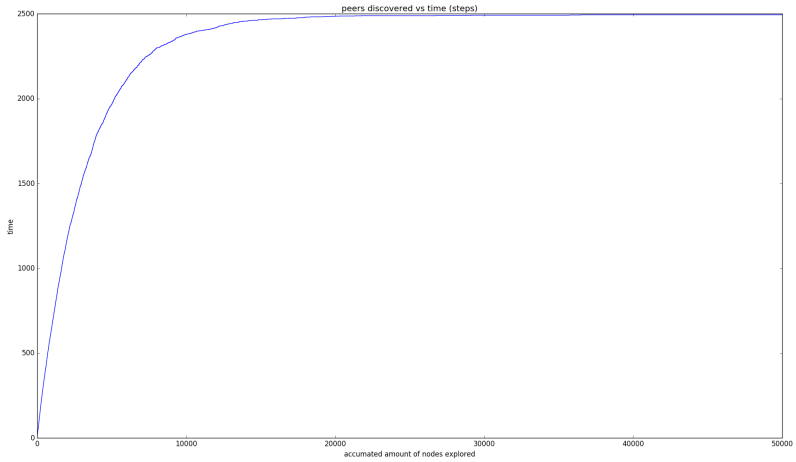

Figure 5.4: Teleport Walker with probability 0.2 Peer Discovery

ing 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. For the Bias Walker, the top 1 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 6 times the average. For Bias Random Walker,that is 7 times the average, that is the worst among the 4 Walker. However, 7 is only 40% more than 5, this slight increase in the load imbalance level is acceptable.So, all Walkers satisfy the load balance requirements.

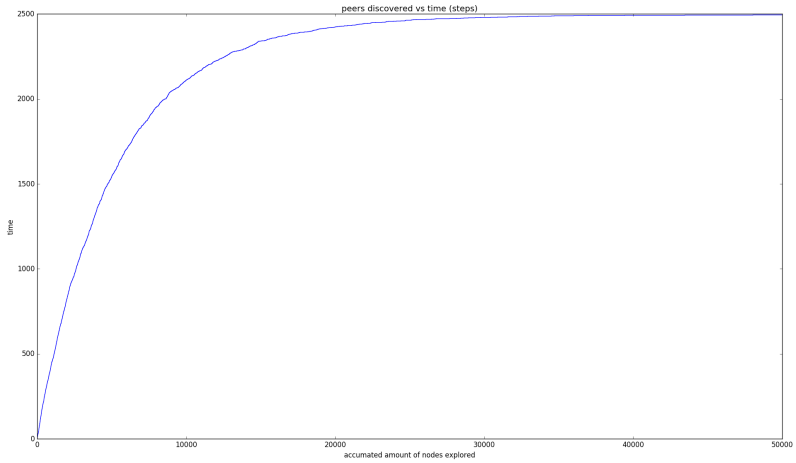

Figure 5.5: Teleport Walker with probability 0.5 Peer Discovery

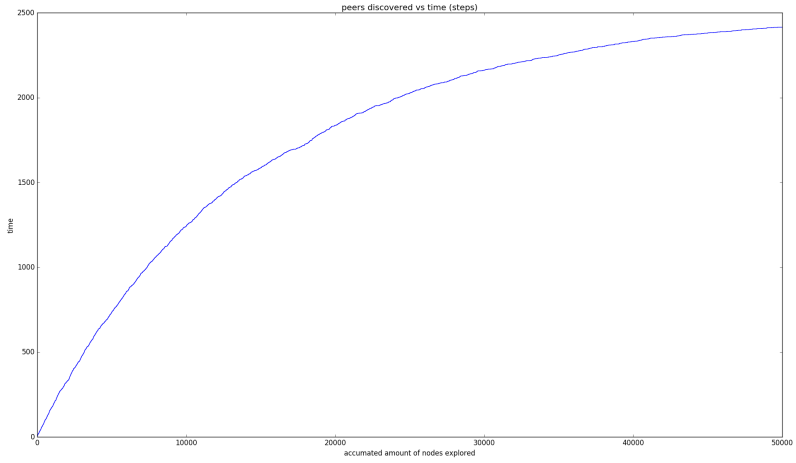

Figure 5.6: Bias Random Walker Peer Discovery

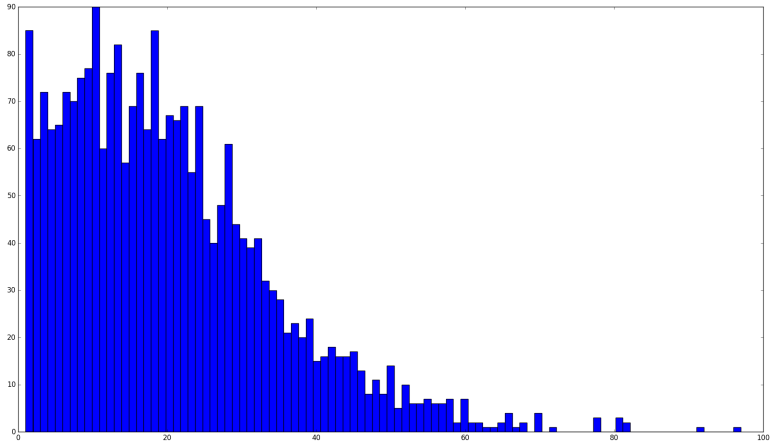

Figure 5.7: Random Walker Load Balance

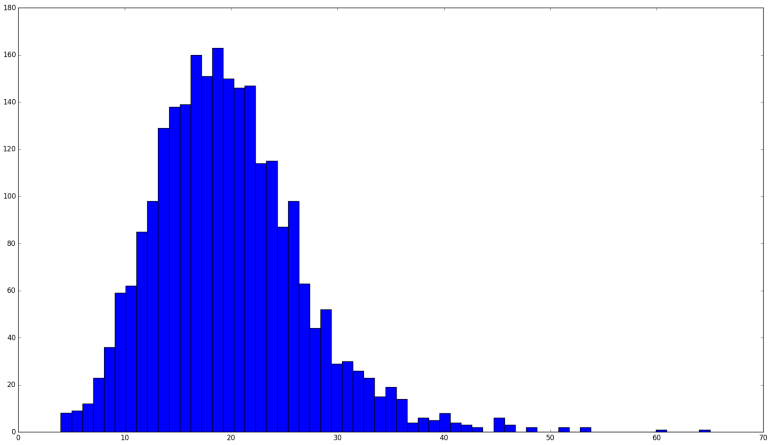

Figure 5.8: Teleport Walker with 0.2 probability Load Balance

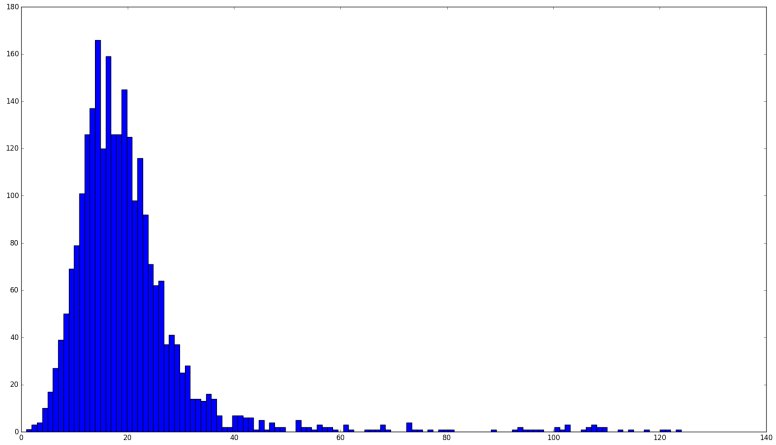

Figure 5.9: Teleport Walker with 0.5 probability Load Balance

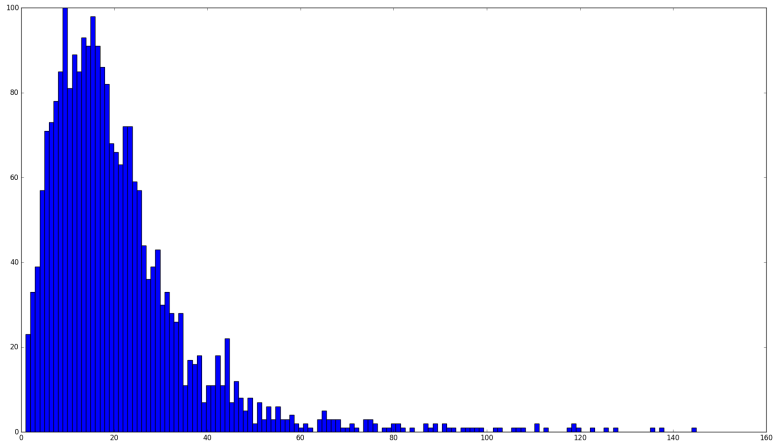

Figure 5.10: Bias Random Walker Load Balance

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this article, our goals include

- create a new Walker which retains functionalities of current Walker in Dispersy, the new Walker should not relies on any modules in Dispersy but should still compatible with current Dispersy network.
- The Walker should be able to treat other peers differently according to their historic behaviors
- Compared with current Dispersy Walker, the new Walker should be able to reduce its probability to visit a sybil peer.

To achieve the first goal, the new Walker still use the same protocol of Dispersy Walker. The new Walker fully retains the Message used by the Dispersy Walker, the structures of all types of Messages keep unchanged. The work flow of the Walker also keeps unchanged, that is: walker periodically sends introduction-request to other peers, will respond incoming introduction-request with introduction-response, and will use puncture-request to conduct NAT puncturing tasks.

To achieve the second 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 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 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.

The second 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 third 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.

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 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[15], $k = 20$ 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.

Bibliography

[1] Bram Cohen. Bitorrent protocol specification, 2017.

[2] John R Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.

[3] ERNESTO. Bittorrent still dominates internets upstream traffic, 2015.

[4] VISA inc. Visa inc. at a glance, 2014.

[5] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[6] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[7] Michel Meulpolder, Johan A Pouwelse, Dick HJ Epema, and Henk J Sips. Bartercast: A practical approach to prevent lazy freeriding in p2p networks. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8. IEEE, 2009.

[8] Jelena Mirkovic and Peter Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[9] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system.

[10] Steffan D Norberhuis. Multichain: A cybercurrency for cooperation. 2015.

[11] Pim Otte. Sybil-resistant trust mechanisms in distributed systems. MSc thesis, Delft University of Technology, December 2016.

[12] Rüdiger 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*, pages 101–102. IEEE, 2001.

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

[14] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[15] Jeffrey Travers and Stanley Milgram. The small world problem.

[16] Pim Veldhuisen. Leveraging blockchains to establish cooperation. MSc thesis, Delft University of Technology, May 2017.

[17] Liang Wang and Jussi Kangasharju. Real-world sybil attacks in bittorrent mainline dht. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 826–832. IEEE, 2012.

[18] wikipedia. Scalability-bitcoin, 2015.

[19] Haifeng Yu, Michael Kaminsky, Phillip B Gibbons, and Abraham Flaxman. Sybilguard: defending against sybil attacks via social networks. In *ACM SIGCOMM Computer Communication Review*, volume 36, pages 267–278. ACM, 2006.