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

30th August 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

Peer Discovery is a critical step in Peer to Peer System. However, it is vulnerable to Sybil Attack. While for different Peer to Peer System, the Sybil Attack have different attacking vector, the models behind many of those Sybil Attack share many common. This article aims to characterize the model of Sybil Attack on some Peer to Peer System, then try to explore suitable Peer Discovery strategy which is resilient to such Sybil Attacks.

Preface

(do it later)

TODO ACKNOWLEDGEMENTS

Changliang Luo

Delft, The Netherlands
30th August 2017

Contents

Chapter 1

Introduction

A distributed system is any system follow the definition provided by [13]:

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

Internet is a typical distributed system: computers distributed across the world own by individuals and organizations are ”independent computers” and for a typical user, the Internet appears to he/her ”as a single coherent system”.

> *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.*

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 accurate, Client/Server architecture can be defined as following[11]:

A more ”modern” one is peer to peer system. There are multiple definitions for peer to peer system,but here I use the one provided by [11]:

> *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)*

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 run bitTorrent clients, and only part of the BitTorrent clients have the file you want to download, hence the prerequisite 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 in 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, how to find out the address of the nodes in S. 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 provide three 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 new version BitTorrent client using Kademlia Distributed Hash Table (e.g. Main Line DHT)[5] 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 send the same request to their own k-neighbors, until they get the address.

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.

Introducing a centralized entity will easily solve the peer discovery problem, but will introduce new problem: the centralized entity will be the performance bottleneck, and the single point of failure.

The strategy of DHT Bittorent client is a desirable one. Strategies which rely on other peers to discover new peers ,not limited in gossiping approaches, show significant potential. But besides the performance aspect, we also need to concern about security issues. Unfortunately, such strategies are not perfect in security aspect, [15] 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 [15]. 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 evade attackers in the network where the majority of the peers are controlled by attackers.The majority of peers controlled by attacker is not an exaggeration, as

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, it has many dependencies on other Tribler components, 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 doesn't use it at all. Third, the Walker doesn't has a good defense mechanism for attacks, for example, Sybil attacks.

This chapter will briefly describe the Walker's protocol and identify the weakness of the protocol. Given the weakness, I will introduce some potential attacks

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 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 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. A Message consists of a header, a 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.

- Distribution: In this article, the only relevant Distribution is DirectDistribution which contains the sending time of the Message

- Resolution" In this article the only relevant Resolution is PublicResolution, which is empty, contains no data.

- Payload: 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 where 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.

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 to create and handle 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 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 first peers a fresh 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. Once 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 following Message involved in the peer discovery process: introduction-request, introduction-response, missing-identity, dispersy-identity, puncture-request, puncture.

- 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.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, 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 receiver of missing-identity.

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 enable multiple devices with different local address to share one public address. For example, in a network, device A with IP 192.168.1.2 and device B 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 A and B, 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 B 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 B is at 35.157.80.100:12345, but it cannot contact peer B, because B never send packet to peer A, B' NAT hence blocks all packet from peer A, B will receive no packet. To solve this, A and B needs cooperation. B 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 B, 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, the receiver of puncture-request should send a puncture Message to the peer A.
- puncture: puncture Message should be sent after receiving puncture request

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.

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 [12],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 created public key, 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[7], 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 [15]. 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 identity can only impose limited influence on the whole system. However, attackers creating a lot of sybils will break those assumption, hence as a single entity (or a few entities), they will have 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 [10]. This Sybil Attack is using sybils to boost reputation hence the attacker more contributions from the network. [10] 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 accounting mechanism in [10] – 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 [15].In [15], 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 miss mapping (from peer B ID to attacker's address) to other peers who send A a FIND NODE. As a consequence, attacker will finally replace B to interact with all peers in the network.

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 tracker, inject their address into the peer list of tracker.
- 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, or at least increase the probability of visiting sybils into a high level.If such attack succeed, the attacker may able 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, compared with developing defense for every variant, we should try to explore a general approach being able to counter all variants. Fortunately, there is a way to achieve this: preventing visit sybils, or at least reduce the probability to visit sybils. We define the problem as follow:

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 in the Network in Definition 1 can freely visit any nodes connected to it to request the 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 provide knowledge about historic behaviors to the Walker.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, implementing an independent Walker which does not rely on other Dispersy modules, this Walker should retain the functionalities of the old Walker; it be compatible with the old Walkers' network.
- Second, the Walker should have a book keeping system which can record the historic behaviors of a peer.
- Third, it should be able to show ability to prevent visiting sybils,at least reduce the chance to visit sybils compared with current Dispersy Walker
- Fourth, validate the new Walker in a Simulated network and evaluate its performance. The validation should include not only the sybil prevention ability, but also the basic functionalities as a Walker.

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[6]. 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 [9]. The concept of Multichain is similar to the concept of the famous Blockchain introduced in [8]. 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 [16]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 [9]. Later, a major upgrade is introduced in [14]. This chapter will introduce the concept of Multichain basing on the work of [14]

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

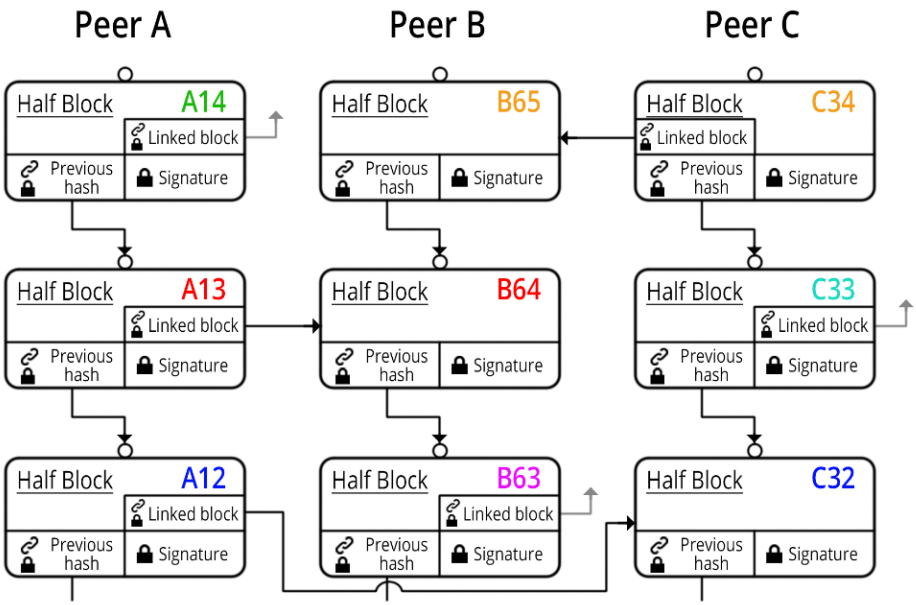

Figure 3.1: Multichain

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

[10] 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.

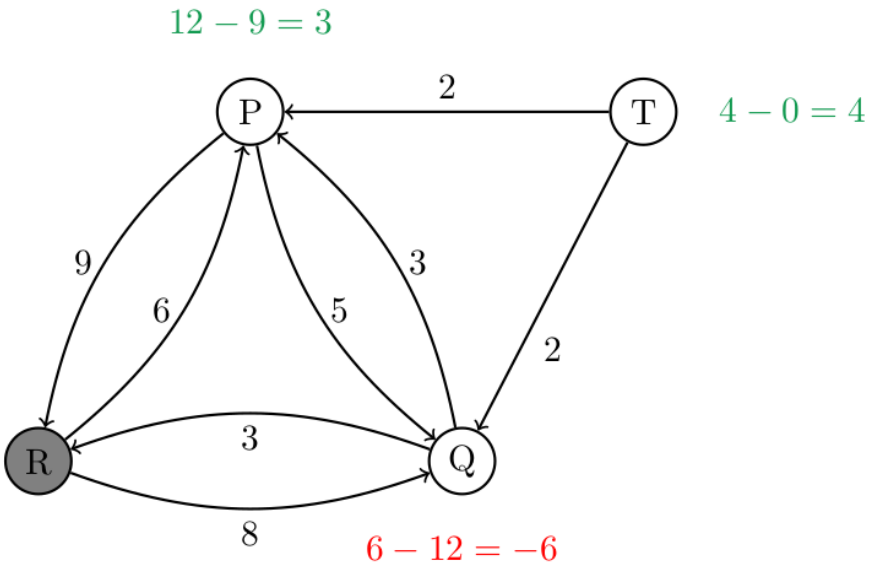

Figure 3.2: NetFlow algorithm

3.6 Walk strategies

[14] 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 [14] still provides valuable information.

The first walking strategy in [14] 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, [14] 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 [14]

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 [14] 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[17] 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.

[17] 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

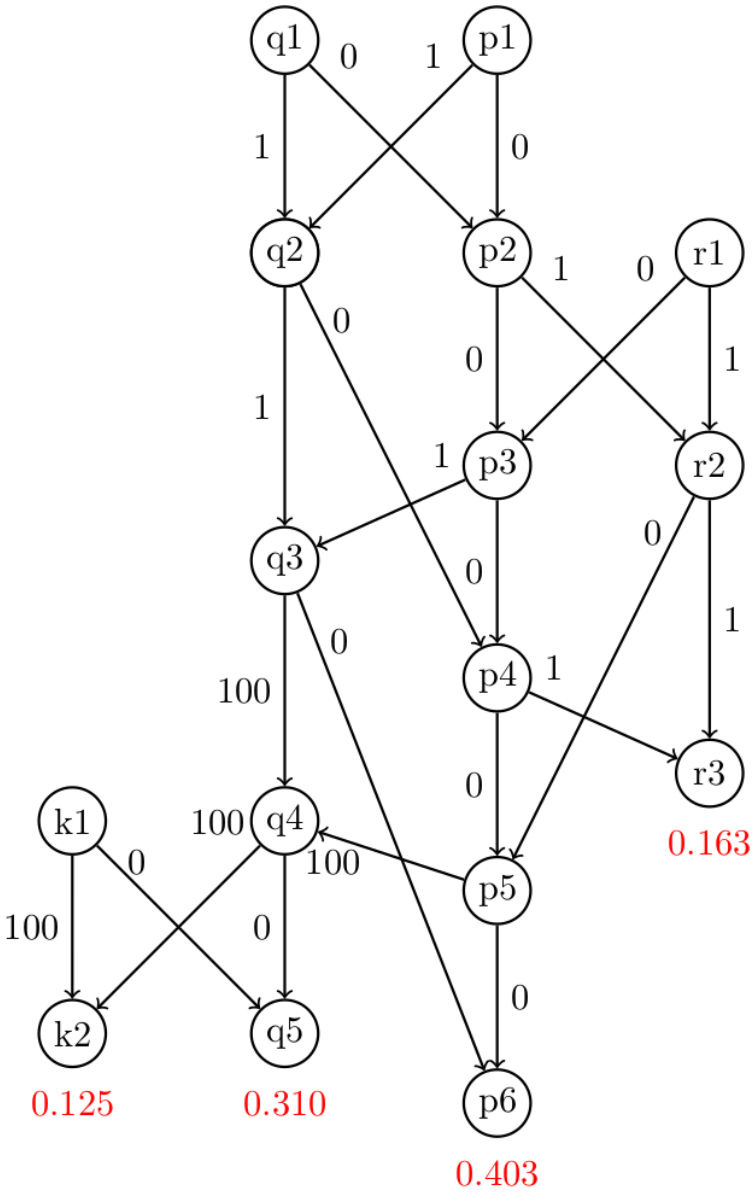

Figure 3.3: NetFlow algorithm

Chapter 4

Design of Transitive-Trust Walker

In this chapter, we describe the design of the new Walker. The new Walker retains the basic functionalities of current Dispersy Walker and also have new functionalities. We will mainly focus on the new functionalities including the Multichain Block collecting,reputation system and walking strategy.

4.1 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.

In our new Walker, we send a crawl-request whenever we receive an dispersy-identity message or a introduction-response from the peers.

4.2 Reputation System

We build our reputation System basing on a graph similar to "Interaction Graph" of [10] in Definition 5, 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 $k = 2$

4.3 Walking Strategies

4.3.1 Bias Walking

Algorithm 3: Pim Veldhuisen's Random Walking strategy

```
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();
```

4.3.2 Teleport Walking

Algorithm 4: Pim Veldhuisen's Random Walking strategy

```
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 random_number2 ≥ 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
    end
end
def walk_towards(node):
      new_peer = node.request_neighbour();
      connected_neighbours.add(new_peer);
      new_peer.request_blocks();
```

Chapter 5

Validation

The goals of this article is implementing a new Walker which can prevent itself walking into sybil region in some degree, in addition, the new Walker, as a Walker should be able to discover peer efficiently and prevent load imbalance. This chapter will validate the newly implemented Walker experimentally. The experiments include: Sybil Evasion experiment, Exploration experiment, Load Balance experiment.

All experiments are done with a single real Walker and a simulated network. All peers in the simulated network are passive, they will not actively walk to any other peers but passively waiting for the real Walker visiting them and then respond to it. For example, a real Walker should send an introduction-request messagge to another peer every 5 seconds, but a simulated peer will not send introduction-request to anyone.Except for introduction-request, a simulated peer use the same logic with real Walker to handle other messages, it can reply you with introduction-response, identity, crawl-reply etc.

Using simulated network is for resources saving consideration.For real Walkers, though they light weight, still consume around one hundred times than a simulated Walker.

5.1 Sybil Evasion Validation

For Sybil Prevention experiments, the Walker will walk in a network consisting of 400 thousand honest peers and 600 thousands sybils. All honest peers lie in the honest region and all sybils lie in the sybil region. The two regions are connected with some attack edges, the number of attack edges range from 50 thousand to 400 thousand with interval 50 thousand,totally 8 configurations, every kind of Walker will be tested with all 8 configurations. The number of visited honest peers and visited sybils will be recorded and used to calculate the evil ratio (number of visited sybils/number of visited honest peers)

There all three kinds of Walker will be tested:

- The first is fully random Walker, which pick a random peer to visit in every step.
- The second is teleport home Walker, which will visit the latest added peer unless it hits the probability to "teleport home", if there is at least one trusted peer, teleport home means teleport to one of the trusted peer; otherwise teleport to a known random peer.
- The third is bias walker, which takes random walk to known peers but assign higher probability to trusted peers.

The second and third Walker are both based on trusted peers. A peer is trusted if it satisfy one of the following:

- It used to upload some data to the verifier (you Walker).
- It is trusted by some of your trusted peers, and the number of hops in trusted chain is not greater than certain number h

The new Walker use $h = 2$

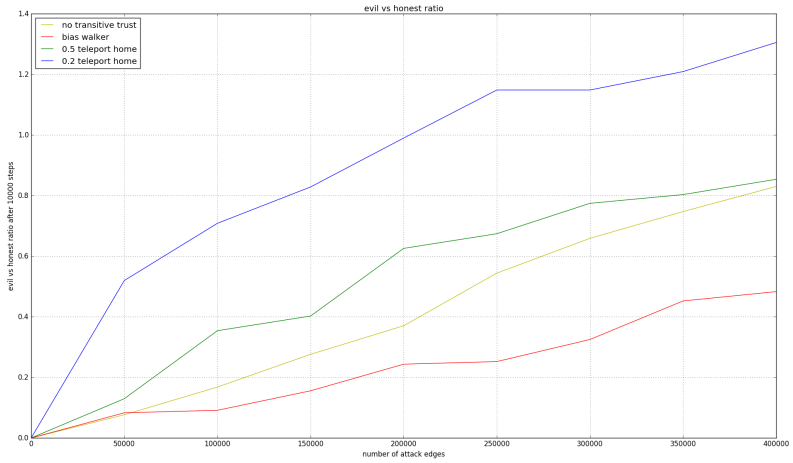

Figure 5.1: evil ratio for different walkers

bias Walker has best performance in sybil evasion, random Walker take the second place, and teleport home Walker with 50% probability to teleport home is the third.

5.2 Exploration Efficiency Validation

random Walker has worst exploration Efficiency, and Walker with 0.2 teleport home probability has the best exploration efficiency

Walker	steps required to exhaust 95% of peers
bias Walker	28758
0.2 teleport home Walker	10349
0.5 teleport home Walker	14989
random Walker	36733

Table 5.1: The steps needed to explore 95% peers in the network

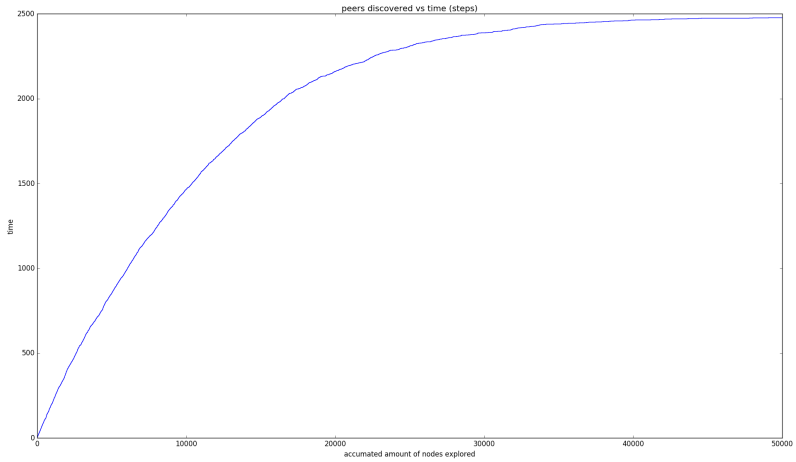

Figure 5.2: peers discovered (accumulated) over time for bias walker

5.3 Load Balancing Validation

References to figures are given with a capital letter for figure, as in “(see Figure 5.9)” or “in Figure 5.9, ...”.

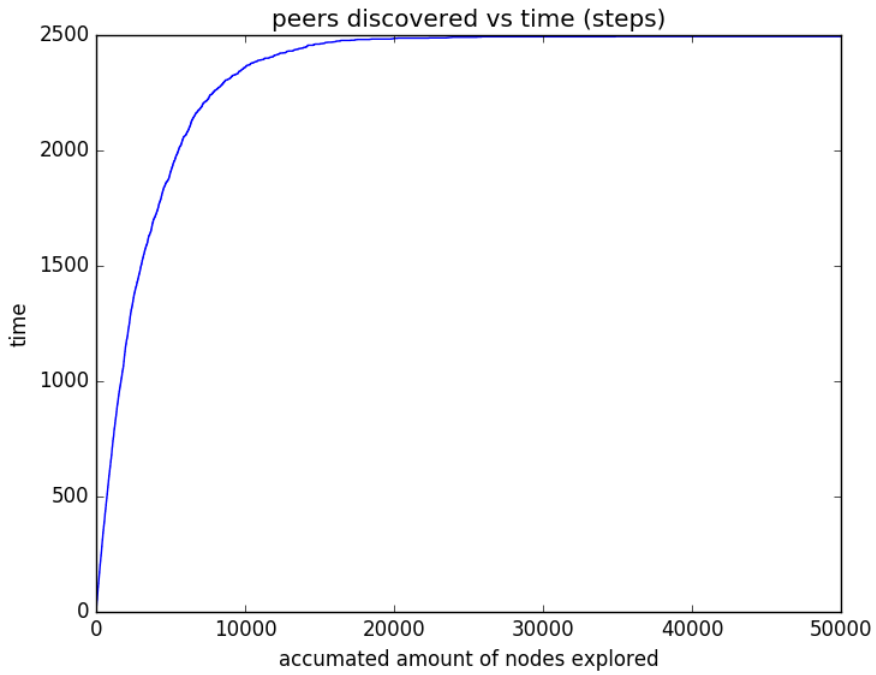

Figure 5.3: peers discovered (accumulated) over time for teleport walker with teleport home probability 0.2

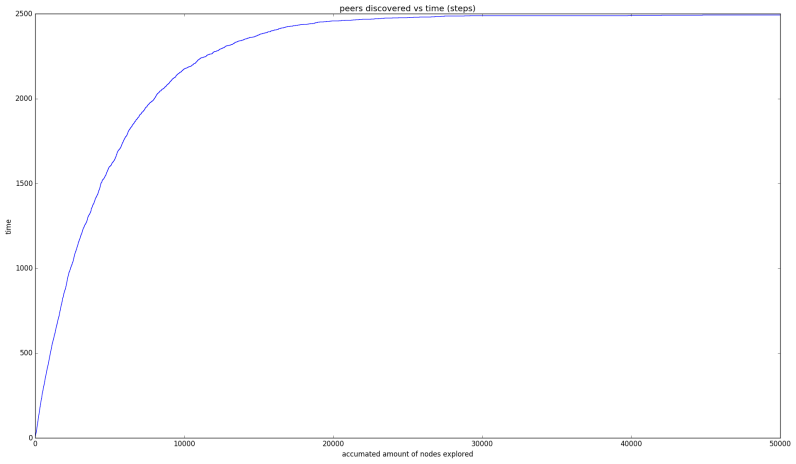

Figure 5.4: peers discovered (accumulated) over time for teleport walker with teleport home probability 0.5

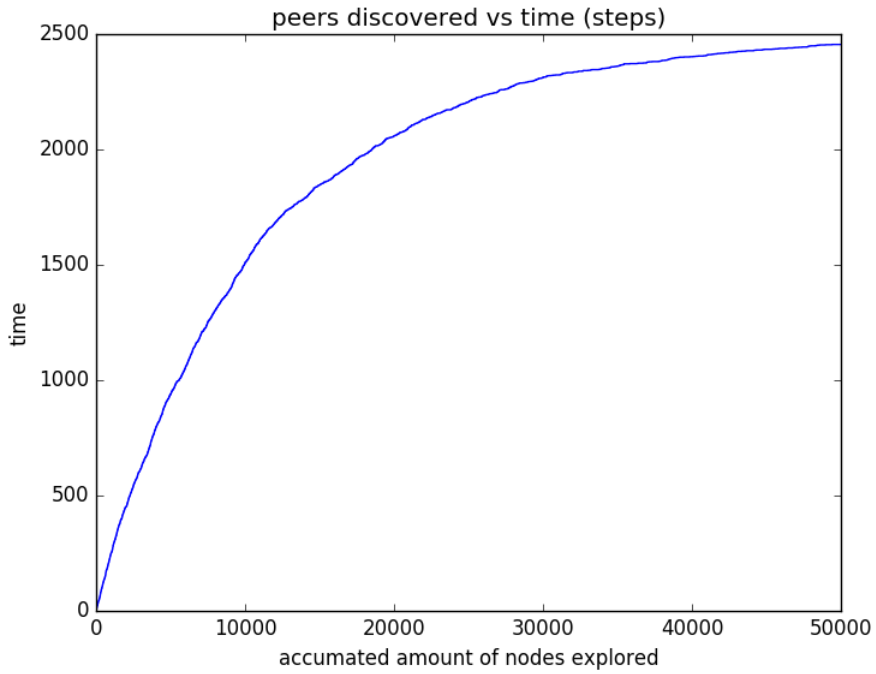

Figure 5.5: peers discovered (accumulated) over time for random walker

Take the fully random Walker as base line, no Walker demonstrate significant load imbalance

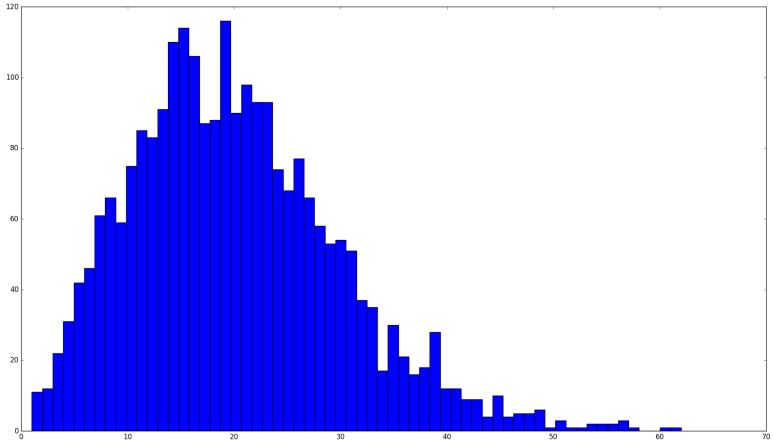

Figure 5.6: histogram of peers visted count with bias Walker

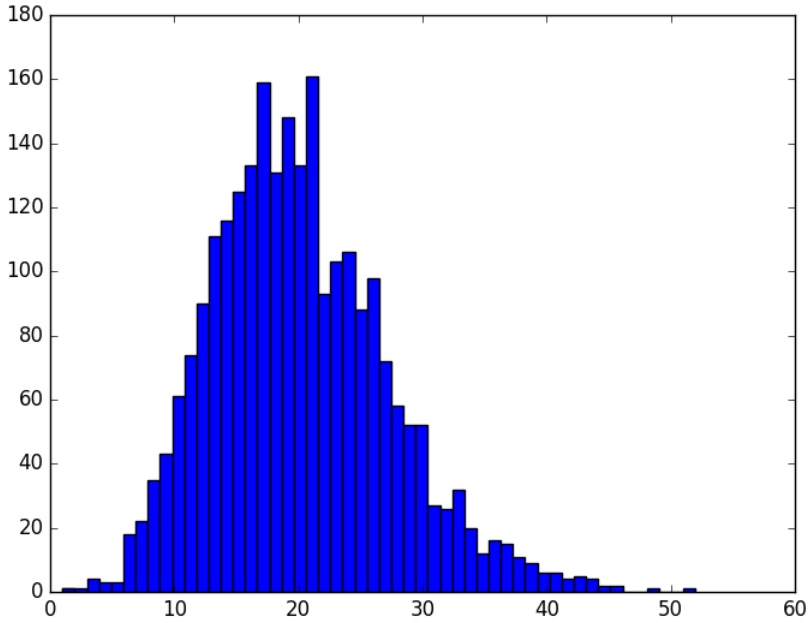

Figure 5.7: histogram of peers visted count with teleport walker with teleport home probability 0.2

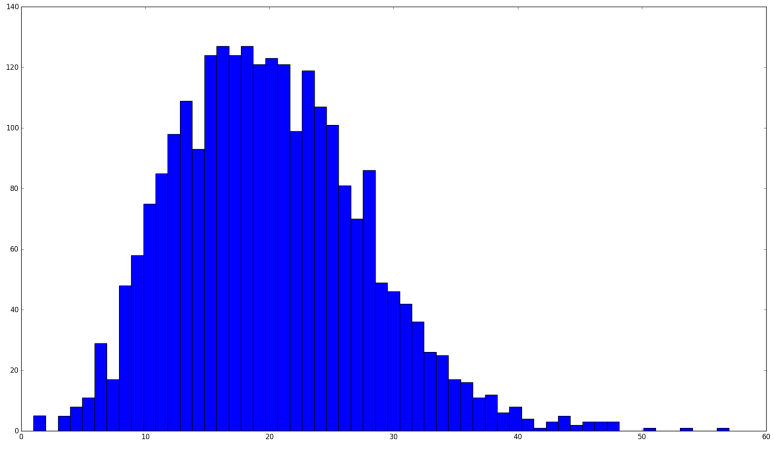

Figure 5.8: histogram of peers visted count with teleport walker with teleport home probability 0.5

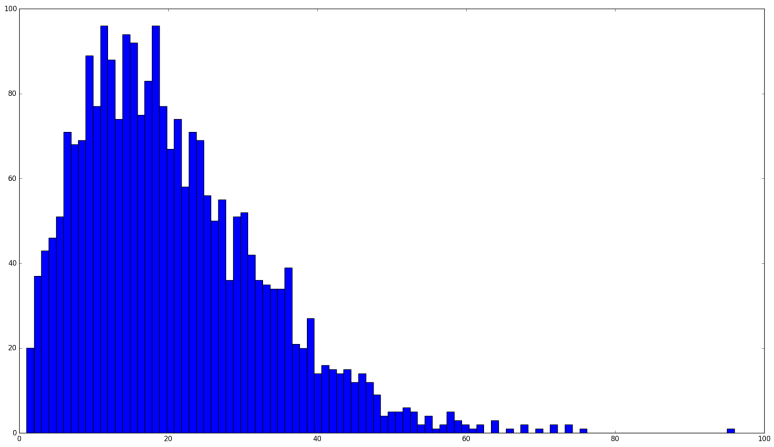

Figure 5.9: histogram of peers visted count with random walker

Chapter 6

Conclusions and Future Work

6.1 Conclusions

TODO CONCLUSIONS

6.2 Future Work

TODO FUTURE WORK

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] 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.

[6] 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.

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

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

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

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

[11] 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.

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

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

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

[15] 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.

[16] wikipedia. Scalability-bitcoin, 2015.

[17] 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.