Science is a wonderful thing
if one does not have to earn one's living at it.

Albert Einstein

CONTENTS

1

INTRODUCTION

Todays P2P applications need to be fast in order to be scalable. Blockchain applications like Ethereum and Bitcoin have a large userbase which puts more pressure on the speed performances of the applications. The users of the P2P applications find each other on the internet with discovery mechanisms embedded in an overlay on top of the OSI model of the internet. Users currently don't take the performance of other peers or the connection quality towards other peers into account when choosing what peers to work together with in the distributed application. In this work we try to improve the connection quality of peers that work together in the P2P applications by embedding a mechanism in the peer discovery mechanism of the overlay that prefers peers with a low latency. A low latency towards other peers is especially important in today's P2P applications.

1.1. THE IMPORTANCE OF LATENCY

Almost all systems have some requirements for latency, defined as the time required for a system to respond to input. Problem domains like web applications, voice communications and multiplayer gaming have latency requirements. In distributed systems latency requirements have become stricter with new applications like trading and anonymity systems. In this work I investigate methods to reduce the latency in distributed systems. [1]

1.1.1. LATENCY IN TRADING

A good example of a user application where low latency communication is important is the trading domain. In the past 30 years, trading has become faster. The time it takes to process a trade has gone from minutes to seconds to milliseconds. "Low Latency" would be under 10 milliseconds and "Ultra-Low Latency" as under one millisecond . It is estimated that 50% of trades in the U.S. are done in high frequency trading with an "Ultra-low latency". Thus, low latency is a major differentiation factor for exchange firms. Some firms state that a 1 millisecond advantage can save an exchange firm 100 million U.S. dollars. [2] An individual trader has numerous advantages when using trad-

ing in a system with low latency: [3]

1. Better decision making: A trader makes trading decisions based on the information the trader has from the market. Other traders send the prices and quantities they offer as orders to other traders. Let's say these traders maintain these orders in an order-book. If these orders arrive later, the individual trader is limited in it's trading decision making.

2. Competitive advantage towards other traders: When an individual trader can trade relatively faster than another trader due to low latency it has a competitive advantage. Let's say a price differentiation takes place, a price suddenly becomes lower. A trader with a relatively lower latency can act on it earlier than it's competitors and take advantage of the lower price before a price correction takes place.

3. Lower latency traders are served with a higher priority. Offering a lower price gives a trader always a higher priority as other traders would buy a product with a lower price faster. However, when the price is the same. The offer that arrives first is served. A trader with a high latency needs to lower its price in order to get a higher priority. If the high latency trader does not lower its price it is simply not served. Also, offers at the same price level with a higher priority have less adverse selection. [4] [5]

Moallemi and Saglam (2013) estimate the latency cost based on cross-sectional data on volatilities and bid-offer spreads in the U.S. between 1995-2005 from the dataset of Ait Sahalia and Yu (2009). The median latency cost approximately increased threefold in the 1995-2005 time period. To obtain the latency cost estimation the data set is used in a model that under simplifications calculates the latency cost. The model assumes an individual trader with a fixed latency of 500ms. As time increases, the cost for this latency also increases. As can be seen later on, the Tribler market has latency's around 150 ms. The assumption of a trader with 500ms is realistic in the Tribler context. For details of the model we refer to the paper of Moallemi and Saglam (2013). [2]

1.1.2. LATENCY IN ANONYMIZATION TECHNIQUES

Anonymization techniques require data to go through different nodes to make it hard to link the sender and receiver of a message. In one of the early anonymization techniques called mixes by Chaum developed in 1981 latency was a big problem. Messages are batched at nodes and a new batch is send forward at a node when n message are received giving a large delay between sending and receiving a single message. [6] In the TOR anonymization technique a solution to the latency problem is provided by forwarding messages in real time between mixes at the cost of the quality of the privacy. With TOR anonymization sender and receiver can be linked when all messages are sniffed in the global passive attack. [7] Because anonymization requires multiple nodes to which data travels a high latency between these nodes is unacceptable for a good working protocol. Figure 1.1 shows an overview of the anonymization in Tribler.

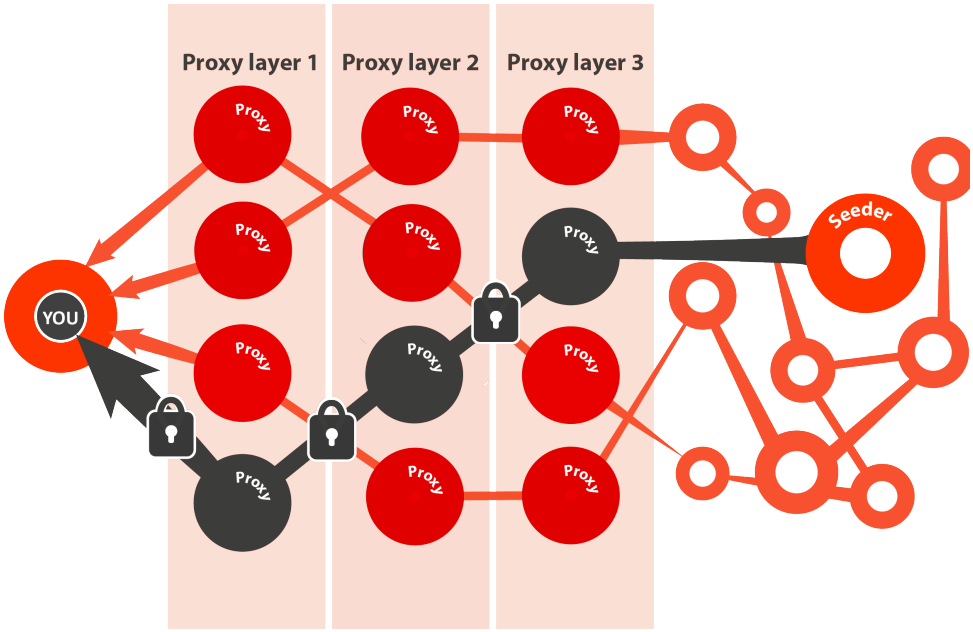

Figure 1.1: Anonymization techniques used in Tribler. There are three layers of the TOR protocol that make anonymous communication between peers.

1.1.3. LATENCY IN PARALLEL ALGORITHMS

In high granularity, fine-grain parallel algorithms one of the primary bottlenecks is communication latency. Only small amounts of computational work is done between communication events and the communication overhead is high because the message needs to be prepared and there is an electrical delay for signal processing between physical network links. These parallel algorithm have a wide range of applications in for instance data mining and knowledge discovery. The algorithms involve decomposing the data into parts based on available information and knowledge. The decomposition allows to do a parallel computation on multiple nodes. [8] [9]

1.2. LOW LATENCY OVERLAY

The current P2P discovery mechanism of the overlay needs to be improved to create an overlay where the peers connect to other peers to who they have a low latency. We call this new overlay the *low latency overlay*. In the new overlay, peers should introduce peers to other peers such that the other peer has a low latency towards the newly introduced peer and peers should only discover other peers to which the peer has a low latency. A latency can only be measured after a peer has been discovered thus the latency towards other peers have to be estimated with an algorithm.

Figure 1.2: Location space of with peers representing dots in the space. The distance between peers estimates the latency.

1.2.1. LATENCY ESTIMATION

The central idea in this thesis to estimate latency's is that peers can be modeled as coordinates in a geolocation space. In 2002 Zhang et al. [10] proposed the GNP system for estimating latency's on the internet based on real measured latency data. In the paper each peer has it's own coordinates in a space. The latency between peers can be estimated by taking the euclidean distance between coordinates in the space. To show this general idea Figure 1.2 shows a coordinate graph of the earth. Each dot represents a peer. The distance between two dots estimates the latency between these two peers. The challenge is to determine the coordinates of the nodes in the space such that the latency's are correctly estimated when calculating the euclidean distance between two coordinates in the space.

1.2.2. LATENCY AND DISTRIBUTED DENIAL OF SERVICE [DDOS] ATTACKS

When peers only contact a select subgroup of peers as their neighbours based on the latency toward these peers a DDoS prevention mechanism is provided. DDoS attacks make use of computers on the internet that have no or poor security. By breaking into these computers floods of data can be send towards a target, overloading the target such that the target becomes unavailable. The computers on the internet that are part of a DDoS attack can be at any location in the world. But, when a peer only is connected to neighbours that are physically close in the real world and to which the peer has a low latency the DDoS attack can only come from these computers. This limits the number of computers that can be used in the DDoS attack and thus provides protection against DDoS attacks. [11]

1.2.3. THE GNP APPROACH

The GNP algorithm proposed by Zhang et al. (2002) [10] proposes a method to calculate the coordinates of the peers based on real-measured latency data. The algorithm requires complete information on all the latency's between peers. It has as input an NxN

matrix that contains all the latency's measured between the N peers in the system. The GNP algorithm will calculate the error of latency estimations and minimize these errors. The error of one latency estimation is the deviation between the measured latency between two peers in the real world and the calculated distance in the geolocation space that estimates the latency between two peers. The GNP algorithm has an objective function which is the addition of all the errors of all the latency estimations in the system. The total number of latency estimations and errors is N^2 because there are N peers in the system. The objective function is minimized using the simplex downhill minimization algorithm. In each step of the simplex downhill algorithm the coordinates of the peers in the geolocation are recalculated and optimized to a lower error. [12]

Only optimizing the objective function is computational complex. A fair amount of m steps is required in the simplex downhill algorithm to minimize the objective function. The number of steps m can vary from 50 to 100000 steps. The more steps are taken, the better the algorithm minimizes the objective function and the lower the error is. Each step requires multiple calls of the objective function because for each coordinate the algorithm has to decide to increase or decrease the coordinate to get a better objective function output. One call of the objective function requires N^2 error calculations for a system with N peers. Thus the total complexity of the algorithm is $O(m * N^2)$. The complexity is polynomial but can be too complex in real world situations. With a large number of peers N the complexity increases squared and the accuracy decreases by lowering m.

1.3. RESEARCH QUESTIONS

In this thesis work we focus on creating a latency overlay that is computational efficient and also has a high accuracy in the Tribler P2P application. The following research question is answered:

How to create a computational efficient low-latency overlay that increases the quality of the connections of peers in the P2P network with low-latency connections?

To answer this question, a number of sub-questions are formulated:

1) Which methods to estimate latency's on the internet have been introduced in the past?

2) How to create a scalable latency estimation algorithm that can be run on Tribler?

3) How to embed the latency estimation algorithm in the low latency overlay?

4) What is the performance impact and accuracy of the new low latency overlay?

2

Related Work

2.1. Tribler

Tribler is a social-based P2P system that is an extension on BitTorrent. It includes social phenomena such as friendship and the existence of communities of users with similar tastes or interests that are exploited in order to increase usability and performance. The social phenomena are used in content discovery, content recommendation and downloading. Tribler Vision and Mission is the following:

"Push the boundaries of self-organising systems, robust reputation systems and craft collaborative systems with millions of active participants under continuous attack from spammers and other adversarial entities."

Since its foundings 10 to 15 scientists and engineers have been working on it full-time and added various new features. As of December 2014 Tribler has a build-in version of a Tor-like anonymity system. It gives superior protection than a VPN, but no protection against resourceful spying agencies. A reputation system is also included that gives incentives for users to upload files instead of just downloading them from the network. A screenshot of Tribler is given in figure 2.1.

2.1.1. Dispersy Overlay

Dispersy is the current OSI overlay in Tribler and the foundation of Tribler. It includes the Peer discovery mechanism, how to puncture NAT boxes and distributed database synchronization. Dispersy maintains a neighbourhood of peers for each peer. The neighbours of a peer are the peers the peer has already discovered and has an active connection with. Dispersy ensures the NAT firewall is punctured of peers in the neighbourhood. Every 5 seconds one peer is disconnected and Dispersy tries to discover a new peer. This keeps the peers in the neighbourhood different over time and prevents against the eclipse attack. The eclipse attack tries to control peers by controlling its neighbourhood and control the data flow to a peer. By refreshing its neighbourhood it is harder to do an eclipse attack on one peer.

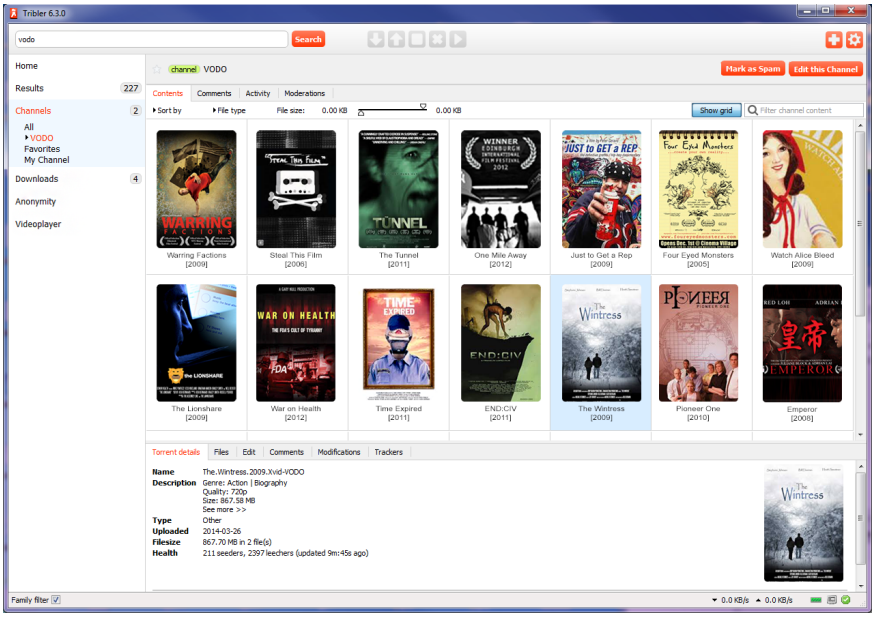

Figure 2.1: A screenshot of the Tribler application. [13]

2.2. INTERNET OVERLAYS

2.2.1. DHASH++

DHash++ is a distributed hash table (DHT) that provides high-throughput and low-latency network storage. A DHT is a hash table in a distributed fashion which makes the hash table very scalable because multiple distributed nodes work together. DHash++ uses the chord lookup algorithm to help it find data and is optimized for low latency. In order to make the requester contact low latency nodes DHash++ uses the Vivaldi latency estimation algorithm. Vivaldi is a similar algorithm comparable with GNP that uses coordinates to estimate latency's. It has the advantage that it is a distributed algorithm where GNP is used locally. Whenever DHash++ nodes communicate with each other they exchange coordinates. In this way a requesting node can predict the latency toward other nodes without having first to communicate with them. [14]

2.2.2. BINNING: TOPOLOGY AWARE OVERLAY CONSTRUCTION

Topological information about relationship in nodes is used to make better routing policies and reduce latency in overlay networks. Latency is reduced by putting nodes that are relatively close to each other in the same bin. Thus nodes in the same bin have a low latency toward each other. The binning strategy is simple, scalable and completely distributed. However, the scheme requires a set of well-known landmark machines spread across the internet. An application node connects to these landmarks and measures its latency and selects a bin based on its measurements. The latency's measured are divided into multiple levels that order the latency measurements. The ordering of the different

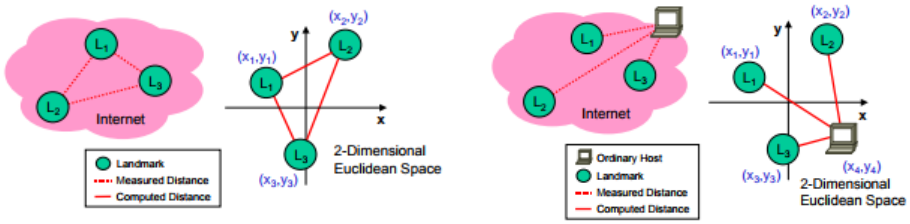

Figure 2.2: Part 1 and 2 of GNP algorithm. The left picture shows the first step of the GNP algorithm with landmark computation. The right picture shows ordinary host computation with ordinary hosts positioning themselves next to landmarks. [10]

levels to each landmark determines the bin of the node. The method described reduces the latency and performance in network overlay construction but violates a completely distributed system because landmarks are being used. [15]

2.2.3. CONSTRUCTING LOW-LATENCY OVERLAY NETWORKS WITH TREE AND MESH ALGORITHMS

Various Tree and Mesh algorithms that construct overlay networks are compared in this paper. To measure the performance of a Tree or Mesh the Pair-wise latency's between all the peers is taken as a performance metric. Also the diameter, the longest of the paths in the tree among all the pairs of nodes is taken as a performance metric. Meshes are created from the trees and compared in a different way. The result is that trees are faster to construct and save considerable amounts of resources in the network. Meshes, on the other hand, yield lower pair-wise latency's and increases the fault tolerance, but at the expense of increased resource consumption. [16]

2.3. LATENCY ESTIMATION SYSTEMS

Various algorithms and systems have been proposed to estimate latency's between hosts in peer-to-peer networks. A lot of these systems are coordinate based approaches with a number of hosts in a space. Each host has a position in the space, the distance between hosts represents the latency between these hosts. The coordinate based approach allows to calculate latency estimations quickly by computing the euclidean distance between two hosts. This makes coordinate-systems very scalable.

2.3.1. GNP ALGORITHM

The GNP algorithm consists of two steps. In the first step a subset of landmarks L from all the hosts H are chosen as landmarks for points of reference. Landmarks enable fast host position calculation in step 2 of the algorithm. Figure 2.2 shows the two steps of the GNP algorithm in a figure. There are normally around 20 landmarks. The coordinates are found by minimizing the difference between the real measured latency's between the landmarks and the computed distances between the landmarks. The minimization is done using the simplex downhill algorithm.

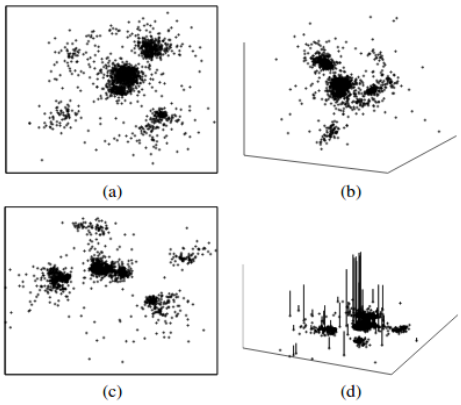

Figure 2.3: The node placement chosen by Vivaldi for the King data set (a) in two dimensions, (b) in three dimensions, (c) with height vectors projected on the xy plane, and (d) with height vectors rotated to show the heights. [17]

In the second step the coordinates of the other hosts are determined. The difference between the measured latency from an ordinary host to a landmark and the computed distance between an ordinary host and a landmark are calculated for every landmark. With the simplex downhill algorithm the sum of these differences are minimized. By this way the position of an ordinary host is determined with relatively low computation.

With a large number of landmarks the algorithm becomes computationally expensive. There is a squared relationship with the number of landmarks in the first step. With host coordination there is only a linear relationship between the number of landmarks and computation time. It is likely that with more landmarks the algorithm becomes more accurate but takes more time to compute. The trade-off between number of landmarks and accuracy has to be made. Because with small N the computation time of the first step can be marginalized, the computation time will be almost linearly dependent on the number of hosts H. [10]

2.3.2. VIVALDI ALGORITHM

Vivaldi is a variant on the coordinate-based systems on estimating latency's and is similar to the GNP algorithm in that it also tries to minimize an error function to find good coordinates for nodes. Vivaldi does this by conceptually placing a spring between each pair of nodes with a rest length equal to the measured latency between these nodes. Every pair of nodes exert a force on both nodes. The force of the first node has the direction towards the second node and vice versa. The strength of the node is equal to the displacement of the spring from rest. The net force on a single node is the sum of all forces from other nodes.

In the simple decentralized Vivaldi algorithm each node participating in Vivaldi simulates its own movements in the spring system. Each node maintains its own coordinates, starting at the origin. When the algorithms starts, the node communicates with its other nodes to obtain the coordinates of other nodes and measure the latency to other

nodes.

Each time the node communicates with another node, it moves it self in the direction of only that node's spring for a short amount of time δ, reducing only the error towards that particular node. Nodes continually communicate with other nodes so that the positions eventually converge to a low error. Figure 2.3 shows an example of node placements based on the King dataset.

Because the algorithm updates itself at every communication it has a bias to more recent samples or nodes that contacted a lot. A countermeasure to this bias would be to maintain a list of more recent samples and favor older samples and samples of nodes that aren't contacted frequently.

Choosing a right δ value is difficult. Large δ value inclines large steps are used in each epoch of the algorithm, but the result is often oscillation and convergence does not happen. Small δ values can lead to convergence but slow.

In order to obtain fast convergence and avoidance of oscillation Vivaldi varies δ depending on how certain the node is about its coordinates. Large δ values will help the node quickly go to a position with low error, while small δ values allows it to refine itself. The change in δ setting in Vivaldi also takes into account the error of the opposing node. When the error of the opposing node is high, the node should not get a lot of weight and thus δ should be lower. With this approach, there is quick convergence, low oscillation and nodes with high error have a lower weight. [17]

2.3.3. NPS SYSTEM

The NPS algorithm improves the GNP algorithm by decentralizing it. In the NPS system, hosts can serve as reference points to other hosts to define its base. This makes landmarks much less critical and landmarks become less of a bottleneck to the system. The GNP algorithm calculates node positioning with a centralized component. In GNP, if an ordinary host wants to calculate its position, it has to probe all landmarks. This makes the landmark nodes and their network access links a bottleneck to the system. If one landmark or the connection towards a landmark fails, the system can hardly recover.

In NPS the minimization function of the GNP algorithm is expanded such that each node computes its own coordinates. This makes the computation of landmarks linearly at each node. The newly calculated position is shared with other nodes and after 1 second of waiting the term is minimized again. The steps repeat until convergence is met which is achieved if after 3 consecutive iterations a landmark position has not moved by more than one millisecond in the euclidean space. The approach can embed 20 landmarks starting from their origin positions in approximately one minute and the resulting positions are just as accurate as the centralized approach. [18]

2.3.4. PIC: PRACTICAL INTERNET COORDINATES FOR DISTANCE ESTIMATION

PIC provides a decentralized solution that scales well and does not rely on centralized infrastructure nodes. Any node in the system can act as a landmark if the coordinates are already calculated. PIC addresses the problem that peers can choose to obstruct the system by for instance sending wrong information or manipulating its own coordinates.

Each new entering node to the system determines the latency to a set of landmarks.

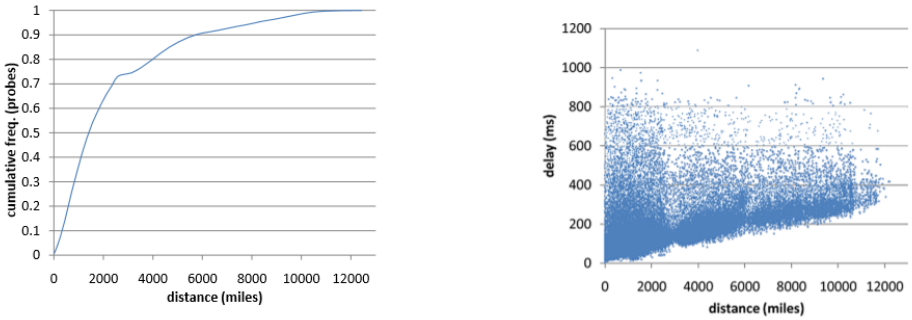

Figure 2.4: The figure on the left shows the cumulative distribution of the distance between consoles. The figure on the right shows the latency's measured for each distance between two nodes in miles. Both figures are from the experiments performed on the data from Xbox game consoles by Leet al al, 2008 [20]

The entering node also obtains the coordinates of each landmark. The new node then computes its coordinate by minimizing the error between the measured distances and computed distances between the new entering node and the landmarks. The authors of the paper experimented with several target error functions to minimize, the one that performed the best was the sum of the squares of the relative errors.

In the PIC algorithm three different strategies have been tested to choose a subset of landmarks out of all nodes. The PIC algorithm with different strategies were tested in different environments with a variable amount of routers. The result tells us that choosing some peers close and some peers randomly gives the best performance of the PIC algorithm in a decentralized setting.

To make PIC more secure a triangle inequality test is introduced. For most of the node triplets on the Internet, the triangle inequality holds. If an attacker lies about its coordinates or its distance to a joining node the attacker is likely to violate triangle inequality. The security test may also be useful when dealing with congested network links. When a link is temporarily congested, it will make the distance between the nodes in the link large and create a triangle violation. Nodes that require links that have congestion will thus be treated as an attacking node and ignored. [19]

2.3.5. LATENCY PREDICTION WITH GEOLOCATION APPROACHES FROM THE EARTH

Lee et al, 2008 tried to do latency prediction with geolocation data. Geolocation data is location data from the earth that is mapped towards IP addresses. The location data was retrieved from Xbox live game session information for Halo 3. The data set covers over 126 million latency measurements over 5.6 million IP addresses. Using the commercial MaxMind GeoIP City database from June 2007, the authors were able to provide the latitude and longitude for over 98% of these IP addresses.

It is hypothesized that the geographic distance between two consoles has a strong correlation with their measured latency. The great-circle distance algorithm is used to calculate the distances between two consoles at a different geolocation. The distance

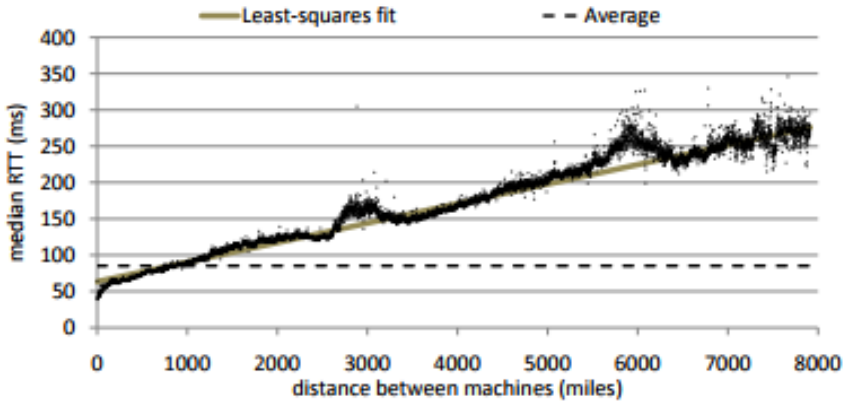

Figure 2.5: The correlation between the distance and latency. The latency data is the median of the data from the Halo 3 players database. The distance data is from MaxMind's IP-to-geo database. There is a clear linear relation between the distance and the median. The slope of the line is 0.0269 ms/mile and the explained variance is 97,6% ($R^2 = 0.976$).

between nodes varies between 0 and 12000 miles. Figure 2.4 shows a cumulative distribution function for the distance between nodes. About 14% of the console pairs traversed over 5000 miles. We have enough samples to examine the correlation between distance and delay.

In the right graph of figure 2.4 the relation between the distance and delay is shown. We see a very strong correlation between the geographic distance and the minimum latency measured between two consoles. Above this minimum there is a lot of noise. The geography of IP addresses is a useful predictor for filtering out pairs of IP addresses that are too far apart to have such a low latency. [20]

2.3.6. HTRAE LATENCY PREDICTION SYSTEM

Htrae is a latency prediction method merging both network coordinate systems (NCS) and earth geo-location approaches. The way this works is by geographic bootstrapping, initializing NCS coordinates in such a way that they correspond to the locations of the nodes in actual space. With better initial positions, Internet latency's can be better predicted.

Figure 2.5 shows the correlation between the distance in miles and latency's. The median is taken at each distance and a linear relation can be seen from figure x. The least-squares fit line is also drawn in the figure. The explained variance percentage is 97,6% which is high, so there is a strong linear relation.

When a new machine enters the system the Htrae algorithm works as follows. At first, the IP-address is looked up in the commercial MaxMind's IP-to-geo database. This gives an initial geo-location for the NCS. A Vivaldi-like algorithm is then used where a node moves in the direction of the forces that pull on the new node by nearby coordinates. The Vivaldi algorithm is adapted to use spherical coordinates instead of a linear

euclidean space to better model the spherical shape of the earth. An uncertainty model is also added that is used to calculate how strong a force to apply when updating coordinates: the greater a moving node's uncertainty, the stronger a force will be. Uncertainty is defined as the difference between the observed and calculated latency's.

The Htrae system implements additional things to improve the algorithm such as Triangle Inequality Violation (TIV) avoidance and autonomous systems correction. When updating a nodes coordinate, Htrae will skip the coordinate update if the measured latency exceeds the predicted latency by some number δ to remove TIV's. A big difference in the estimated latency and predicted latency is usually caused by inefficient routing between two nodes. Inefficient routing causes a large delay between two nodes compared to the sum of delays via a more efficient route. [21]

2.4. TRIANGLE INEQUALITY VIOLATIONS

Triangle Inequality Violations (TIVs) have an impact on the performance of neighbour selection in P2P systems. A TIV exist if a node A is close to a node B and the node B is close to node C, but node C is very far away from node A. These TIVs make it hard for latency estimation algorithms to properly estimate latency's because it makes it hard to model peers as coordinates in a geometric space. TIVs exist because of routing policies and the structure of the internet and these are not going to change. Thus TIVs will remain in the future. Various studies have reported Triangle Inequality Violations (TIV) in the Internet delay space. For instance, when taking two peers in real-world datasets as many as 40% of these peer pairs have a shorter routing path trough an alternative peer instead of the internet. Next to assymetric routing is common where the upstream and downstream capacities of a link are not equal. [22]

2.5. OPTIMIZATION FUNCTIONS

Various methods have been published to minimize an objective function. In latency estimation algorithms minimizing an objective function is often a part of the algorithm. The performance of the minimization function greatly effects the performance of the estimation algorithm. In this paragraph we discuss a few minimization functions.

2.5.1. SIMPLEX DOWNHILL ALGORITHM

The simplex downhill algorithm is an applied numerical method used to find the minimum or maximum of an objective function in a multidimensional space. It is applied to optimization problems for which derivatives are not known. For a n dimensional space it maintains a set of $n+1$ test points. The behaviour of the objective function is measured by changing the test points slightly. The algorithm extrapolates the behaviour of the objective function for each test point. So, for each test point is decided whether increasing or decreasing the test point would give a better result for the objective function. The test points are then replaced by better new test points based on what gives the best solution for the objective function. The algorithm takes several steps in which it measures the behaviour of the objective function when test points are changed and updates the test points in the direction that gives the best solution for the objective function. When the objective function is converged towards a minimum the algorithm quits. [12]

2.5.2. L-BFGS-B

The Broyden-Fletcher-Goldfarb Algorithm (BFGS) is an optimization method that tries to improve on simple optimization functions such as the simplex downhill algorithm with various mathematical tricks. The basis of the algorithm is similar to other optimization techniques in that it tries to optimize a set of test test points. Because derivatives are not available the Hessian matrix is too complex to be calculated. Instead the algorithm tries to estimate the inverse Hessian matrix to make decision on how to improve the test points for the objective function. The L-BFGS algorithm is a version of BFGS that uses limited memory. Only a few vectors are maintained that represent the approximation implicitly. This makes L-BFGS particularly suited to problems with very large variables. For instance more than 1000 variables. The L-BFGS-B algorithm extends L-BFGS algorithm to handle some mathematical constraints. [23] [24]

3

PROBLEM DESCRIPTION

The low latency overlay should be embedded in the current dispersy protocol. Dispersy currently supports NAT Puncturing and protects against the eclipse attack and NAT-timeouts. These features need to be maintained when implementing the low latency overlay. To make good design decisions with regard to the latency overlay we will explain in this chapter the peer discovery mechanism of dispersy.

3.1. COMPUTATIONAL, MEMORY AND BANDWIDTH EFFICIENT ALGORITHMS

With a large number of peers N in the P2P network the algorithms should still be computationally and memory efficient. Computing the lowest latency algorithm should be computationally efficient and accurate. If the computation requirements on the algorithms becomes too high Tribler does not function well anymore. There is extra latency introduced when an algorithm blocks a node for a certain amount of time. Most algorithms developed rely on a set of centralized landmarks to make computation efficient. In the decentralized Tribler setting it is not possible to assign certain peers as landmarks because all other peers should then know who the landmarks are. Tribler works with a peer discovery mechanism that does not support centralized components. The algorithms developed so far also require a lot of latency information. The GNP algorithm requires N^2 of measured latency's for a network with N peers. In P2P systems network can the number of peers can become millions of peers.

The algorithms should next to computationally efficiency also be memory efficient and be efficient in bandwidth usage. As peers collect latency's that are measured by other peers the number of latency's stored in memory and send over the internet can become large. If all peers maintain all the latency information they ever received and send share all their latency information to other peers the memory usage is N^2 where N is the number of peers in the network. A choice has to be made what latency's to send to other peers to lower bandwidth consumption and what latency's to store to lower memory usage. The choice depends on what effect the information loss has on the performance

of the latency estimation algorithms. In the next chapters we will further evaluate the design choice and evaluate the effect of the information loss in experiments.

3.2. DECENTRALIZED ALGORITHMS

The algorithms used in the overlay should be completely decentralized and cannot depend on centralized components like is the case in the GNP algorithm. Tribler is a decentralized system and does not favour some peers over others. Instead it has a peer discovery mechanism that determines which peers communicate with each other. Full decentralization eliminates central components which can be shut down, require more maintenance, can become bottlenecks and might provide extra security threats. Tribler is one of the few systems in the world that does provide full decentralization. Other P2P systems like bit-torrent require web sites to track users in the system. [25]

3.3. SECURITY REQUIREMENTS

The low latency overlay should be a secure system such that the Integrity of the system remains intact. It should not be possible to tamper the system and let certain peers become favoured over others when selecting neighbours. The messages send over the P2P network should be encrypted and the origin of the messages should be authenticated. Next to that the algorithm used in the overlay should be designed in such a way that no information is send over the network that would make it easy to tamper with the system. For instance, if the coordinates of individual peers would be send over the system it would be easy for a peer to attack the system by sending false coordinate information to other peers.

3.4. EMBEDDING THE LOW LATENCY OVERLAY IN THE PEER DISCOVERY MECHANISM

Peer discovery is constructed in such a way that it allows easy incorporation of a discovery of low latency peers. To show this we will first explain how peer discovery works in Tribler. In the dispersy implementation of the peer discovery mechanism a request and response mechanism is build to test the communication between two peers. The result is a list of peers called the neighbouring list that contains peers to which the peer always can exchange data. The communication lines between two peers in the neighbouring list are symmetrical by nature. If peer A has peer B in its neighbouring list, peer B also has peer A in its neighbouring list. Thus, Both peers A and B assume the role of client and server in the P2P network. To let the peer discovery mechanism work on the large scale of the internet, random computers have to be able to communicate to each other on the internet.

There are four phases in the peer discovery mechanism of Tribler. These four phases represent one step and multiple steps are a walk. By walking each peer discovers a set of known peers that are that peers neighbourhood.

1. peer A chooses a peer B from its neighbourhood and it sends to peer B an introduction-request;

2. peer B chooses a peer C from its neighbourhood and sends peer A an introduction-response containing the address of peer C; peer A will add the address of node C to its candidate list.
3. peer B sends to peer C a puncture-request containing the address of peer A;
4. peer C sends peer A a puncture message to puncture a hole in its own NAT.

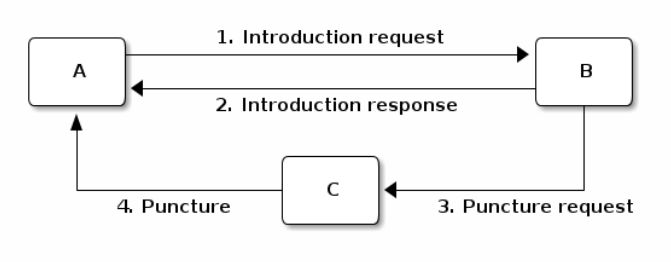

Figure 3.1: Overview of peer discovery in Tribler

Candidate lists can be easily manipulated with the well known Sybil attack. By creating a large number of pseudonyms that are colluding, the attacker can force to populate the neighbouring lists of victims by only introducing other pseudonyms to the victim. If a victim accidentally selects an attacker node, the attacker node introduces other attacker nodes which then introduce again other attacker nodes until only attacker nodes are in the victim neighbouring list. [26]

3.4.1. NAT PUNCTURING

To directly message a peer of a local network the NAT box has to be punctured. The puncturing is integrated in dispersy in the peer discovery mechanism. Firewalls on the internet are designed to block communication between two random computers on the internet for security reasons based on the client-server model and not for P2P networks. Most firewalls allow all outgoing connections and allow only incoming connections that are a response to an outgoing connection. This is great for the client-server model: A client can easily make a connection to a server from an outgoing port and the server can give a response to an incoming port that the firewall of the client only opens for this particular connection request from the client to the server. A server simply opens one incoming port that serves all requests from clients and clients send their requests to this open port. In P2P networks each client also acts as a server and the firewall should therefore allow incoming connections from other peers.

Network Address Translation (NAT) is designed for the client-server model and not suitable for a P2P setting. Figure 3.2 gives an overview of the NAT protocol. 64% of the computers connected to the internet do Network Address Translation (NAT) to hide the IP and port combination of computers from a local network to the internet. The ip addresses and ports of the local peers 1,2 and 3 are hidden from the peer on the internet with the NAT box. The NAT box has two IP addresses. One is available for the local network and one for the internet. The peer on the internet only communicates with the NAT box and the NAT box translates the ip,port combination to a peer from the local network.

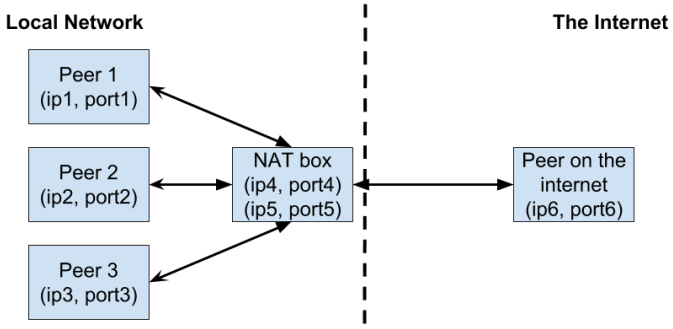

Figure 3.2: Network Address Translation (NAT). The NAT box has two ip, port combinations. ($ip4, port4$ is available on the local network and $ip5, port5$ is available on the internet.

The peer on the internet cannot distinguish between the three local peers if it wants to address one of the local peers and send messages to it. Therefore the local peers always have to act as clients and initiate the connection. The NAT box identifies and remembers the peer that initiated the connection and makes the translation for the peer on the internet that gives a response to the NAT box. The peer on the internet can never initiate a connection and is forced in the server-role. [27] [28] [26]

After both node A and C send a message to each other, the NAT firewalls of both nodes are punctured and the nodes are able to communicate with each other. This is called NAT puncturing. In the second phase of one step in the peer discovery mechanism peer A knows the address of peer C and will add peer C to its candidate list. Node C knows the address of A because it received it in the third phase of the the step from peer B. Node C then punctures a hole in its own firewall by sending a message to node A in the fourth phase. This message is blocked by the firewall of A and is never received. This does not matter because the goal of the puncture message from node C is to puncture a hole in the NAT firewall of node C. After node C has send the puncture message, node A is able to connect to node C. Node A has to puncture it's own NAT firewall by sending an introduction request message in the next step of the peer discovery mechanism.

3.4.2. ROBUST NODE SELECTION

To prevent against eclipse attacks dispersy has implemented a preventing node selection policy. In the next paragraph we will furhter explain on the dangers of an eclipse attack. A dispersy node will divide his candidate list into three categories:

I) Trusted nodes

II) Nodes we have successfully contacted in the past

III) Nodes who have contacted us in the past, either through.

a) Nodes that have sent an introduction-request; or

b) Nodes that have been introduced to another node.

Nodes that have replied to an introduction-request message are put into Category II, while the node they introduce is put in Category IIIb. Nodes that have send us an introduction-request are placed in Category IIIa. A special list of predefined nodes, i.e.

trackers is put in the trusted node category. A node which was introduced to us moved from Category IIIb to Category II after a successful connection attempt.

When selecting a node, a node will choose from its candidate list with pre-defined probabilities. The trusted node category has a probability of 1%, 49.5% is determined by category II and category IIIa and IIIb both get 24.75%. After choosing a category, the node will select the node by which the node had the most recent interactions with. This is due to NAT-timeouts. NAT-firewalls will close inactive connections after a certain timeout. If the NAT-firewall closes the port, any message sent to this node will never arrive. [26] [28]

Dividing the nodes into the categories described above has a dampening effect on a possible eclipse attack. If the attacker tries to perform an eclipse attack by introducing nodes that are controlled by the attacker, the size of Category III will increase. Increasing the size of this category only has a limited effect on the selection probability of this attacker node. However, if the attacker has a lot of resources he can still eclipse a node. This is why trusted nodes are also used by dispersy.

Every 100 steps a trusted node is contacted. When this happens the entire neighbourhood list gets cleaned removing any attacking nodes. Trusted nodes by itself are less susceptible to attacks as they are contacted by a constant stream of honest nodes. Attackers should ensure that there are more attacking nodes than honest nodes when contacting it for a successful attack. P2P networks now already have the size of more than 4 million nodes working concurrently, so attacking a trusted node seems unlikely to succeed.

Nodes from the neighbouring list after a certain amount of time. The amount of time is determined by the node timeout data measured by Halkes et al (2011) [27]. Introduced nodes are removed after 25 seconds and nodes that are send to or received an introduction-request from after 55s are also removed. In combination with a step time of 5 seconds the average node degree becomes around 11 seconds. [28]

3.4.3. ECLIPSE ATTACK

Eclipse attacks have large implications on P2P. In the eclipse attack an attacker can gain partly or complete control over the data that is received by a victim node. This is achieved by manipulating the candidate lists of the victim and its neighbours. When selecting a node it is important to take into consideration that attacker nodes might become part of the candidate list. If the colluding attackers control a large part of the neighbourhood of a victim node they can "eclipse" victims by dropping or rerouting messages that attempt to reach them. In the case of complete control over the neighbours of a victim peer (all neighbours are colluding attackers) the attackers gain full control over all the traffic toward the victim. [29]

Eclipse attacks have large implications on P2P systems that use block-chain. They allow the attacker to filter the victim's view of the block-chain, use computing power of the victim for its own use or separate the the network into two parts creating allowing the attacker two create two separate block-chains. (See Figure 3.3). Next to that the eclipse attack is also a useful building block for other attacks:

1) Engineering block races A block race occurs in a block-chain when two miners discover blocks at the same time. One of these miners receives mining rewards for that

block and his block will become part of the block-chain while the other miner will be ignored and create an "orphan" block. Attackers can forge block races by holding back mined blocks that are mined by eclipsed miners. Once a non-eclipsed miner discovers a competing block the block mined by the eclipse miner is released later resulting in an orphan block for the eclipsed miner.

2) Splitting mining power By eclipsing a large part of the miners from the rest of the network, the 51 % mining attack becomes easier. The attacker gains control over 51 % of the mining power in the network which allows to create a separate block-chain (Further details). To make the reduction in mining power from eclipsed miners less detectable, miners could be eclipsed gradually or intermittently. Figure 3.3 shows a network where eclipsed nodes split the network in two. This split could be used to launch the 51 % attack.

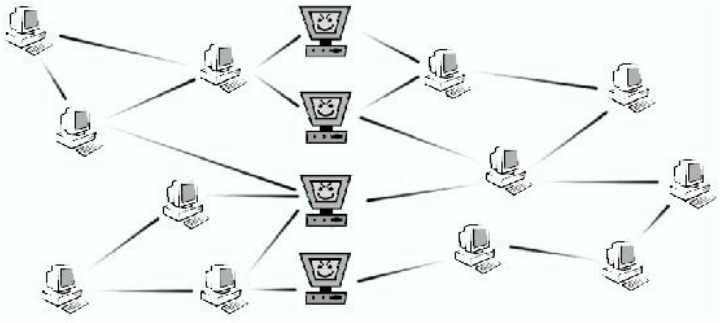

Figure 3.1: An Eclipse Attack: the malicious nodes have separated the network in 2 subnetworks.

Figure 3.3: Separating a network with the Eclipse attack

3) Selfish mining The attacker can decide to eclipse certain miners to make sure that other miners that are controlled by the attacker get more mining power. This is realized by blocking all discovered blocks by eclipsed miners. Later in time the attacker increases the mining power its own miners by only giving a limited view on the block-chain to eclipsed miners obstructing the mining of eclipsed miners even more. The fraction of nodes used to eclipse other miners is denoted as a and the fraction of nodes that is used for honest mining is denoted as b. When more miners are eclipsed a is increased and b is decreased. However, with high a mining becomes easier for the fraction b of honest miners left.

4) 0-confirmation double spend In a 0-confirmation transaction the attacker exploit systems where a merchant gives a confirmation of the transaction to a customer before the transaction is verified by the block-chain. This happens sometimes in systems where it is inappropriate to wait 5-10 minutes before a transaction in a block gets confirmed. For instance in the retail service system BitPay or in gambling sites like Betcoin. The coins spend by the customer to the merchant is double spend by the attacker. The attacker first eclipses the merchant. When the merchant wants to confirm transaction T as payment for the goods of the customer, the attacker double spends the bit-coins in the

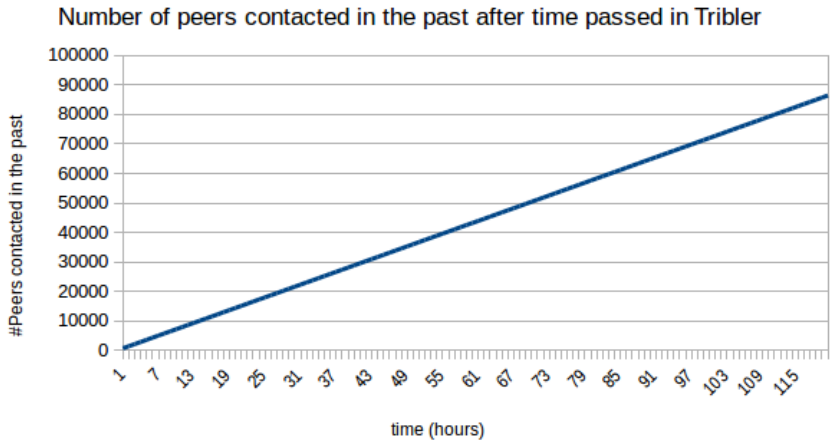

Figure 3.4: The figure shows the number of peers that have been contacted in the past by a peer and have been in the neighbouring list of that peer as time proceeds. A new peer is added to and removed from the neighbouring list every 5 seconds. After 120 hours 86400 peers have been added to the neighbouring list.

network with transaction T' but sends an confirmation of T to the merchant. Because the merchant is eclipsed he can never tell the network about T. When the attacker is the customer he can rewire the money back to himself with T' and thus not pay for the goods. This attack has happened in a real world situation.

5) N-confirmation double spend In a system with an N-confirmation transaction the attacker can also double spend coins from a merchant with an N-confirmation double-spending attack. In an N-confirmation transaction the merchant only releases goods after the transaction is confirmed in a block of depth N - 1 in the block-chain. The attack requires that not only the merchant is eclipsed, but also a certain fraction of miners. The attacker receives a transaction T from the eclipsed merchant and send T only to the eclipsed miners. The eclipsed miners incorporate T into their view of the block-chain V'. The confirmation of T from the eclipsed miners is send to the merchant who releases the goods to the attacker. After this has happened, the block-chain view V of the non-eclipsed miners is send toward the merchant and the eclipsed miners. Next, the block-chain view V' containing T is orphaned, and the attacker acquired goods without paying. [30]

3.5. HANDLING CHURN

In the context of P2P systems, churn is the dynamics of peer participation in the network. The arrival and leaving of peers. [31] Peer selection is designed in such a way that around every 5 seconds a peer arrives and leaves in the neighbouring list of every peer. A peer leaving from the neighbourhood list means that the connection between that peer is broken and the peer cannot be contacted anymore for latency information. The arriving peer in the neighbourhood list adds latency information to the current peer. he latency

information could be latency's measure from the leaving peer toward other peers and the latency measured from the current peer to the leaving peer. The implication of the current peer selection mechanism is that new latency information of other peers arrives every 5 seconds while throughout the run of Tribler the neighbouring list maintains to be the size of around 11 peers. Figure 3.4 shows the expansion of the numbers of peers that have been in contacted in the past and in the neighbouring list as time proceeds. The latency information of peers contacted in the past can be stored and reused again for the latency estimation algorithm. When storing latency information of peers contacted in the past the latency information could become outdated and less reliable. The latency overlay should take the measurement date of latency information into account and delete too old information. Old latency information can also give pressure to the memory and should at some point be deleted.

4

System Design

To make the latency estimation algorithms computationally and memory efficient we will work with incremental algorithms. In this chapter we will focus on how we designed the low latency overlay. The overlay will try to get the average latency towards peers in the neighbourhood of a peer as low as possible. The overlay will have to calculate and estimate the latency's between peers to know what peers to introduce to other peers and what peers to walk to and add to the neighbouring list. The estimation of latency's is done with algorithms that are inspired by the GNP algorithm and the other algorithms described in Chapter 2. A comparison of the different algorithms we propose in this chapter is done with experiments described in chapter 5. How the new algorithms are embedded in Tribler such that all design criteria of the previous peer discovery mechanism still hold is also described in this chapter.

4.1. Low Latency node selection

The low latency overlay does not always have to perfectly introduce the peer that has the lowest latency toward the peer that did the introduction-request. As long as the accuracy of the overall algorithm is still good. The low latency selection also has to be incorporated in the current node selection process in such a way that there is still protection against the eclipse attack and NAT-timeouts. In the current node selection, the use of the groups have a dampening effect on the eclipse attack. Next to that, the oldest node is currently selected from the groups to prevent NAT-timeouts.

In the new node selection policy for the overlay, the different groups described in chapter 3 maintain and the node with the lowest latency is selected from each group. Because the group structure maintains the new overlay still protects against the eclipse attack. Nodes stay for such a short time in the neighbouring list that NAT-timeouts also do not become a problem. Introduced nodes are removed after 25 seconds and nodes that are send an introduction-request or where introduction-requests were received from are removed after 55 seconds from the neighbouring list.

To give a good overview the new node selection policy for an introduction request is the following. Dispersy node divides a nodes candidate list into three categories:

I) Trusted nodes
II) Nodes we have successfully contacted in the past
III) Nodes who have contacted us in the past, either through.
a) Nodes that have sent an introduction-request; or
b) Nodes that have been introduced to another node.

When selecting a node to add to the neighbouring list, a node will choose from its candidate list with pre-defined probabilities. The trusted node category has a probability of 1%, 49.5% is determined by category II and category IIIa and IIIb both get 24.75%. Instead of choosing the node to which the selecting node has had the most recent interaction with, a random node from the top 4 lowest latency's is chosen from each category. The lowest latency cannot be chosen from each category because this will result in a repeated selection of that node once the algorithm is converged. To load balance the node selection among other low latency peers, one of the peers in the top 4 lowest latency's of each category are chosen.

The peer C to introduce to peer A, who sended the introduction request, is simply a peer from the neighbouring list of peer B to which peer A has the lowest latency. Peer A, B, C are the peers from the peer discovery mechanism description in Chapter 3. Only a peer from the neighbouring list of peer B can be introduced because the NAT firewall of peer C has to be punctured to let peer A connect to it. Peer B can only send a puncture request to peer C if peer C is a neighbour because otherwise there wouldn't be a stable connection between peer B and peer C.

4.2. RETRIEVING LATENCY INFORMATION WITH DISPERSY MESSAGES

Dispersy needs to collect latency information in order for latency algorithms to work. In addition to that dispersy also crawls latency's from other peers to retrieve the latency's other peers have with each other. The dispersy message cells collect latency information and stores them into memory blocks in the RAM.

4.2.1. PING TO NEIGHBOURING PEERS

Every peer send all peers in its neighbouring list a ping message every PING TIME INTERVAL seconds. This means the interval time between pinging peers is a fixed ping time interval setting equal to some constant plus a random variable drawn from a uniformly distributed distribution between 0 and 3 seconds. The standard setting for PING TIME INTERVAL is 3 seconds.

For every ping message send toward a neighbouring peer the current time is stored to compare with the arrival time of the response time and calculate a latency. The ping payload contains the ip and port of the peer sending the ping message. Also the time at which the message is send is added to the payload. The payload format for the ping message is shown in figure 4.1.

After receiving a ping message, the peer checks whether it not already received a ping message from the ip, port and time combination given in the payload of the received ping message. If the message is not yet received a pong response is given to the ip and

IP address	Port	Time

Figure 4.1: Ping and Pong payload.

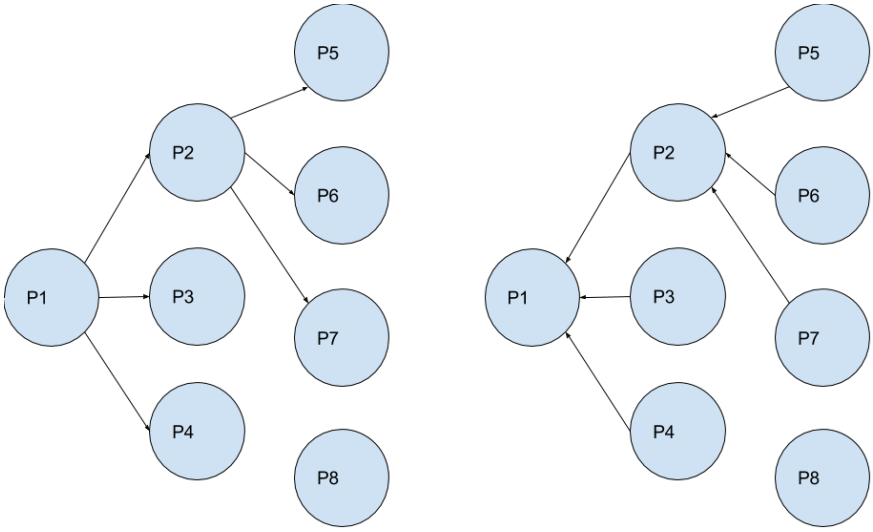

Figure 4.2: The left figure shows what happens when P1 sends a crawl request. The crawl request is forwarded to its neighbours P2, P3 and P4. These neighbours forward the crawl request to their neighbours to reach a maximum number of neighbours. In the right figure the latency response message is shown. All peers send back their latency information to the peer from who they received the crawl request message. These peers forward the latency response message back until the original crawler P1 is reached. In the example P5, P6 and P7 send their latency information to P2 who forwards the latency information to P1.

port combination received from the ping message payload. The pong payload contains the ip and port of the peer that received the ping message and is given a response and contains the same time as received in the ping message. See figure 4.1 for the payload byte format of the pong message.

4.2.2. CRAWL FOR LATENCY INFORMATION

A crawling mechanism is active on every peer to collect measured latency's from other peers that were collected with ping and pong messages. A crawl message is send repetitively in an interval. Every CRAWL TIME INTERVAL seconds a crawl request is send by each peer to every peer in its neighbouring list. The standard CRAWL TIME INTERVAL is 15 seconds. Each peer that receives a crawl request message forwards this message to other peers and send its latency's back toward the requesting peer with a latency response message. By forwarding the latency request message more peers are reached that send back latency information.

When a peer returns latency information as a reply to a latency request message it sends this latency information back to the peer who send the request. When the request message was forwarded the latency response message is also forwarded back to the peer who send the request until the original crawler is reached. As peers can only contact

other peers in their neighbouring list the forwarding construction is necessary. Peers cannot directly send back the latency information to the initiator of the crawl because there is no reliable connection between these peers and the crawl initiator. A reliable connection cannot be set up because the NAT firewall should first be punctured with peer discovery. An overview of the forwarding mechanism is shown in figure 4.2. In a later paragraph we will explain how the forwarding mechanism is programmed.

An overview of the latency request payload is shown in figure 4.3. The IP address and port of the peer requesting the crawl is stored in the message. The hop count variable denotes how many times the message has been forwarded. The peer that sends the first crawl message sets the hop variable to 0. The relay list contains a list of unique variables that is used by the response latency message to know to which peer the latency response should be forwarded back. The hop variable is increased each time the message is forwarded. If the hop count exceeds the MAXIMUM HOP COUNT variable the message is not forwarded anymore.

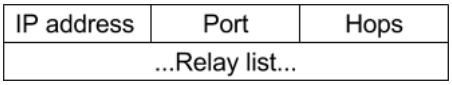

Figure 4.3: Overview of crawl request message.

The latency response message payload is shown in figure 4.4. The IP address and port contain the address of the peer giving the latency response message. The relay list is used by the mechanism to forward latency response messages back toward the peer that originally send the crawler request. The latency's in the payload are all the latency's that are send backward toward the original crawler. The latency's are stored in a dictionary with the two addresses of one latency as key and the latency between these two addresses as the answer to that key. The dictionary is serialized to a string to easily transfer them in the payload.

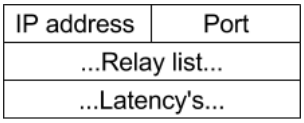

Figure 4.4: Overview of latency response message.

THE FORWARDING MECHANISM

We will further explain how the forwarding mechanism works in the code. Crawl messages are forwarded by peers to reach more peers that can return latency's. The returned latency's are send back to the original requester with the same route as the requests were send but then backward. Both the crawl request message and latency response message contain a relay list that is used in the forwarding mechanism.

In the first part of the mechanism the crawl request messages are forwarded to other peers as can be shown in figure 4.5. Each time the message is forwarded a unique $relay\_id$ is created by the peer and is added to the relay list. When a peer receives a crawl request

message the address of the sender is saved in the *relay* dictionary that is maintained by the peer. The last *relay_id* on the relay list in the message is used as a key in the *relays* dictionary. With the *relay_id* as key the peer can know to which address the latency response has to be send back in the second part. The unique *relay_id* is created using the global time variable in dispersy plus the address of the peer creating the unique id. The global time variable is a lamport clock used for message ordering inside a dispersy community. With global time each message used in the community can be uniquely identified with in combination with the member who send the message and the community itself. The combination of global time and address thus gives a unique identity variable. The *relay_id* has to be unique to make the response always arrive at the right peer. If *relay_id* is not unique the key in the *relay* dictionary might be overwritten and the response message could arrive at another peer.

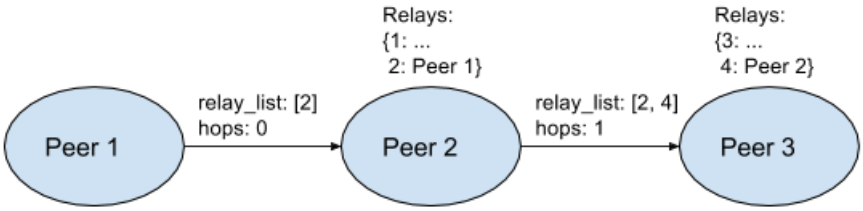

Figure 4.5: Schematic scheme of peer forwarding. In each communication line the *relay_list* is given. Each peer adds a new relay id to the *relay_list*. When a peer receives a message the *relays* dictionary is updated with the last added *relay_id* as key and the peer who send the message as result. The hop count is also increased at each forward.

In the second part of the mechanism the latency responses are send backward to the peer that initiated the crawl. An overview of this mechanism is shown in figure 4.6. At each arrival of a latency response message the last *relay_id* of the *relay_list* in the message is popped of the list and used as a key in the *relay* dictionary. As can be shown in figure 4.6 the key gives the address back of the next peer in the forwarding chain to eventually end at the crawl initiator. The dictionary key is also deleted as the latency response is forwarded back and the key is of no more use. By deleting the dictionary key the crawl mechanism stays memory efficient.

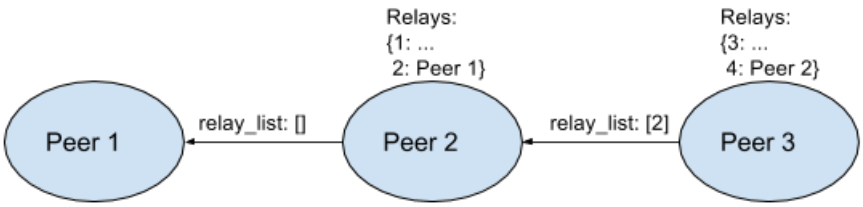

Figure 4.6: Schematic scheme of peer forwarding mechanism upon return. When the hop count exceeds the hop count limit the latency's are returned. The peer pops the last *relay_id* from the *relay_list* and uses this id to lookup the peer to backward the latency's to in the *relays* dictionary.

Sometimes the peer to which the latency response has to forwarded back is no more in the neighbouring list of a peer. In that case the latency response simply cannot be forwarded anymore and the crawl initiator will never retrieve the latency's. But, as the latency crawler is activated in an interval the crawl initiator will eventually maybe retrieve the latency's of the peer that left the neighbouring list.

4.3. INCREMENTAL ALGORITHMS

We focus on online incremental algorithms to predict the latency's to get a computationally and memory efficient solution. A schematic view of an online incremental algorithm is given in figure 4.7. An online incremental algorithm does not require the total input of all the measured latency's at once but instead the input is given over time. At each new time point when a new input vector is given to the algorithm new solutions are calculated. When new information is added to the incremental algorithm a solution is immediately calculated. The new information updates the solution in such a way that when all information is eventually fed to the algorithm a final solution is calculated. Calculating a new solution upon new information is called a step in the incremental algorithm and does not require so much computational power. An online incremental algorithm differs from a normal incremental algorithm in that there is no knowledge on future input, while with normal incremental algorithms there is complete knowledge and decisions can be made with future input knowledge. [32] [33]

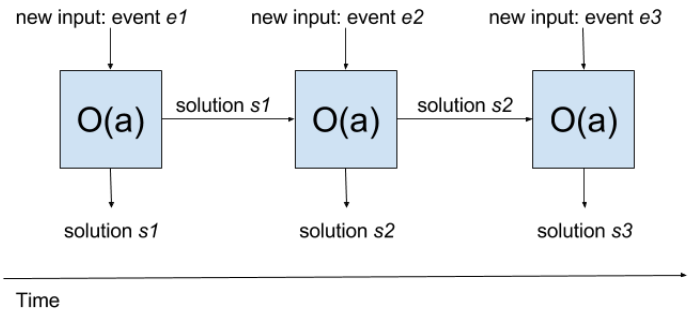

Figure 4.7: Overview of an online incremental algorithm. At each step a new input event e is added to the algorithm. A small computation with $O(a)$ complexity is used to calculate a new solution s. The new solution is used in the next step of the algorithm.

By taking steps the incremental algorithm chops the problem into pieces that are easy to compute and do not block the processor of a peer. This is very important in Tribler because peers cannot block the system. If that happens, the latency's between peers will increase because peers will wait for the processor to finish its computation. However, chopping the computation into pieces can come at the cost of the accuracy of the latency estimation. At each step future information input cannot be taken into account in the calculation of the new solution. The relation between the new information and future information added to the incremental algorithm can only be analyzed by looking at information that was added in a past step. Information that was added in the current step is future information from the perspective of the information added from

the past. To what extent computational time can be spend at looking at information that was added in the past to increase the accuracy is explored in the experimental chapter with various incremental algorithms we will describe in the next paragraph.

4.3.1. EMBEDDING INCREMENTAL ALGORITHMS INTO TRIBLER

A new Incremental Algorithm step is run in a time interval equal to COORDINATE TIME INTERVAL + $uniform(0,3)$ seconds where $uniform(0,3)$ is a draw from a uniform distribution. New information is added to the algorithm when peers enter the neighbourhood. To retrieve the new information the new entering peer should first be send a crawl request to retrieve the new latency information. Once in a time interval neighbours of a peer are crawled. The incremental algorithm has to wait until this crawl is finished. The new low latency overlay maintains a list P_{new} of peers that entered the neighbourhood and also maintains whether the peers of P_{new} were already crawled. P_{new} is used in the latency estimation algorithms.

4.4. LATENCY ESTIMATION ALGORITHMS

In order to discover what are the best practices for the low latency overlay a number of algorithms are implemented inspired on the algorithms in the literature.

4.4.1. NAIVE ALGORITHM

The first algorithm is a naive coordinate-based algorithm where an error function is minimized that is equal to the difference between the estimated latency's based on coordinates in a geometric space and real measured latency's. It assumes that there are N hosts in the system and it further assumes that hosts H are coordinates in a 2 dimensional geometric space S. Every host $H_n \in H$ has its own coordinate C_n^S in S. Because S is geometric the distance function between two host coordinates $d(C_1^S, C_2^S)$ is easily calculated by taking the euclidean distance between the two hosts H_1, H_2. The error function requires that latency's are measured and collected by hosts. The resulting crawled latency's give the measured distance between two hosts. The function $md(H_1, H_2)$ is equal to the measured latency between hosts $H_1 \in H$ and $H_2 \in H$.

The following minimization function is calculated to compute the coordinates of nodes:

$$f_{obj}(C_1^S, ..., C_N^S) = \sum_{C_i, C_j \in \{C_1, ..., C_N\}, H_i, H_j \in \{H_1, ..., H_N\} | i > j} = \epsilon(d(C_1^S, C_2^S), md(H_1, H_2)$$

where $\epsilon(.)$ is the error measurement function:

$$\epsilon(coordinate\_distance, latency\_distance) = (coordinate\_distance - latency\_distance)^2$$

The minimization function used is BFGS. This algorithms allows to minimize to the error function with less minimization steps while remaining a good performance of minimization. The reason this algorithm is chosen is explained in the experimental section. BFGS can vary in the number of function calls it requires. With more function calls the BFGS might have a better minimization performance, but the computation becomes more expensive. The complexity of BFGS is $O(m * error)$ where m is the number of error function calls and $error$ is the complexity of the error function.

The complexity of the error function is $O(N^2)$. Because the number of error function

calls is negligible the total complexity of the algorithm is $O(N^2)$. Tribler is not able to distinguish between landmark and non-landmark nodes as in the GNP algorithm. Therefore, no computational efficiency's based on central components such as in the GNP algorithm can be applied. Because every pair of coordinates and their representing Hosts are added in the sum function the complexity of one sum function is $O(N^2)$. There is a squared relationship between the number of peers N and the efficiency of the algorithm. With large N the algorithm can become too computationally expensive. In large P2P networks, N can easily become around 100000 nodes. In the experimental section we explore how fast with increasing N the naive algorithm becomes computationally too expensive.

4.4.2. Simple Incremental Algorithm

The simple incremental algorithm only updates the coordinates of new entered peers P_{new} to the neighbourhood. In the experimental section we call this algorithm "Inc". It is similar to the Naive Algorithm in that there is also a 2 dimensional geometric space S where every host $H_n \in H$ has its own coordinate $C_n^S \in C$. In the text hosts are sometimes called peers, they have the same meaning. The distance functions are also $md(H_z, H_b)$ for the measured latency between two hosts a and b and $d(C_a^S, C_n^S)$ for the euclidean distance between the two coordinates representing hosts a and b. In all other incremental algorithms described in this section these assumptions apply. The way the coordinates are calculated is however different in each algorithm.

In the simple incremental algorithm "Inc", only the coordinates C_a^S of each peer in P_{new} is updated by minimizing its error function. Peers measure the latency's toward their neighbours and remember the latency's measured toward past neighbours. A subset L from the crawled latency's is taken that are all the latency's between peer a and the neighbours and past neighbours of a. For each latency $l \in L$ there are two peers p_1 and p_2 which are the peers where the latency l is measured between. The collection of all these peers minus peer a we call P_{sub} with coordinates C_{sub}. Because the latency's in L are all the latency's measured between peer a and its neighbours and past neighbours, C_{sub} are therefore all the coordinates of neighbours and past neighbours of peer a. For each of the peers $p_n \in P_{sub}$ the coordinate $C_n^S \in C_{sub}$ is retrieved or created. Whenever there is a new unknown peer $p_n \in P_{sub}$ which has not yet have coordinates in C_{sub} its initial coordinates $C_n^S \in C$ are created randomly by taking two draws from a uniform distribution function from 0 to 1. All coordinates that are created in the past by the peer who executes the algorithm are called C. After that the coordinate $C_a^S \in C$ of the new entering peer a is calculated by minimizing the following function:

$$Inc_{obj}(C_a^S) = \sum_{C_i^S \in C_{sub}} \epsilon(d(C_a^S, C_i^S), md(H_a, H_i))$$

where $\epsilon(.)$ is the error measurement function:

$$\epsilon(coordinate\_distance, latency\_distance) = (coordinate\_distance - latency\_distance)^2$$

The minimization is done with the BFGS algorithm like as in the naive algorithm. The complexity of one minimization function call is $O(|L|)$ where $|L|$ is the size of the number of latency's measured by one peer. $|L|$ becomes larger as time progresses as peers have had more neighbours and thus more latency's measured towards neighbours. The minimization function is called for each peer in P_{new} for one step of the "Inc" algorithm.

However, the size of P_{new} is negligible so the total complexity of one step in the "Inc" algorithm is $O(|L|)$.

4.4.3. INCREMENTAL ALGORITHM WITH R RANDOM REPEAT

The Incremental algorithm with R random repeat extends the "Inc" algorithm by also updating the coordinates of other peers than the new entering peers P_{new}. We call this algorithm in other section "RandomRepeat". In each step after the "Inc" algorithm is run, R random coordinates $(C_1^S, C_2^S, C_j...^S, C_R^S) \in C$. are updated with a similar minimization function as the minimization C_a^S in the "Inc" algorithm. All coordinates that are created in the past by the peer who executes the algorithm are called C. The minimization function that is called for each of the R randomly chosen coordinates is equal to the minimization function of "Inc". The "RandomRepeat" extension is:

for each $C_j^S \in (C_1^S, C_2^S, C_j...^S, C_R^S)$ do

$$Inc_{obj}(C_j^S) = \sum_{C_i^S \in C_{j_{sub}}^S} \epsilon(d(C_j^S, C_i^S), md(H_j, H_i))$$

where $\epsilon(.)$ is the error measurement function:

$$\epsilon(coordinate\_distance, latency\_distance) = (coordinate\_distance - latency\_distance)^2$$

The subset of coordinates $C_{j_{sub}}^S$ is calculated in the same way as in the "Inc" algorithm by taking a subset of latency's L_j from the crawled latency's. L_j is equal to all the latency's between peer $H_j \in H$ and the neighbours and past neighbours of peer $H_j \in H$.

The total number of times the minimization function is called is $R+1$ times. The function is called R times extra for the extension and once called for the "Inc" algorithm. The complexity of the algorithm is thus $O((R+1)*|L|)$. In the experimental section we will test with various numbers of R to see its impact on the computation time and accuracy. It will be most likely that a larger R will increase the accuracy but lower the computation time. A good design choice for R will depend on the results of these experiments.

4.4.4. INCREMENTAL ALGORITHM WITH R FIXED REPEAT

With a random repeat of node updates some nodes are updated more frequently than others. A structured repeat of coordinate updates of other nodes is implemented to further improve the accuracy of the R random repeat algorithm. We call this algorithm "Repeat" later in this document. The structured repeat ensures that all coordinates C are updated once before the same node is updated again. In this way no nodes are left behind in updating and no nodes are updated more frequently than other nodes. The "Repeat" algorithm is implemented by numbering each coordinate of C. When C increases the new coordinates are given a new number incrementally. So the first coordinate that was put in C is given the number 1, the second the number 2 and so on. Each time the "Repeat" algorithm is executed, a new subset of R nodes of C is selected for updating. Thus the first time the coordinates with a number smaller than R are selected from C, the second time the coordinates with a number between R and $2R$ are selected etc. If after n times $nR > |C|$, the selection starts again from the beginning at the low numbers of C. The complexity of this algorithm is the same as the R random repeat version because again the coordinates of R nodes are updated with the same minimization

function. Thus the complexity is $O((R+1)*|L|)$.

4.4.5. INCREMENTAL ALGORITHM WITH R FIXED REPEAT AND TRIANGLE INEQUALITY VIOLATION PREVENTION

The Incremental Algorithm with R fixed repeat and Triangle Inequality Violation (TIV) Prevention is an extension on the "Repeat" algorithm. In further sections we call this algorithm "TIV". The problem of Triangle Inequality Violations is solved by ignoring peers who are estimated to contribute to a TIV. Ignoring means that the coordinates and latency's towards these peers are ignored in the minimization functions of both the "Inc" part of the algorithm and the "Repeat" part of the algorithm. To estimate what latency's contributed to TIV's the "prediction error" is calculated for every latency that is measured in the past by the peer executing the algorithm. The prediction error is equal to the euclidean distance between the coordinates of the peer pair in the latency divided by the latency. So for every latency $l \in L$ and peer pair H_1, H_2 of l the following prediction error is calculated:

$$prediction\_error = \frac{d(C_1^S, C_2^S)}{md(H_1, H_2)}$$

The three latency's with the largest prediction error are ignored and not used in minimization calculations. The sorting of the latency's according to prediction error has as complexity $O(Llog(L))$. The total complexity of the algorithm becomes $O(L^2 * log(L) * R)$.

4.5. HIGH QUALITY OVERLAY

In this section we describe the optimization's made to the low latency overlay to make it ready for the real world. The optimization's provide low computational time and low bandwidth usage.

Because the low latency overlay has a low latency bias, only peers that are close to the calculating peer are important and only latency's toward these peers are remembered. For every peer in the system only 100 latency's toward the top 100 closest peers to the calculating peer are remembered. Some algorithms update the coordinates over time and require that latency information received from other peers is remembered and stored in the memory of each peer. Only the latency's toward the closest peers are important to the algorithm because we are only interested in the closest peers and the coordinate positions need to be optimized toward the closest peers. There is explicitly chosen for a top 100 of closest peers because this gives good memory efficiency.

It gives computational advantage that the coordinate of a peer only has to be minimized toward 100 peers. This is the case because only a top 100 of latency's toward close peers is stored in the memory.

The repeated coordinate updating in the algorithms is spread in multiple times to save computational speed. For instance, if 5 coordinates need to be updated before a new entering peer coordinates are calculated, the minimization function is called 5 times with pauses of non computation. In these pauses Tribler can do other stuff and the maximum computation is the optimization of only one minimization function and thus stays the maximum computation time low.

To save bandwidth is the forwarding mechanism and crawling for latency's disabled and are latency's instead piggybacked with the pong message. A peer piggybacks all latency's known toward other peers in the pong message. Because the ping message is send frequently and the low latency overlay is built in such a way that new peers are also added to the neighbouring list, a peer will eventually receive latency's from many different peers. The new pong message payload with latency's piggybacked can be seen in Figure 4.8.

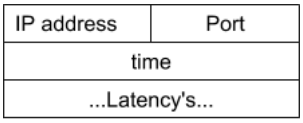

Figure 4.8: Pong payload for high quality overlay with latency's piggybacked.

5

EXPERIMENTS

In the experimental section we do a local experiment to test the different algorithms and two experiments in a decentralized Tribler setting with 30 and 500 nodes to test the algorithms in a real world application.

5.1. INCREMENTAL ALGORITHM

In this section, we describe the performance metrics used to measure the performance of the incremental algorithm and discuss the experimental results.

5.1.1. PERFORMANCE METRICS

COMPUTATIONAL TIME

The computational time metric shows how much time Tribler is computing something and is blocked. If the blocking time becomes too high, Tribler does not function anymore. To fully evaluate the performance of the incremental algorithm the trade-off between the computational time and the accuracy of the algorithm needs to be explored. The computation of incremental algorithms is divided over time. Every time a peer explores a new neighbour peer a new data vector containing the latency's measured by the newly explored peer is added to the latency data-set of the exploring peer. The computational time it takes to process this new data vector can easily be measured by taking the time difference of the time before and after the computation. The accuracy change after each incremental step of the algorithm is harder to measure and requires specifically designed metrics. In the experiments in the decentralized Tribler setting the high quality overlay spreads the computation of the repeated updates of coordinates over time to further enhance the quality of the overlay. The code for the repeated update is the same and is therefore added to the computational time measurement as a separate measurement in time.

We use two metrics to measure the accuracy performance of the algorithm: ranking accuracy and relative error.

RANKING ACCURACY

How good the algorithm has neighbours in his neighbourhood that are the peers to which the latency is the lowest is measured by ranking accuracy. The idea is that after each incremental step we can calculate the predicted distances between neighbouring peers and we know the real distances based on the measured latency's towards neighbours and past neighbours. We then sort the predicted distances and measured distances towards neighbours and past neighbours to calculate a top closest peers for both the predicted distances and measured distances. The ranking accuracy is defined as the percentage of peers that is both in the top list of predicted closest peers and in the top list of the measured closest peers. If the accuracy is 50% accurate then 50% of the peers of the predicted closest peers list are also in the top measured closest peers list. The size of both top lists is equal to 10% of the total number of peers that were neighbours or past neighbours of the peer. The list size thus increases as new neighbours enter the neighbourhood of a peer. This is done to give compensation for the increasing number of latency's measured.

RELATIVE ERROR

The relative error metric measures how well a predicted distance matches the corresponding measured distance. This metric is also used to measure the performance of the GNP algorithm [10]. For each predicted distance that can be calculated between two peers the relative error is defined as follows:

$$relative\_error = \frac{|predicteddistance - measureddistance|}{min(predicteddistance, measureddistance)}$$

A value of zero implies a perfect prediction as then the predicted distance and measured distance are equal. A value of one implies the predicted distance is larger by a factor of two. The relative error metric measures the overall predictive performance of the algorithm while ranking accuracy is a good metric to evaluate the selective performance of low latency peers of the algorithm. Both metrics do not necessarily imply each other. A good selective performance might have a bad relative error and vice verse.

5.1.2. RESULTS OF LOCAL EXPLORATION OF ALGORITHMS

The algorithms described in section 5 have been implemented and tested on one computer with complete information. The computer runs a dual core 2.8 GHz processor. With complete information we mean all peers know all latency's to each other. Thus, if the swarm size is n peers large, a single peer a knows $n-1$ latency's to all the other peers. With complete information the algorithms should run as optimal as possible.

In the experiment the location based latency estimation algorithms are tested on an increasing number of peers that are added to the problem. A new peer entering the system has the same effect as a new peer entering the neighbourhood of the low latency overlay. In the beginning of the algorithm there are 0 peers in the system. The first and second peer entering the system only know the latency toward each other. Every time a new peer is added to the system an incremental step of the algorithm is taken. It is assumed that the new entering peer has already measured all latency's toward all other peers that have been added to the system in the past. Thus all latency's towards neigh-

bours and past neighbours have already been measured. The new entering peer thus has complete information about the latency's. In Tribler the peer first has to measure all the latency's toward other peers that were added in the past to know this information.

Graphs of the computation time, "Ranking Accuracy" and "Relative error" performance metrics are shown in figures 5.1, 5.2 and 5.3. The computational time of the naive implementation grows exponentially, while the computational time of the incremental algorithms grow linear. If the computation time becomes larger than 0.5 seconds, the computation becomes impractical and will block the application. The application will react later or not react at all to new incoming events reducing user experience and increasing the latency between peers. The incremental algorithms also become impractical with increasing swarm size, in particular the Repeat20 and RepeatStructured algorithm. The RepeatTIV and Inc algorithms have relatively low computation time with also large swarm size. This makes them practical to use from computational time perspective.

The RepeatTIV (or TIV algorithm) algorithm has the best performance while the naive algorithm has the worst performance. The naive algorithm shows a higher score on the "Relative Error" performance metric and a lower score on "Ranking accuracy" compared to the incremental algorithms. This is surprising as it was expected that the naive implementation gives a more accurate performance as more calculative effort is done to get a good performance. The performance of the incremental algorithms are close to each other on both performance metrics. The larger the swarm size the closer the performance of the incremental algorithms are to each other. The RepeatTIV algorithm has a higher "Ranking accuracy" and lower "Relative Error" with a swarm size below 150 peers. In particular the "Ranking Accuracy" differs and is relatively higher for RepeatTIV. Also RepeatStructured has a slightly better performance compared to Repeat20 and Naive for both performance metrics.

5.1.3. EXPLORATION OF DIFFERENT ALGORITHMS IN DECENTRALIZED TRIBLER SETTING

The algorithms are tested in two Tribler experiments: one with 30 Tribler instances (nodes) and another with 500 Tribler instances. With a low number of nodes it is easier for one node to obtain more latency information of the entire network because the entire network is small.

All decentralized experiments are managed by the Distributed ASCI 5 supercomputer (DAS5). Gumby is used to calculate and collect performance metric data. Every 5 seconds the performance metrics "Computational time", "Relative error", "Ranking accuracy" and the average latency towards peers in the neighbourhood are printed towards files maintained by each individual Tribler instance. At the end of the experiment, all metric information is aggregated towards a single file resulting in a collection of performance metric data at each individual timestamp every five seconds. The mean, standard deviation and 95% confidence intervals of the collection of data at each time stamp are calculated at the end of the experiment and printed on a single graph. To every interval in the system is a random number of seconds added to distributed the computation and bandwidth usage over the nodes in the DAS5 supercomputer.

Instead of real measured latency's, latency's are extracted from the King Dataset. [34]

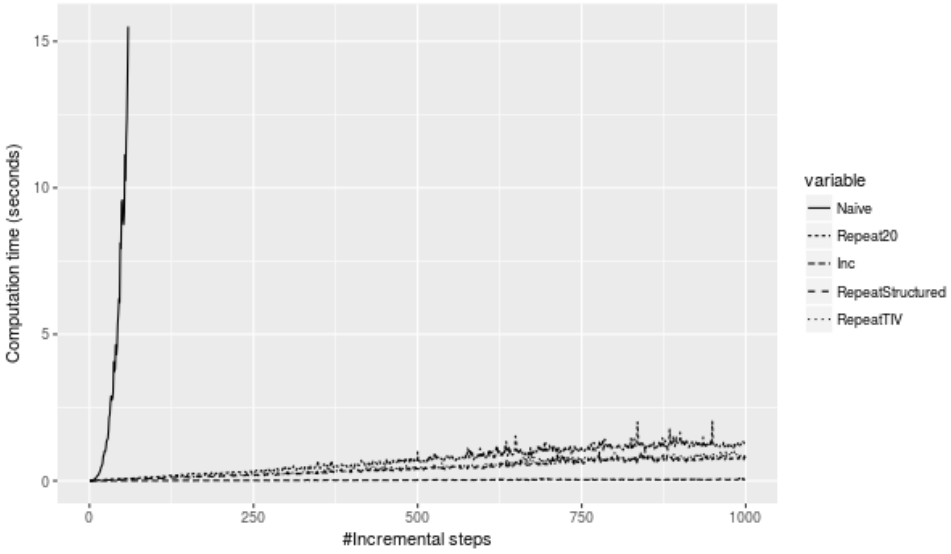

Figure 5.1: Graph of the computation time for every algorithm as the number of peers entering the system increases.

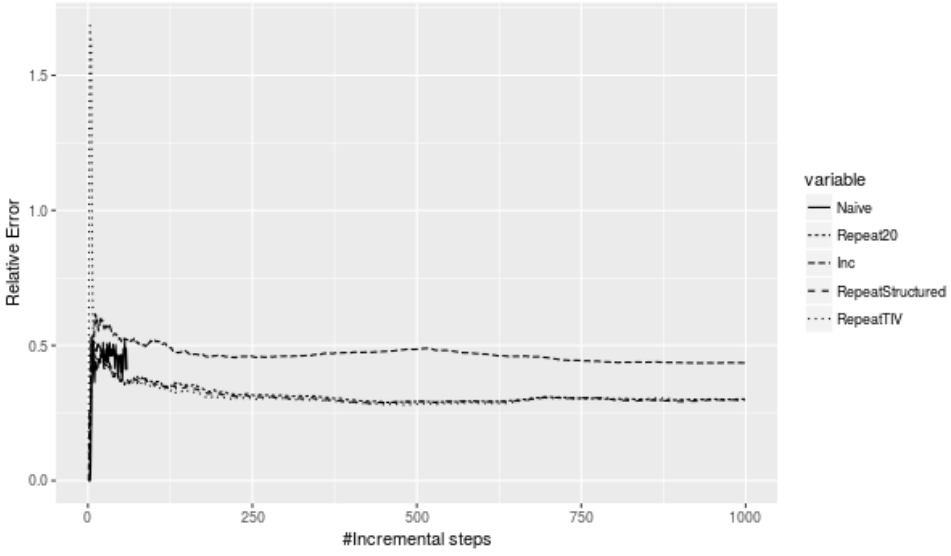

Figure 5.2: Graph of the relative error development for every algorithm as the number of peers entering the system increases.

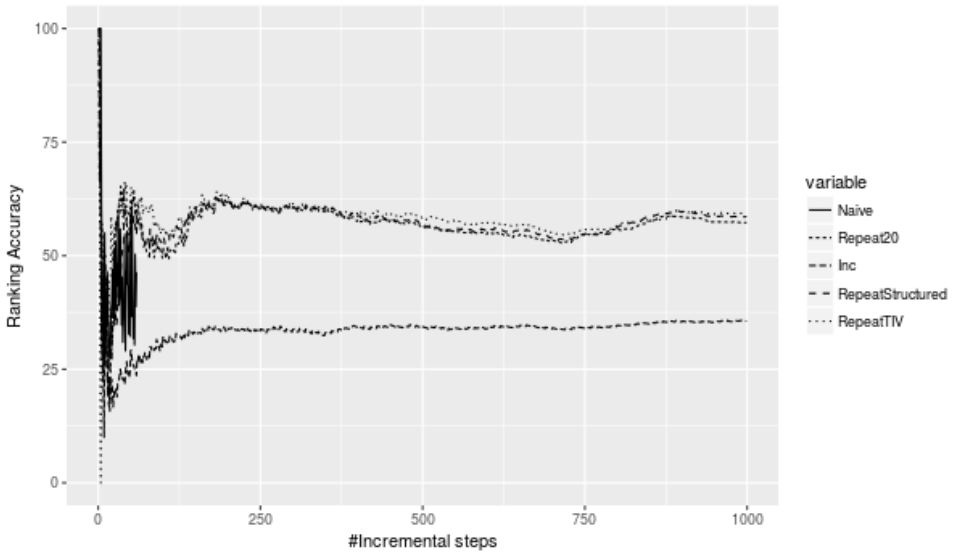

Figure 5.3: Graph of the ranking accuracy development for every algorithm as the number of peers entering the system increases.

Each Tribler instance runs the LatencyCommunity which is responsible for the collection of latency data and the execution of the algorithms. The King Dataset contains a NxN matrix with latency information between two nodes. For instance, the entry on row n and column m contains the latency of a ping from node n to m. This latency is different of the latency measured by node m to n which is presented by the entry of row m and column n. Because the algorithm assumes that there is a single latency between two peers the latency of n towards m and m towards n is averaged. Each Tribler instance is given a unique ID by gumby starting from the number one incrementally increasing upwards. The ID is coupled to the IP, port combination of each tribler instance. A Tribler instance can lookup the ID of another node with the IP, port combination. The ID is used in the matrix to lookup latency's. For instance, if a node with ID k wants to know the latency from itself towards another node with ID l it can lookup the latency in the matrix at row $k-1$ and column $l-1$.

30 NODES EXPERIMENT RESULTS

The goal of the experiment with 30 nodes is to show that the algorithms work in a simple setting. The high quality overlay is used and the experiment only lasts around 5 minutes. The accuracy of the different algorithms is measured with the accuracy measurement variables "Ranking Accuracy" and "Relative Error". The meaning of these variables is described above. The graphs of these accuracy variables and the computation time for an experiment with 30 nodes are shown in Figures 5.4, 5.5 and 5.6.

The Ranking Accuracy variable shows a lot of variation in the beginning of the experiment. This behaviour can be explained with the fact that in the beginning no latency's have been crawled yet. All the positions of the neighbours are put in random position.

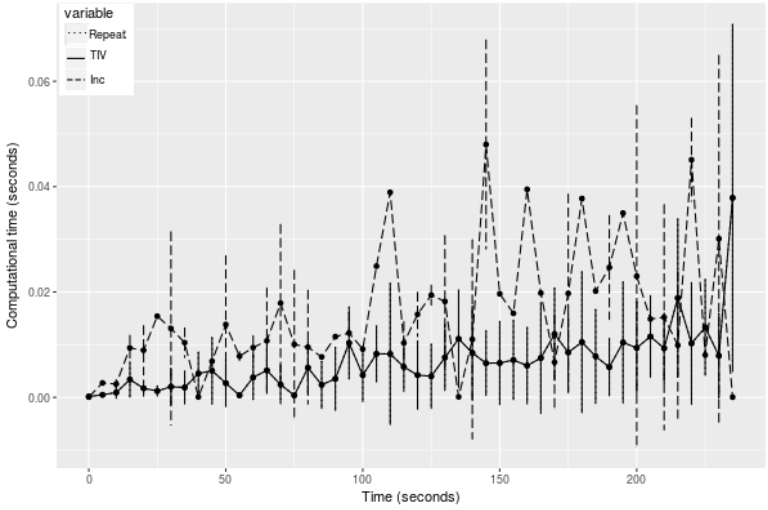

Figure 5.4: The figure shows the computational time for one step of the incremental algorithm as time in the experiment progresses. The number of nodes in the experiment are 30. The dots represent the average computation time over every peer for each different algorithm at a certain point in time. The bar at every dot represents the size of the variance relative to the mean.

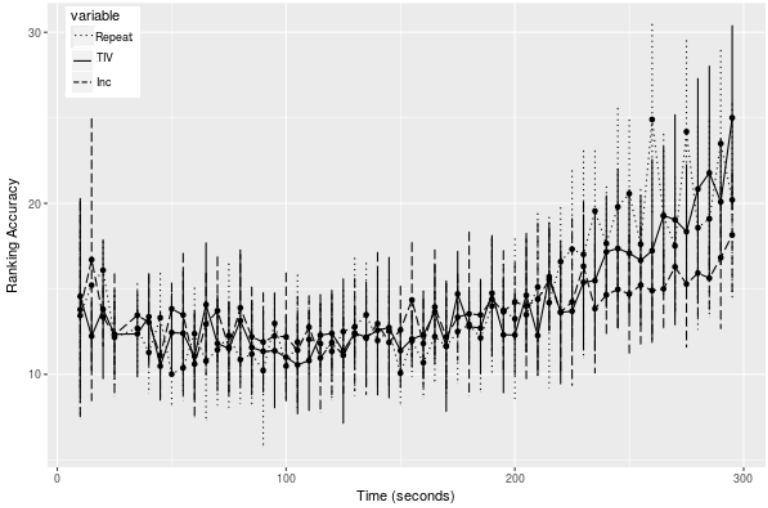

Figure 5.5: The figure shows the ranking accuracy of all peers as time in the experiment progresses. The number of nodes in the experiment are 30. The dots represents the average ranking accuracy over every peer for each different algorithm at a certain point in time. The bar at every dot represents the size of the variance relative to the mean.

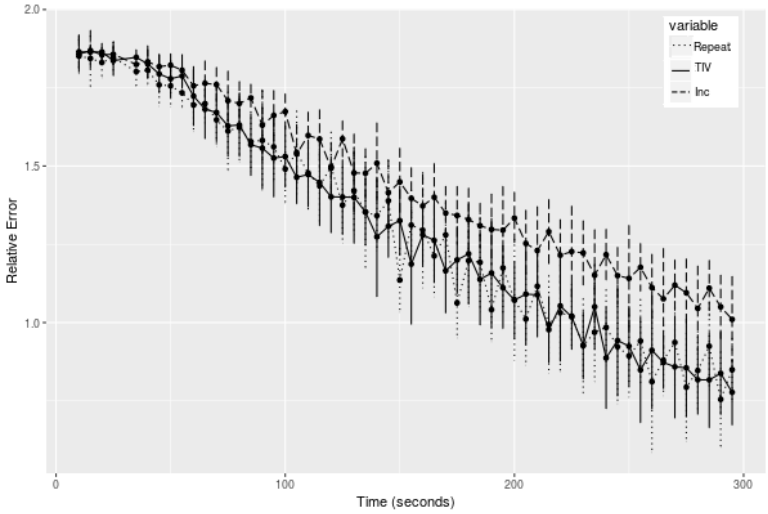

Figure 5.6: The figure shows the relative error of all peers as time in the experiment progresses. The number of nodes in the experiment are 30. The dots represents the average relative error over every peer for each different algorithm at a certain point in time. The bar at every dot represents the size of the variance relative to the mean.

The positioning can either end op lucky or not and this gives the large variation in the ranking accuracy. From around 30 seconds the ranking accuracy increases for all algorithms and the large variation also disappears. The relative error decreases and the variation is gaining as time progresses. This means that each node shows a different run of the algorithms in which variation between the different nodes increases over time.

When comparing the different algorithms it can clearly be seen that the TIV algorithm works best, followed by the Repeated algorithm and following last only the Inc algorithm. The relative error and ranking accuracy of all algorithms decreases but is unable to converge in 5 minutes. The Inc algorithm works slower to a better value of the objective function than the Repeated and TIV algorithm. The TIV and Repeated algorithm have around the same progression towards a better value of the objective function. After 5 minutes the average ranking accuracy of the TIV and Repeated algorithm is around 25% while the ranking accuracy of the Inc algorithm is around 18%. The larger accuracy of TIV can be explained with the fact that peers with large prediction errors are excluded from the minimization process. The positions of these nodes is not improved and the distance function remains large. Nodes with a large distance are not often in the top latency's and thus the ranking accuracy is higher. The Repeated algorithm performs better than the Inc version because nodes that were calculated in the past are updated to get a better performance of the algorithm.

The computational time of all algorithms are below 0,06 seconds. The computational time is higher for the Repeated Algorithm compared to the Inc algorithm because the coordinates of more peers are calculated per step. The TIV algorithm has a bit higher computation because for each peer of which the coordinates are calculated some extra computation to remove triangle inequality violations. In the beginning of the process

the computation time for the Repeated and TIV algorithm is lower. When time passes more latency's are collected and the computation of the coordinates becomes a bit more computationally intensive.

When doing a student t -test on the different algorithms on average latency we get the results as can be seen in the Table below. In all the t-tests used a confidence interval of 0.05 is used with the Welsh t-test. The t-test tests whether the mean difference between two groups can be attributed to random variation or that there is indeed a statistical difference between two groups. The Null hypothesis H_0 states that the difference in mean can be attributed to random variation. If there is a difference which cannot be attributed to random variation the Null hypothesis is rejected. We compare the means of the latency's measured slightly after the beginning of the experiment (between 30 and 40 seconds) and the mean of the latency's measured at the end of the experiment (between 290 and 300 seconds).

All algorithms have only differences that can be attributed to chance. All p-values are higher than 0.05. When comparing the average latency results of the different algorithms combinations all p-values are again higher than 0.05 so there is no statistical difference. The outcomes of the t-test can be seen in the Table below.

Algorithm	Mean 30 < t < 40	Mean 290 < t < 300	p	H_0
Inc	0.06977719	0.06861256	0.4389	True
Repeated	0.07097009	0.06843923	0.1159	True
TIV	0.06678266	0.06848889	0.2594	True

Algorithm	Variance 30 < t < 40	Variance 290 < t < 300
Inc	0.0009966341	0.000966647
Repeated	0.001151103	0.001076714
TIV	0.0009966341	0.000966647

	Mean 290 < t < 300	Mean 290 < t < 300	p	H_0
Inc, Repeated	0.06861256	0.06843923	0.8795	True
Inc, TIV	0.06861256	0.06848889	0.9113	True
Repeated, TIV	0.06843923	0.06848889	0.9645	True

500 NODES EXPERIMENT RESULTS

The goal of the experiment with 500 nodes is to let the high quality overlay converge with a high number of nodes. The experiment with 500 nodes has the same measurement methods as the result with 30 nodes but the results are different. The experiment is executed over a period 12000 seconds which are 3 hours and 20 minutes. We will look at each measurement variable in the following paragraphs. The graphs of the performance metrics and computation time are shown in Figures 5.7, 5.8 and 5.9. The ranking accuracy measures the accuracy of the top 10 latency's in the neighbourhood instead of the top 10% which would result in a top 50 with 500 nodes. This change is implemented to let ranking accuracy be a indicator of how good the algorithm estimates the top 10 low latency neighbours.

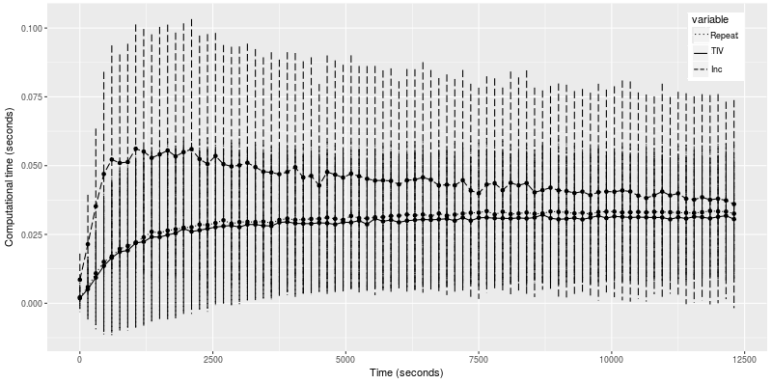

Figure 5.7: The figure shows the computational time for one step of the incremental algorithm as time in the experiment progresses. The number of nodes in the experiment are 500. The dots represent the average computation time over every peer for each different algorithm at a certain point in time. The bar at every dot represents the size of the variance relative to the mean.

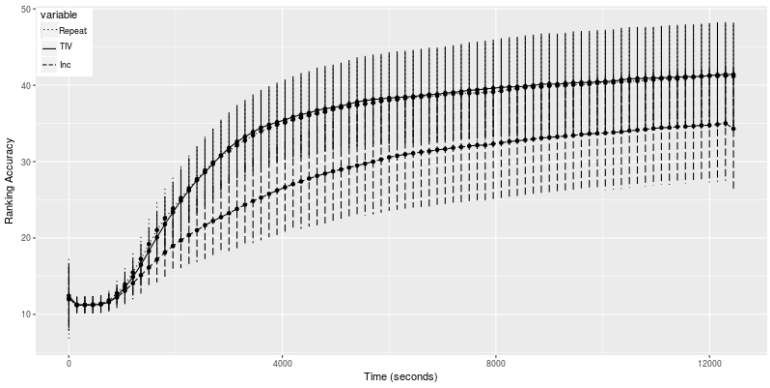

Figure 5.8: The figure shows the ranking accuracy of 490 of the 500 peers as time in the experiment progresses. The number of nodes in the experiment are 500. 10 nodes are added halfway. The dots represents the average ranking accuracy over every peer for each different algorithm at a certain point in time. The bar at every dot represents the size of the variance relative to the mean.

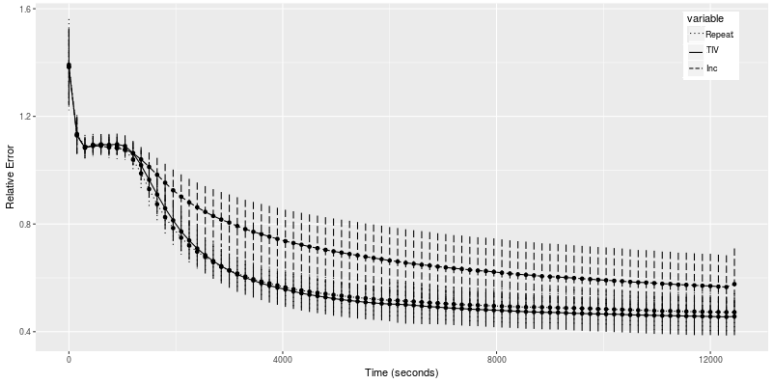

Figure 5.9: The figure shows the ranking accuracy of 490 of the 500 peers as time in the experiment progresses. The number of nodes in the experiment are 500. 10 nodes are added halfway. The dots represent the average relative error over every peer for each different algorithm at a certain point in time. The bar at every dot represents the size of the variance relative to the mean.

The ranking accuracy has the same random period at the beginning of the experiment with a lot of variance. The ranking accuracy converges for the Repeated and TIV algorithm toward 40 %. The Inc algorithm converges with a lower speed toward 35 %. The variance of the accuracy becomes higher as time progresses. What can be seen is that the accuracy still is not converged to its final value while the relative error is for the TIV and Repeated algorithm.

The relative errors do converge to a final value for all algorithms. The TIV and Repeated algorithm have a stronger decline than the INC algorithm. After 3 hours and 20 minutes the TIV and Repeated algorithm have declined toward a relative error of 0.43 with the TIV algorithm a relatively smaller relative error. The INC algorithm decreases slower towards 0.5 in 3 hours and 20 minutes. At the start of the algorithm the relative error decreases fast and then has a small period of around 15 minutes in which it is stable for all algorithms. Then after these 20 minutes all algorithms are decreasing toward convergence. An explanation for the stable phenomena at the beginning could be that the algorithms have to wait until more latency's are measured and coordinate positions can be updated adequately to increase the error objective function.

All algorithms show good results for computation, with all computation times below 0.1 seconds. The Inc algorithm seems to have a larger computation time in the beginning of the experiment and the computation time decreases as time in the experiment progresses. Eventually the computation time converges to an average value around 0.03 seconds for all algorithms. In case of the TIV and Repeated algorithm both the time taken for the original minimization function and the computation time for each repeat have been added to the measurements. The computation of the Inc algorithm eventually converges to the same average value of the TIV and Repeated algorithm. An explanation could be that in the Repeated and TIV algorithms the coordinates are already at a good position and adding a new coordinate or updating old coordinates takes less computation time. The coordinates representing the peers are already at a relatively good position

and require not so much change. This could also explain why the Inc algorithm takes less time to compute as the accuracy of the algorithm is converged and coordinates are at a good position.

T-tests are applied to test this the average latency of the neighbourhoods of all the peers in the experiment. The results can be seen in the Table below. When looking at the results we see that there is a significant difference between the beginning and end of the Inc algorithm. The average latency increases from 0.1815305 to 0.1944564. This is surprising because the latency should be lower. There is no significant result for the Repeated and TIV algorithms and these algorithms have in every previous experiment performed better. The variance is around 0.018 for all algorithms.

Algorithm	Mean 30 < t < 40	Mean 12490 < t < 12500	p	H_0
Inc	0.1815305	0.1944564	2.2e-16	False
Repeated	0.1800181	0.1774035	0.08489	True
TIV	0.1825986	0.1810986	0.3001	True

Algorithm	Variance 30 < t < 40	Variance 12490 < t < 12500
Inc	0.01823406	0.02514181
Repeated	0.01815356	0.01799468
TIV	0.01816127	0.01699357

When comparing the different algorithms for the end of the experiment with t-tests the latency differences are statistically different for all algorithms. The results are that the Repeated and TIV algorithm have a better average result at the end of the experiment compared to the Inc algorithm. This is no surprise as these algorithms also have a higher ranking accuracy and lower relative error. There is also a significant difference between the TIV and Repeated algorithm where the TIV algorithm has a higher average latency. This is surprising as the TIV algorithm seems to have a little better ranking accuracy and lower relative error compared to the Repeated algorithm.

	Mean 12490 < t < 12500	Mean 12490 < t < 12500	p	H_0
Inc, Repeated	0.1944564	0.1774035	2.2e-16	False
Inc, TIV	0.1944564	0.1810986	2.2e-16	False
Repeated, TIV	0.1774035	0.1810986	4.702e-06	False

5.1.4. EXTRA NODES ADDED TO THE EXPERIMENT

The behaviour of new entering nodes is explored in this section. In the 500 nodes experiment, 10 nodes were added to the experiment halfway after 1 hour and 40 minutes. The other 490 nodes are already new convergence with low relative errors and high ranking. The development of the relative error and ranking accuracy of these two nodes can be seen in Figure 5.10 and 5.11.

The convergence time appears to be the same for all algorithms, except for a minor speed enhancement at the beginning. There is no value added in that other nodes have already calculated the coordinates of peers. There also is no difference between

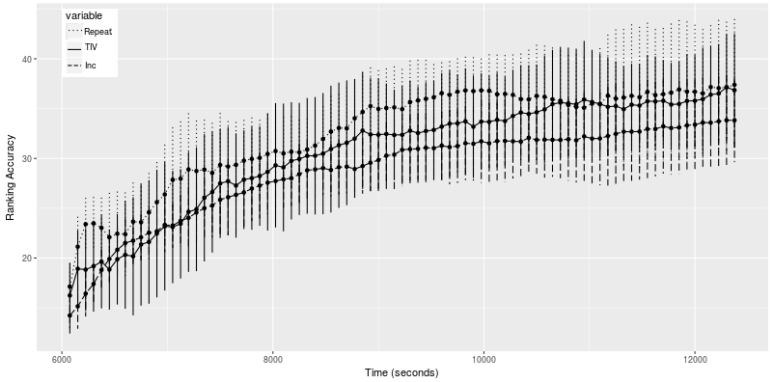

Figure 5.10: The figure shows the ranking accuracy of 10 of the 500 peers as time in the experiment progresses. The number of nodes in the experiment are 500. The 10 nodes in the graph that are added halfway are shown. The dots represents the average ranking accuracy over every peer for each different algorithm at a certain point in time. The bar at every dot represents the size of the variance relative to the mean.

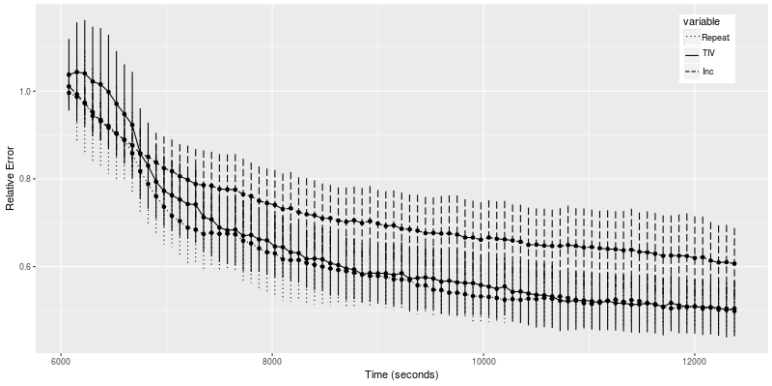

Figure 5.11: The figure shows the relative error of 10 of the 500 peers as time in the experiment progresses. The number of nodes in the experiment are 500. The 10 nodes in the graph that are added halfway are shown. The dots represent the average relative error over every peer for each different algorithm at a certain point in time. The bar at every dot represents the size of the variance relative to the mean.

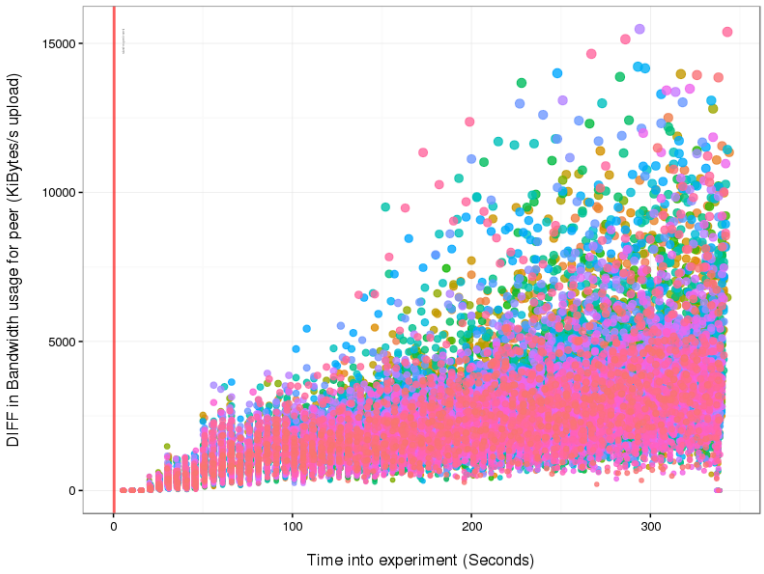

Figure 5.12: Overview of Cost in Bytes of the low latency overlay over time in a 500 node experiment with an overlay that uses the forwarding mechanism for crawling.

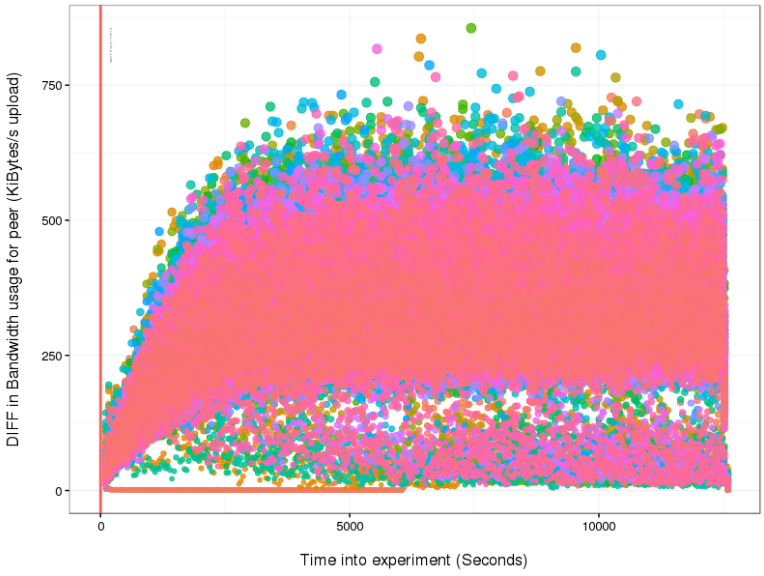

Figure 5.13: Overview of Cost in Bytes of the low latency overlay over time in a 500 node experiment with a high quality overlay.

the different algorithms compared to the normal situation. The Repeated and TIV algorithms convergence a bit faster than the Inc algorithm but it still takes the same amount of time compared to the normal situation. A difference between the normal situation is that the behaviour in the beginning differs. There is no stable period in the beginning where nodes have to wait upon latency collection to further decrease the objective function. With the extra nodes added the objective continually decreases as time progresses until convergence is reached. This gives a small speed enhancement in the beginning compared to the normal situation.

5.1.5. COST IN BYTES

The number of bytes shared in the low-latency overlay increases over time. The low-latency overlay measures and shares latency's with other peers. The byte cost of the communication is therefore the same for every algorithm. In a standard dispersy experiment the total byte cost of the communication is measured for every peer in KiBytes upload. The byte costs for an experiment with 500 nodes have been measured in the overlay with crawling and the high quality overlay. The derivatives of both measurements are given in Figure 5.12 and 5.13. The result is a byte cost increase as time increases for the forwarding mechanism. The reason of this increase is that every peer has more latency's crawled. These latency's are all shared with other peers. The packet size of one latency share has increased and thus also the total byte cost derivative increases toward values over 10000 KiBytes / second. In the high quality overlay the byte cost does not increase over time and remains around 400 KiBytes / second. Only in the beginning it increases as then new latency's are measured over time.

6

Future Work

The current low-latency overlay provides a basic overlay with low computation that uses incremental algorithms to estimate latency's between peers. There are some points of improvement that should be taken into account before P2P applications can rely on the low-latency overlay. The algorithms haven't been tested on large networks. Next to that, triangle inequality violations (TIVs) could be further investigated because the algorithm that tried to counter triangle inequality violations worked the best. Finding better algorithms to improve on TIVs could further increase the accuracy of the latency estimation algorithms.

Experiments with the low latency overlay on large networks should be done to make the low latency overlay applicable in these large networks. It takes several days for a peer to get 100000 different neighbours to measure the latency with. When such large number of latency's have been measured the algorithm should still be computationally and memory efficient. Next it should be tested whether the latency estimation algorithms converge towards a result with high accuracy. The algorithm should also be able to handle large latency inputs with more than 100000 latency's at each step of the incremental algorithm. It is very important that one incremental step is executed computationally efficient. If the computation at one step of the incremental algorithm is too much Tribler could block other processes and therefore increase the latency of the network.

Research on how to counter Triangle Inequality Violations (TIVs) and algorithms that deal with lack of information can further improve the latency estimation algorithms. The experiments so far have shown that TIV prevention proofs to be quite useful. Other possibilities to detect TIVs and prevent them should be further investigated. The accuracy is affected a lot by lack of latency information. The more latency's are measured the better the algorithms performs. The lack of information is especially important in the decentralized Tribler setting because peers only measure latency's toward neighbours and cannot measure latency's to other peers. A solution could be to create an algorithm that can handle lack of information more efficiently. Another solution could be to gain more latency information via other ways. For instance with IMCP ping messages.

7

Conclusion

Building a low latency overlay is not as easy as it seems. Tribler has a number of limiting properties that make it impossible to go for a simple implementation of an existing algorithm. The low-latency overlay should be integrated in the current peer discovery mechanism of Tribler, because the current peer discovery mechanism has a few properties that should be maintained. The current peer discovery mechanism adds randomness in its choices of peer selection to prevent against the eclipse attack. If the new low latency overlay does not have countermeasures against such an eclipse attack, nodes could be controlled by adversaries or nodes could receive false information about other peers or about things from a P2P application.

The integration of the low latency overlay in the current peer discovery mechanism limits the efficiency of the overlay. Peers are getting in and out of the overlay which limits the ability to measure latency's towards peers in the P2P network. Next to that, peers are divided into groups to prevent against the eclipse attack. If a peer wants to introduce a peer to another peer or wants to add a peer to its neighbouring list it cannot choose the peer with the lowest latency, but instead has to choose a peer from one of the groups.

To make good decisions on what peer to introduce to another peer and what peer to add to the neighbouring list the latency's between two arbitrary peers in the P2P network have to be estimated. A peer can only measure latency's toward peers who are inside the neighbourhood. Therefore, a latency estimation algorithm is implemented to make good decisions to get a low latency neighbourhood. There are numerous ways to estimate the latency's between peers. Some of these algorithms work better than the other. It is important to take the computational expense of the algorithms into consideration. When the P2P network is large the latency estimation algorithms can become computational inefficient. Incremental algorithms are used to divide the computation over time. Next to that it is beneficial to counter peers who cause Triangle Inequality Violations (TIVs) in the latency estimation algorithm. Algorithms that deal with TIVs perform better and have a better accuracy, especially in the Tribler environment where latency information sometimes lacks.

7

BIBLIOGRAPHY

[1] Andrew Brook. Evolution and practice: low-latency distributed applications in finance. *Queue*, 13(4):40, 2015.

[2] Ciamac C Moallemi and Mehmet Sağlam. Or forum—the cost of latency in high-frequency trading. *Operations Research*, 61(5):1070–1086, 2013.

[3] Giovanni Cespa and Thierry Foucault. Insiders-outsiders, transparency and the value of the ticker. 2009.

[4] Lawrence R Glosten. Is the electronic open limit order book inevitable? *The Journal of Finance*, 49(4):1127–1161, 1994.

[5] Patrik Sandås. Adverse selection and competitive market making: Empirical evidence from a limit order market. *The review of financial studies*, 14(3):705–734, 2001.

[6] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[7] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[8] YY Yao and Ning Zhong. Potential applications of granular computing in knowledge discovery and data mining. In *Proceedings of World Multiconference on Systemics, Cybernetics and Informatics*, volume 5, pages 573–580, 1999.

[9] Umesh Kumar V Rajasekaran, Malolan Chetlur, Girindra D Sharma, Radharamanan Radhakrishnan, and Philip A Wilsey. Addressing communication latency issues on clusters for fine grained asynchronous applications—a case study. In *International Parallel Processing Symposium*, pages 1145–1162. Springer, 1999.

[10] TS Eugene Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 170–179. IEEE, 2002.

[11] John Ioannidis and Steven M Bellovin. Implementing pushback: Router-based defense against ddos attacks. In *NDSS*, volume 2, 2002.

[12] John A Nelder and Robert Mead. The downhill simplex algorithm. *Computer Journal*, 7(S 308), 1965.

[13] Tribler. A screenshot of the tribler application downloaded from https://www.tribler.org/, 2018.

[14] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M Frans Kaashoek, and Robert Morris. Designing a dht for low latency and high throughput. In *NSDI*, volume 4, pages 85–98, 2004.

[15] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Topologically-aware overlay construction and server selection. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1190–1199. IEEE, 2002.

[16] Yang Chu, Sanjay Rao, Srinivasan Seshan, and Hui Zhang. Enabling conferencing applications on the internet using an overlay muilticast architecture. *ACM SIGCOMM computer communication review*, 31(4):55–67, 2001.

[17] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 15–26. ACM, 2004.

[18] TS Eugene Ng and Hui Zhang. A network positioning system for the internet. In *USENIX Annual Technical Conference, General Track*, pages 141–154, 2004.

[19] Manuel Costa, Miguel Castro, R Rowstron, and Peter Key. Pic: Practical internet coordinates for distance estimation. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 178–187. IEEE, 2004.

[20] Youngki Lee, Sharad Agarwal, Chris Butcher, and Jitu Padhye. Measurement and estimation of network qos among peer xbox 360 game players. In *International Conference on Passive and Active Network Measurement*, pages 41–50. Springer, 2008.

[21] Sharad Agarwal and Jacob R Lorch. Matchmaking for online games and other latency-sensitive p2p systems. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 315–326. ACM, 2009.

[22] Yun Mao and Lawrence K Saul. Modeling distances in large-scale networks by matrix factorization. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 278–287. ACM, 2004.

[23] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[24] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[25] Johan A Pouwelse, Pawel Garbacki, Jun Wang, Arno Bakker, Jie Yang, Alexandru Iosup, Dick HJ Epema, Marcel Reinders, Maarten R Van Steen, and Henk J Sips. Tribler: a social-based peer-to-peer system. *Concurrency and computation: Practice and experience*, 20(2):127–138, 2008.

[26] Tribler. Dispersy documentation downloaded from http://dispersy.readthedocs.io/en/devel/index.html, 2016.

[27] Gertjan Halkes and Johan Pouwelse. Udp nat and firewall puncturing in the wild. *NETWORKING 2011*, pages 1–12, 2011.

[28] Niels Zeilemaker, Boudewijn Schoon, and Johan Pouwelse. Dispersy bundle synchronization. *TU Delft, Parallel and Distributed Systems*, 2013.

[29] Atul Singh, Miguel Castro, Peter Druschel, and Antony Rowstron. Defending against eclipse attacks on overlay networks. In *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, page 21. ACM, 2004.

[30] Ethan Heilman, Alison Kendler, and Aviv Zohar. Eclipse attacks on bitcoin's peer-to-peer network.

[31] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202. ACM, 2006.

[32] Alexa Megan Sharp. *Incremental algorithms: solving problems in a changing world*. Cornell University, 2007.

[33] Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[34] Stefan Saroiu Krishna P. Gummadi and Steven D. Gribble. King dataset downloaded from https://pdos.csail.mit.edu/archive/p2psim/kingdata/, 2002.