Low Latency Overlay in P2P networks

Building a decentralized low latency overlay in P2P networks

by

Bastiaan van IJzendoorn

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday January 1, 2013 at 10:00 AM.

Student number:	4024850	
Project duration:	March 1, 2012 – January 1, 2013	
Thesis committee:	Prof. dr. ir. J. Johan Pouwelse,	TU Delft, supervisor
	Dr. E. L. Brown,	TU Delft
	Ir. A. Aaronson,	Acme Corporation

This thesis is confidential and cannot be made public until December 31, 2013.

CONTENTS

1

INTRODUCTION

In the field of distributed systems new uprising applications appear such as cryptocurrencies and multiplayer gaming where computers are required to work together fast and without interruption. A distributed system is a system where computers work together and coordinate with each other by passing messages to each other. In recent years new advancements in the research on cryptocurrencies have shown new promising applications such as identity systems and online contracts. The applications make use of the internet, a international network between computers. On top of the internet an overlay is build called a peer-to-peer (P2P) network that connects computers on the internet together by introducing and connecting them to each other in a smart way. The computers in these P2P networks are called peers or nodes and there is no central element in the P2P networks that connects computers to each other. Instead computers are introduced to each other based on some pre-defined criteria. If the communication between the nodes are efficient in the P2P network, all the applications that make use of the P2P network called P2P applications can benefit from these efficiency's. In this thesis work we try to improve the efficiency between nodes in a P2P network to let all P2P applications benefit by improving the response time of nodes between each other. The response time between two nodes in a P2P network is called the latency between these two nodes.

1.1. THE IMPORTANCE OF LATENCY

Almost all systems have some requirements for latency, web applications, voice communication applications and multiplayer gaming applications all have latency requirements. In recent years latency requirements have increased with new applications such as trading in cryptocurrencies and systems that feature anonymous communication. We will discuss the latency requirements for some of these applications in further detail to show that a low latency between nodes in a P2P network can benefit these applications. [1]

1

LATENCY IN TRADING

A good example of an application where low latency communication is important is the trading in cryptocurrencies. Low latency communication means communication with a low response time. In the past 30 years, trading on the internet 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 the following advantages when trading 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]

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.

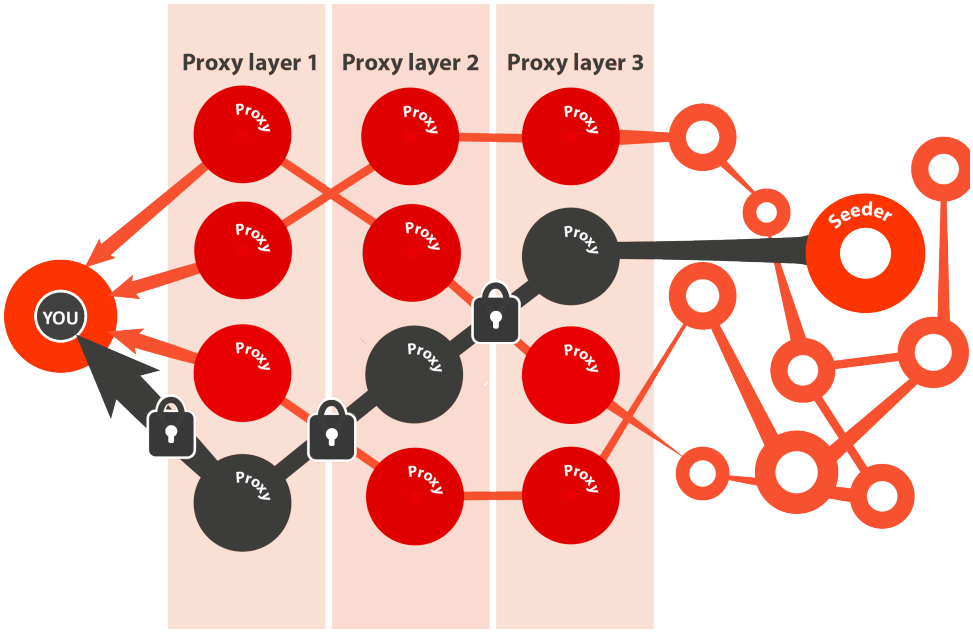


Figure 1.1: Anonymization techniques used in Tribler. There are three layers of the TOR protocol that make anonymous communication between peers.

LATENCY IN PARALLEL ALGORITHMS

In parallel algorithms one of the primary bottlenecks is the communication latency. The primary reason for this is the large amount of communication between nodes required in these algorithms. Only small amounts of computation work is done between communication events but the overall amount of communication is large. Parallel algorithms are used in a wide range of applications in for instance data mining and knowledge discovery. [8] [9]

1.2. LOW LATENCY OVERLAY

A new P2P overlay needs to be constructed to create an overlay in such a way that peers connect to other peers such that the latency between these peers is low. Peers to which a peer has a low latency with are called the low latency peers of that peer. The new overlay is called the low-latency overlay. In most existing P2P overlays a peer discovery mechanism is set up where peers introduce peers to each other based on some pre-defined set of rules. The introduced peers toward a peer are called the neighbours of that peer. The introduction mechanism allows the system to remain decentralized, e.a. the system is without a central authority that connects peers. In the new low latency overlay, peers should introduce low latency peers to other peers. This mechanism is called low latency introduction. A peer has to analyze and estimate what would be the low latency peers for another peer in the P2P network to provide low latency introductions

The central idea in this thesis is to estimate latency's between peers in a P2P network

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

to make low latency introduction possible. There are already existing latency estimation algorithms available that estimate latency's between computers in a network based on latency measurements. 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. Determining the coordinates can be computationally expensive and need to be done well in order to achieve a high level of accuracy in estimating latency's. Since 2002 other latency estimation algorithms were proposed for computationally efficient and accurate latency estimation on the internet resulting in more than fifteen years of research. These algorithms are discussed in a later chapter.

1.3. RESEARCH QUESTIONS

In this thesis work we focus on creating a latency overlay that is computational efficient and provides peers with low latency neighbours. The low latency overlay will be implemented in a real world P2P network. The following research question is answered:

How to create a computational efficient low-latency overlay that decreases the latency between connected peers in the P2P network?

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 in a real world P2P network?

3) What is the computational performance and accuracy of the new low latency overlay?

2

Related Work

In the past 15 years several methods have been proposed to estimate latency's between computers on the internet. These methods could be used in combination with current P2P technology to create the low latency overlay. This chapter describes Tribler as a state of the art P2P system that is used as a basis for the low latency overlay in section 2.1. Section 2.2 and 2.3 describe the latency estimation algorithms developed so far and the optimization functions that are used by the latency estimation algorithms. At last are some previously designed overlay systems described. 2.4

2.1. Tribler

The current state of the art peer-to-peer systems include social phenomena such as friendship and the existence of communities of users with similar tastes or interests. Tribler is such a social-based P2P system and is an extension on BitTorrent. The social phenomena are exploited in content discovery, content recommendation and downloading to increase usability and performance. The Vision and Mission of Tribler 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 founding 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.

Dispersy Overlay

Tribler is built upon Dispersy, the current internet overlay in Tribler. It is designed to send messages around in groups of peers in a decentralized P2P network. The groups of peers in the network are grouped together in communities where each community has

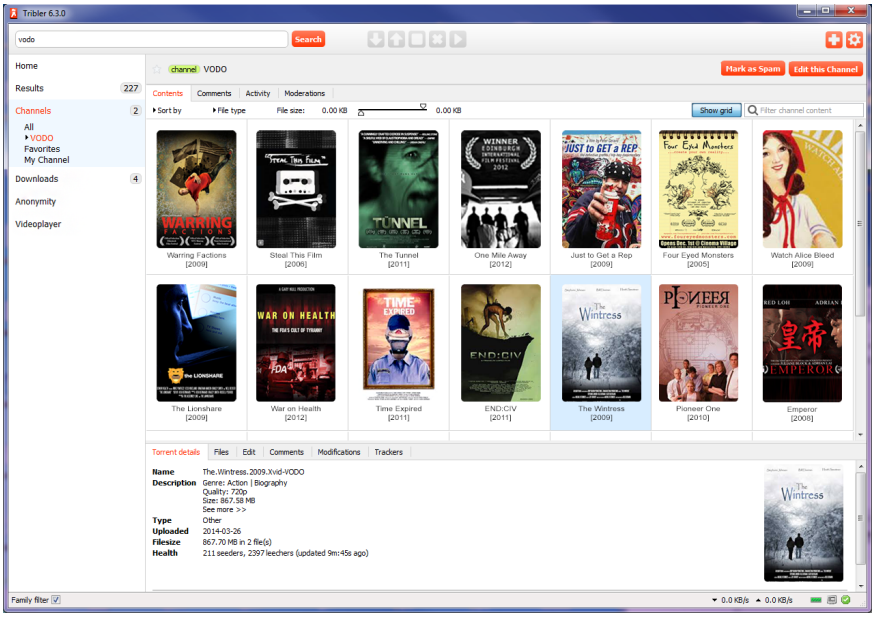


Figure 2.1: A screenshot of the Tribler application. [11]

its own purpose and design requirements. There are communities developed for P2P file sharing, TOR anonymity tunnels and market exchanges. Dispersy not only enables communities to exchange messages between peers in the network but also automatically connects peers to other peers in the network. There is a peer discovery mechanism that automatically introduces peers to other peers and makes connections to newly introduced peers by puncturing their firewall.

2.2. OPTIMIZATION FUNCTIONS

Optimization functions are often part of the latency estimation algorithms and need to be discussed first before latency estimation algorithms can be fully understood. In this paragraph we discuss the Simplex Downhill algorithm and the L-BFGS-B optimization algorithm as two important algorithms used in the overlay. The optimization functions are algorithms that minimize an objective function in an efficient way.

SIMPLEX DOWNHILL ALGORITHM

The most used optimization function in the literature is the simplex downhill algorithm. It is an applied numerical method used to find the minimum or maximum of an objective function with a multidimensional input space. It is applied to optimization problems for which derivatives of the objective function are not known. When optimizing an objective function with a n dimensional input space it maintains a set of $n+1$ test points where each test point reflects an input variable plus one extra test point. 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 such a way that they give a better

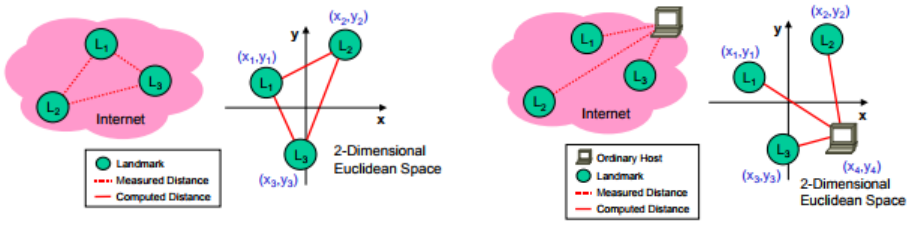


Figure 2.2: Part 1 and 2 of GNP algorithm. The left picture shows the first step of the GNP algorithm with landmark computation. The right picture shows ordinary host computation with ordinary hosts positioning themselves next to landmarks. [10]

solution for the objective function. When the objective function is converged towards a minimum the algorithm quits. For each test point is in each step decided whether increasing or decreasing the test point would give a better result for the objective function. If increasing a test point gives a better result of the objective function the test point is increased, if decreasing gives a better result the test point is decreased. Eventually the objective function is minimized and changing the test points does not give better results. When that happens the algorithm quits. [12]

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 of the input space are not available the algorithm tries to estimate the inverse Hessian matrix to make decisions on how to improve the test points for the objective function. The L-BFGS algorithm is particularly suited for problems with large amount of input variables. For instance more than 1000 variables. [13] [14]

2.3. LATENCY ESTIMATION ALGORITHMS

The latency estimation algorithms that are described in this section are coordinate-based latency estimation algorithms. Each host is represented by a position with coordinates in a space. The distance between the hosts in the space represents the two-directional estimated latency between these two hosts. Once the coordinates of the hosts in the space are determined the latency between two arbitrary hosts can be quickly estimated by taking the euclidean distance between the two positions that represents the hosts in the space.

GNP ALGORITHM

The first algorithm published is the GNP latency estimation algorithm published in 2002 and 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. The landmarks of step 1 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 with the simplex downhill algorithm.

In the second step the coordinates of the ordinary hosts that are not landmarks are determined. This is again done with a minimization of an objective function. The objective function is the sum of the differences between the measured and estimated latency from an ordinary host to all landmarks. With the simplex downhill minimization algorithm the objective function is minimized. Because the number of landmarks is low the position of an ordinary host is determined with relatively low computation.

With a low number of landmarks the computation time is only linearly dependent on the number of hosts H. However, with a large number of landmarks the algorithm becomes computationally expensive. There is a squared relationship between the amount of computation in the first step and the number of landmarks. In the second step there is only a linear relationship between the number of ordinary hosts and computation time. It is likely that with more landmarks the algorithm becomes more accurate but takes more time to compute. Therefore a trade-off between the number of landmarks and accuracy has to be made. In most applications of the GNP algorithm the number of landmarks is low and only around 20 landmarks and thus the computation time of the first step can be marginalized. [10]

NPS ALGORITHM

The NPS latency estimation algorithm is shortly published after the GNP algorithm in 2004 and improves it 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. [15]

VIVALDI ALGORITHM

Vivaldi is also published shortly after the GNP algorithm in 2004 and it conceptually differs from GNP in that it places a spring between each pair of nodes with a a rest length equal to the measured latency between the nodes. It is a variant to the GNP algorithm in that it also tries to minimize an error function to find good coordinates for nodes. Every pair of nodes exert a force on both nodes. The force of the first node has the direction

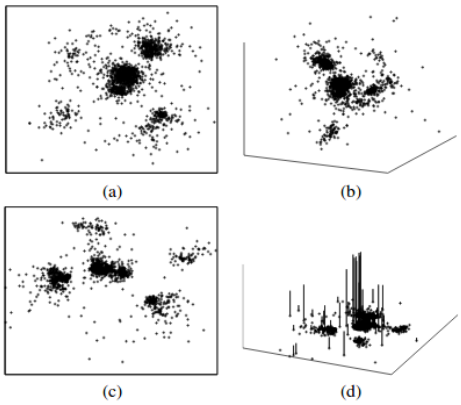


Figure 2.3: The node placement chosen by Vivaldi for the King data set (a) in two dimensions, (b) in three dimensions, (c) with height vectors projected on the xy plane, and (d) with height vectors rotated to show the heights. [16]

towards the second node and vice verse. 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. [16]

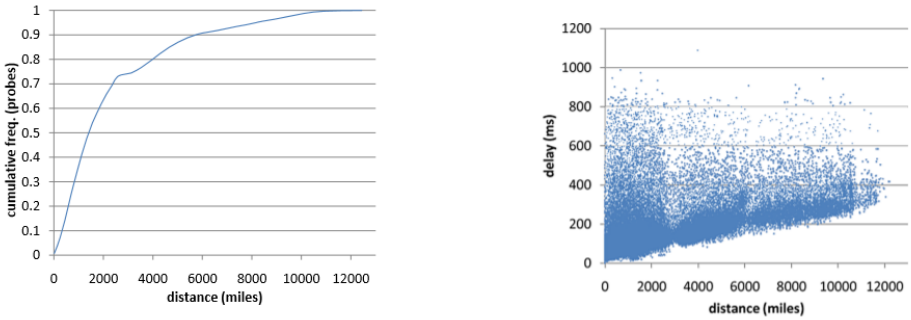


Figure 2.4: The figure on the left shows the cumulative distribution of the distance between consoles. The figure on the right shows the latency's measured for each distance between two nodes in miles. Both figures are from the experiments performed on the data from Xbox game consoles by Leet al al, 2008 [18]

PIC

The Practical Internet Coordinates for Distance Estimation (PIC) is another variant on the GNP algorithm published in 2004 that 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. 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. [17]

LATENCY ESTIMATION WITH GEO-LOCATION

Lee et al tried to do latency estimation with geolocation data in a publication in 2008. 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

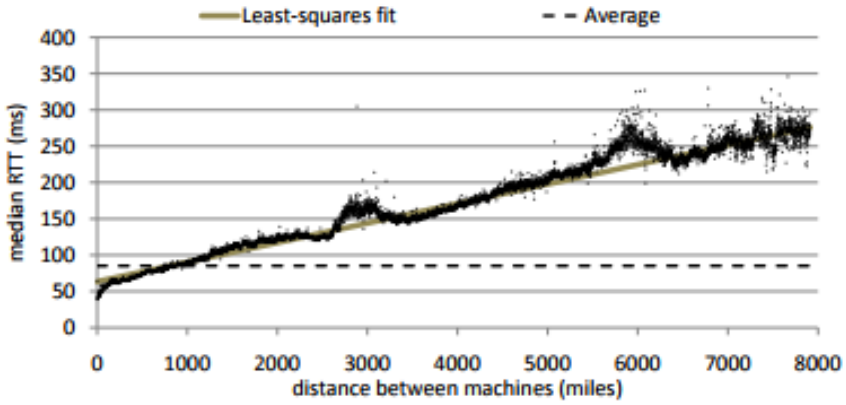


Figure 2.5: The correlation between the distance and latency. The latency data is the median of the data from the Halo 3 players database. The distance data is from MaxMind's IP-to-geo database. There is a clear linear relation between the distance and the median. The slope of the line is 0.0269 ms/mile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

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 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. [18]

HTRAE LATENCY ESTIMATION SYSTEM

Htrae is a latency prediction method published by Microsoft in 2012 merging both network coordinate systems (NCS) and earth geo-location approaches. It is one of the most advanced latency estimation algorithms so far. 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. 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 that 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 data-sets as many as 40% of these peer pairs have a shorter routing path trough an alternative peer instead of the internet. Next to asymmetric routing is common where the upstream and downstream capacities of a link are not equal. [19] 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 TIVs. 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. [20]

2.4. INTERNET OVERLAYS

In this section we describe two internet overlays as examples of systems that are build on top of the internet and try to improve themselves with low latency's. The literature provides theoretically concepts but publications with implemented applications are limited.

DHASH++

DHash++ is a distributed hash table (DHT) overlay that provides low-latency network storage. A DHT is a hash table in a distributed environment 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 to the GNP algorithm and uses coordinates to estimate latency's. However Vivaldi is a distributed algorithm where GNP can only be used locally. Whenever DHash++ nodes communicate with each other they exchange

coordinates. By this way a requesting node can predict the latency toward other nodes without having to communicate with them first. [21]

BINNING: TOPOLOGY AWARE OVERLAY CONSTRUCTION

Binning uses topological information about the relationship in nodes to make better routing policies and reduce latency in overlay networks. Nodes are grouped together in bins. The latency is reduced by putting nodes that are relatively close to each other in the same bin. 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 levels to each landmark determines the bin of the node. The method described reduces the latency and performance in network overlay construction but results not in a completely decentralized system because landmarks are being used. [22]

3

Problem Description

The main problem that is faced when creating the low-latency overlay in a real world P2P network is to make efficient latency estimation algorithms. Each peer in the low-latency overlay should be able to estimate the latency's between other peers such that the peers in the overlay can give introductions with low latency peers toward each other. Next to that, the low latency overlay should not be able to be taken down easily and be able to function if though some nodes might go down. At last, the low latency overlay should be resilient against certain attacks like the eclipse attack and Sybil attack.

3.1. Performance of latency estimation algorithms

The first requirement of the low latency overlay is that with a large number of peers N in the P2P network the latency estimation algorithms should still be computationally and memory efficient. If the algorithm computation takes too long, the computation can block a node in the P2P network. The node is then waiting for the algorithm computation to finish and does not respond on communication. When this happens, the latency of a random peer toward the blocking node increases and this cannot happen in the low latency overlay. Thus computational efficiency becomes a very important requirement for the system.

The algorithms should next to computationally efficient also be memory efficient and 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 share all their latency information to other peers the memory usage is N^2 where N is the number of peers in the network. The algorithms developed so far require that such amounts of latency information is stored. For instance, the GNP algorithm requires N^2 of measured latency's for a network with N peers.

The algorithm in the overlay should be able to deal with these large amount of measured latency's because in normal P2P networks the number of peers in the network can become millions. A choice has to be made in the algorithm about what latency's to send

to other peers to lower bandwidth consumption and what latency's to store in a peers memory to lower memory usage. The drop in latency information implies an information loss that could decrease the performance of the latency estimation algorithms. In the next chapters we will further evaluate the design choice and we will evaluate the effect of the information loss in the experimental section.

3.2. PEER DISCOVERY

A second requirement is that peers should be able to communicate with each other even though some peers are behind a NAT box. In the next paragraph is explained what a NAT box is. The communication is enabled in a peer discovery mechanism where peers are introduced to each other. It is impossible for the peer discovery mechanism to have a central authority as this will imply a central point that can be taken down and therefore let the whole system collapse. With a central authority in the overlay the system will become harder to maintain, could easily become the bottleneck of the performance of the system and would give some extra security threats. The code of the central authority would become different from the other peers and would require separate updating and monitoring which would increase the cost of maintenance of the entire system significantly. With no central authority the overlay can only be shut down if the entire internet is shutdown.

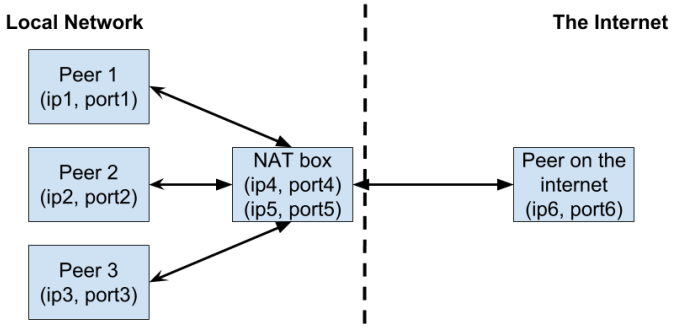


Figure 3.1: Network Address Translation (NAT). The NAT box has two IP, port combinations. $(ip4, port4$ is available on the local network and $ip5, port5$ is available on the internet.

NAT BOXES

Many computers lack a direct internet connection and are forced to take the initiative in communication. Computers on the internet are 64% of times connected to a NAT box in a local network that connects the computers in the local network to the internet. Figure 3.1 gives an overview of such a local network with a NAT box. A peer in a local network cannot be directly messaged by peers on the internet because the NAT box blocks incoming communication. Network Address Translation (NAT) is designed for the client-server model and not for a P2P network. 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. [23] In figure 3.1 are the IP addresses

and ports of the local peers 1,2 and 3 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 with the address available for the internet and the NAT box translates the IP, port combination to an IP, port combination of a peer from the local network. 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 peers in the local network 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 when a response is given to the NAT box. The peer on the internet can never initiate a connection and is forced in the server-role. [24] [25]

3.3. SECURITY REQUIREMENTS

SYBIL ATTACK

In the Sybil attack an adversary creates multiple pseudonym peers in the P2P network that flood or spam the network with false information. It is hard to solve the sybil attack in a decentralized P2P network because there is no central authority that can verify the identity of peers and distinguish between pseudonym peers and non-pseudonym peers. An adversary is able to take peers down with Distributed Denial of Service [DDoS] attacks by pseudonym peers. For instance, if an adversary wants to take down a target peer it could send a lot of peer introduction requests to a target peer with multiple pseudonyms. The target peer would be unable to respond to all introduction requests send by the pseudonyms and will be completely occupied with handling the requests from the pseudonyms. When this happens the target peer is taken down because it is unable to respond to messages received from normal non-pseudonym peers and is also unable to send messages toward other non-pseudonym peers. is able to subvert features of the P2P network such as a reputation system that maintains a reputation of peers. [26]

With multiple pseudonym peers an adversary could also manipulate certain features of a P2P application like for instance the reputation system of P2P networks. By letting pseudonyms collude with each other, the pseudonyms could gain a false high reputation. In a lot of P2P applications reputation systems are important to let the application function well. For instance, P2P file sharing applications make use of reputation systems to incentivize peers to not only use the P2P application but also contribute to the P2P network by sharing files with other peers. In P2P file sharing applications the reputation of a peer is often defined as the amount of data shared with other peers. Another example is the TrustChain online currency that also uses a reputation system to incentivize contribution to the application. If the reputation of a peer is low the peer will be denied service. [27] [28]

ECLIPSE ATTACK

Eclipse attacks have large implications on P2P networks. 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 neigh-

bourhood 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]

The eclipse attack is a very powerful and generic attack. We will provide several examples in the world of cryptocurrencies where eclipse attacks are used and have direct financial consequences. In most cryptocurrency systems a decentralized blockchain is used where transactions of the cryptocurrency are stored. Eclipse attacks are a powerful building block for the following attacks on cryptocurrencies.

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 3.2 shows a network where eclipsed nodes split the network in two. This split could be used to launch the 51 % attack.

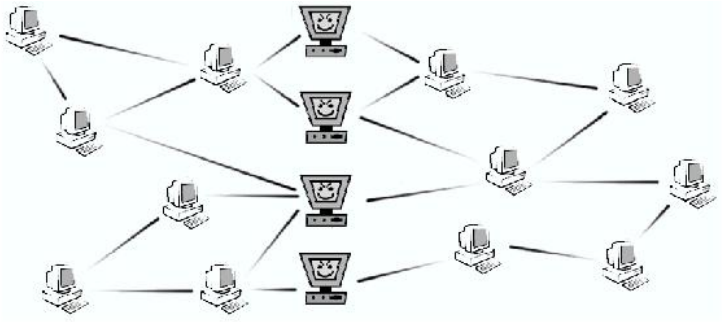


Figure 3.1: An Eclipse Attack: the malicious nodes have separated the network in 2 subnetworks.

Figure 3.2: 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 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]

4

OVERLAY DESIGN

In this chapter we will focus on how the low latency overlay is designed. First the latency estimation algorithms are described that estimate latency's between peers to enable good introductions. A good introduction is a introduction of a peer with a low latency toward the peer receiving the introduction. Incremental algorithms are used to make the latency estimation algorithms computationally and memory efficient. In the first section a description of incremental algorithms and the latency estimation algorithms is given. In the second section is described how the low latency overlay is designed into Tribler. The low latency overlay measures and obtains latency information from peers and saves this information memory efficient. The measured latency information is used by the latency estimation algorithms to give good introductions. At last is discussed how the introductions are given in the low latency overlay.

4.1. LATENCY ESTIMATION 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 4.1. 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 new input is given to the algorithm a new intermediate solution is immediately calculated. The new information updates the solution in such a way that when new information is fed to the algorithm the algorithm will eventually converge to a final solution. Calculating a new solution when new information added is called a step in the incremental algorithm and should not require much computational power. [31] [32]

The 5 latency estimation algorithms are explained in the following paragraphs. Each algorithm is given a unique name to distinguish it later in the experiment Chapter. Using incremental algorithms gives a computational benefit, but chopping the computation into pieces could give a worse accuracy of the latency estimation algorithms. The incremental algorithm chops the problem into pieces that are easy to compute and do

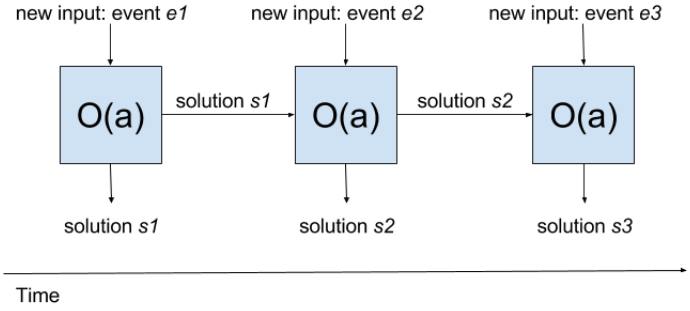


Figure 4.1: Overview of an online incremental algorithm. At each step a new input event e is added to the algorithm. A small computation with $O(a)$ complexity is used to calculate a new solution s. The new solution is used in the next step of the algorithm.

not block the processor of a peer. This is very important because if a peer blocks due to large computations, Tribler will wait for the processor to finish its computation. When waiting Tribler cannot respond to or send messages and thus will the latency between peers increase. Incremental algorithms give an accuracy cost because there is incomplete information because future latency measurements cannot be taken into account in a normal step. To mitigate the accuracy cost of this information loss the relation between past added and newly added information has to be analyzed. The latency estimation algorithm could look back at information added in a past step at a normal step to increase accuracy. To what extent computation time should be spend by the algorithms at looking to information that was added in the past to increase the accuracy is explored in the experimental chapter. The incremental algorithms that look back in the past are called RepeatStructured and RepeatTIV.

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.

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 $O(|L|)$.

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.

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 "RepeatStructured" 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 $O((R+1) * |L|)$.

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 "RepeatStructured" algorithm. In further sections we call this algorithm "RepeatTIV". 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 $O(L^2 * log(L) * R)$.

4.2. IMPLEMENTATION INTO TRIBLER

In this section is described how the low latency overlay is implemented into Tribler. First is described how the peer discovery mechanism work. Next is described how the low latency overlay obtains latency information. At last is described how the low latency overlay introduces peers to other peers. The goal of the low latency overlay is to give low latency peer introductions to other peers. A low latency peer introduction is an introduction of a peer A to another peer B such that the latency between peer A and peer B is low. In order to achieve this the latency between peer A and peer B has to be estimated with one of the latency estimation algorithms described in the previous paragraph. The latency estimation algorithm requires measured latency's between peers that are obtained by the overlay. The low latency overlay is build on top of dispersy and Tribler. The overlay consists of 1200 lines of code and can be downloaded open source from Github from https://github.com/basvijzendoorn/tribler/. Two test suits are written to test the code development, one for unit-tests and one for integration tests. The unit-tests have a test coverage of 68% and the integration tests have a test coverage of 70%.

PEER DISCOVERY AND NAT PUNCTURING

To explain what changes to the peer discovery are made to implement the low latency overlay the design of the current peer discovery mechanism of dispersy is first explained. In the current implementation of the peer discovery mechanism a peer introduction request and response mechanism is build. This mechanism requests another peer for an introduction and the other peer gives a response. The result is a list of peers that each peer maintains called the candidate list or neighbourhood of a peer. The peers in the candidate list are called the neighbours of a peer. A peer can always exchange data between two peers in the candidate list. The communication between two peers in the candidate list are always symmetrical. This means both peer *A* and peer *B* can send messages toward and receive messages from each other. If peer *A* has peer *B* in its candidate list then peer *B* also has peer *A* in its candidate list. The symmetrical property implies that both peers *A* and *B* assume the role of client and server in the P2P network and therefore the NAT firewall of one of the peers has to be punctured. This happens also in the peer discovery mechanism.

There are four phases in the current peer discovery mechanism. These four phases represent one step in the walk toward peers. Multiple steps are called a walk toward peers. By walking toward new peers each peer discovers a set of peers called its neighbourhood. The four phases are also shown in an overview in figure 4.2.

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 to introduce to peer A and sends peer A an introduction-response containing the address of peer C; peer A will add the address of node C to its neighbourhood.
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.

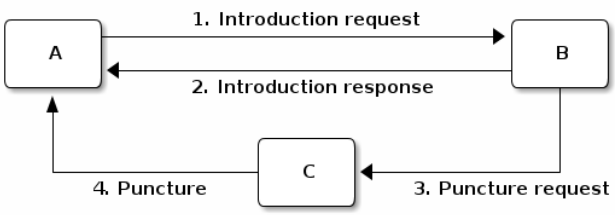


Figure 4.2: Overview of peer discovery in Tribler

The NAT puncturing mechanism is integrated in the peer discovery mechanism and works by sending puncture messages to other peers to puncture a hole in the NAT of the sender. In the third step of a peer discovery step peer *B* asks peer *C* to puncture a hole in its NAT for peer *A*. Peer *C* does this by sending a message toward peer *A* and therefore opening a port in its own firewall such that peer *A* can send a response. The two-way communication is complete if peer *C* sends another message to peer *A* to puncture a

hole in the NAT of peer *C* to enable peer *A* to send messages to peer *C*. Because both peer *C* and peer *A* send messages that punctured their NAT firewall the peers can communicate with each other without having to worry about the NAT firewalls.

NODE SELECTION

To prevent against eclipse attacks a node selection policy for nodes is implemented by dispersy to send an introduction request to. The candidate list is divided into 4 categories and Nodes are selected with pre-defined rules from these categories. The categories are:

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 are divided into the 4 categories according to the phases of the peer discovery mechanism. After a connection request is send from node *A* toward a node *B*, the node *B* is put into category IIIb of node *A*. If node *B* sends back a connection response and the connection is successful node *B* is moved from category IIIb to category II of node *A*. When an introduction-request is received the node that send the introduction-request is put in Category IIIa. Thus when node *B* receives the introduction-request from node *A*, node *A* is put into category IIIa of node *B*. The trusted nodes category consists of a special list of pre-defined nodes.

When selecting a node to send an introduction request to in a step, a choice is made from the 4 categories with pre-defined probabilities. The trusted node category I is chosen with a probability of 1%, category II is chosen 49.5% of times and category IIIa and IIIb are both chosen 24.75% of times. After a category is chosen, the node with the most recent interactions is selected from the selected category. Because some firewalls close inactive connections after a certain timeout the node with the most recent interactions is chosen. A closed connection is useless as then both nodes cannot communicate with each other anymore. [24] [25]

Dividing the nodes into the categories as described above has a dampening effect on an eclipse attack. If the attacker tries to perform an eclipse attack by introducing adversary nodes to a target node, the adversary nodes are only added to category III and not to category II. Because category II has a 49,5% selection probability when selecting a node for a step the adversary nodes will not always be selected. The node selection policy only mitigates an eclipse attack, it is still possible to do an eclipse attack with a lot of resources. To give extra protection the trusted node group is added. Whenever a node selects the trusted node group with a probability of 1% the candidate list is completely reset and all adversary nodes in the candidate list are automatically removed.

4.2.1. OBTAINING LATENCY INFORMATION

In this section are the mechanisms explained to measure and obtain latency information between peers. There are two mechanisms in the low latency overlay to obtain latency information: the ping-pong mechanism and the crawling mechanism. The ping-pong

mechanism has a double purpose. The first and most important purpose is to measure the latency's toward peers in the neighbourhood. The second purpose is to share previously measured latency's with other peers at the same time when latency's are measured. The crawling mechanism is an extra feature added to the overlay to quickly share latency information between peers. In contrast to the ping-pong mechanism is the crawling mechanism bandwidth inefficient. Therefore is the crawling mechanism not enabled by default in the low latency overlay.

PING-PONG MECHANISM

The latency between peers is measured with a ping-pong mechanism and previously measured latency's are also shared to other peers with the ping-pong mechanism. The ping-pong mechanism is started every PING TIME INTERVAL seconds by every peer in the P2P network. By default the value of the PING TIME INTERVAL is set to 2 second to frequently update the latency information between peers and to let the ping-pong mechanism do not consume too much bandwidth. It is important to frequently update the latency information between a peer and its neighbours for two reasons. At first, the latency to newly entered peers in the neighbourhood should quickly be measured to use this information in the incremental latency estimation algorithm. Secondly, the latency between peers can change over time. Nonetheless, the latency information cannot be updated too frequently because this will increase the bandwidth cost and processing time of the low latency overlay too much.

When the ping mechanism is activated every peer sends all peers in its candidate list a ping message. The peers who receive the ping message return a pong message. The time between send and return is measured to obtain the latency between two peers. The time when the ping message was send is stored by the peer to compare later with the return time of the pong response message. Upon arrival of the pong message the difference between the send and return time is calculated to obtain the latency toward the neighbour.

The ping message contains the IP and port of the peer sending the ping message, the time and 10 previously measured latency's added to the message. The payload format for the ping message is shown in figure 4.3. The low latency overlay always sends a new batch of 10 previously measured latency's toward the other peer that have not been send before. With a ping time interval of 2 seconds 50 measured latency's are send toward the other peer in 10 seconds with 5 ping messages. When all measured latency's are already send toward the peer the ping-pong mechanism will repetitively send measured latency's. All the measured latency's are maintained in a list by every peer. The ping-pong mechanism will first send the first 10 measured latency's, then the second 10 messages etc. When all measured latency's are send the ping-pong mechanism will start again from the beginning and send the first 10 measured latency's, followed by the second 10 measured latency's etc.

After receiving a ping message a pong response is given to the IP and 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. Figure 4.4 shows the payload format of the pong message.

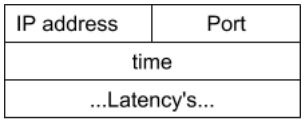


Figure 4.3: Ping payload.

Figure 4.4: Pong payload.

To make the low latency overlay memory efficient, only the top 100 latency's from one peer toward the top 100 closest peers send by ping messages are stored at every peer. The top 100 closest peers are the 100 peers that have the lowest latency to the peer receiving the ping messages. These 100 peers are estimated by the latency estimation algorithm. The low latency overlay has a low latency bias and thus only the latency's that are close to a peer are important and only the lowest latency's are remembered.

CRAWLING LATENCY INFORMATION

By default the crawling mechanism is not active on every peer to collect measured latency's from other peers that were collected with ping and pong messages. Every CRAWL TIME INTERVAL seconds a crawl request is send by each peer to every peer in its candidate 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 candidate 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 4.5. In a later paragraph we will explain how the forwarding mechanism is programmed.

An overview of the latency request payload is shown in figure 4.6. 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.

The latency response message payload is shown in figure 4.7. The IP address and

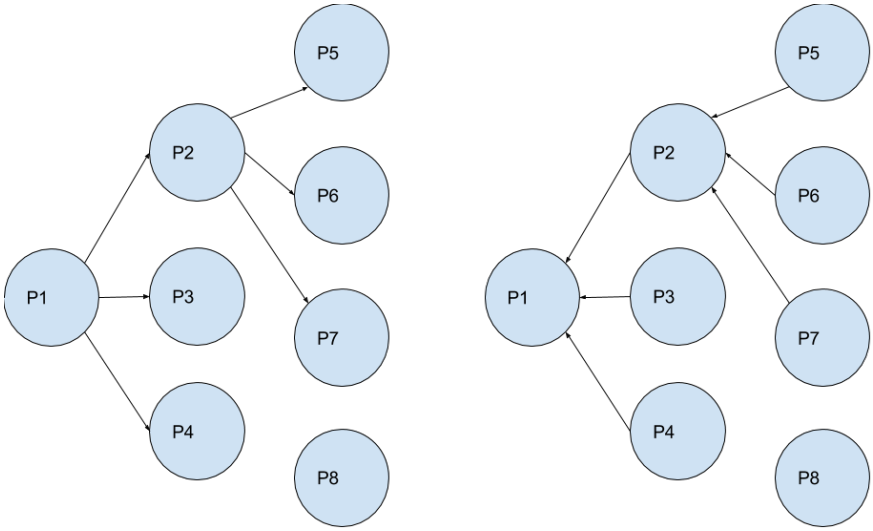


Figure 4.5: The left figure shows what happens when P1 sends a crawl request. The crawl request is forwarded to its neighbours P2, P3 and P4. These neighbours forward the crawl request to their neighbours to reach a maximum number of neighbours. In the right figure the latency response message is shown. 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.

IP address	Port	Hops
...Relay list...		

Figure 4.6: Overview of crawl request message.

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.

IP address	Port
...Relay list...	
...Latency's...	

Figure 4.7: Overview of latency response message.

THE FORWARDING MECHANISM

We will further explain how the forwarding mechanism is implemented. 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 4.8. 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.

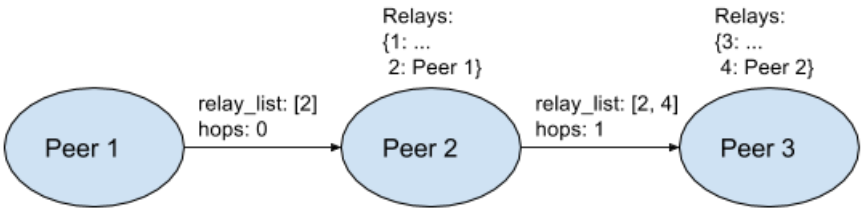


Figure 4.8: Schematic scheme of peer forwarding. In each communication line the $relay\_list$ is given. Each peer adds a new relay id to the $relay\_list$. When a peer receives a message the $relays$ dictionary is updated with the last added $relay\_id$ as key and the peer who send the message as result. 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 4.9. 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 4.9 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.

Sometimes the peer to which the latency response has to forwarded back is no more in the candidate 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 la-

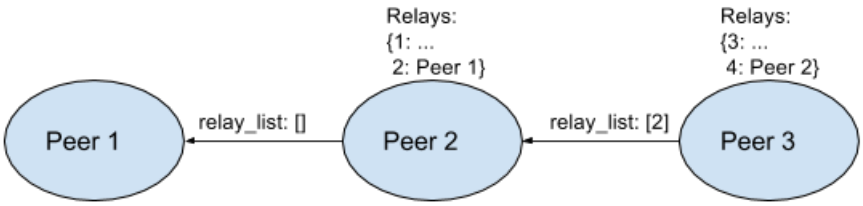


Figure 4.9: Schematic scheme of peer forwarding mechanism upon return. When the hop count exceeds the hop count limit the latency's are returned. The peer pops the last $relay\_id$ from the $relay\_list$ and uses this id to lookup the peer to backward the latency's to in the $relays$ dictionary.

tency crawler is activated in an interval the crawl initiator will eventually maybe retrieve the latency's of the peer that left the candidate list.

4.2.2. LOW LATENCY OVERLAY

A few changes are made in the peer discovery mechanism to enable low latency introductions. At first the peer introduction mechanism is changed. In the second phase of a step the introduction peer C is chosen from the neighbourhood of peer B to introduce to peer A. In the new low latency overlay the introduction peer is chosen in such a way that the latency between peer A and the introduced peer C is low. Peer B knows what peers have a low latency with peer A by using the results of the latency estimation algorithm. Peer B can simply calculate the distances between peers to peer A in the map that was constructed by the latency estimation algorithm to estimate all the latency's between peer A and other peers in the network. When the new overlay introduces always the peer with the lowest latency toward peer A the same peer could be introduced multiple times in different steps. This effect is called hammering of introduction peers. This can happen when the latency estimation algorithm gives the same results in two different steps. To avoid the hammering effect the new overlay chooses randomly one peer of the top n peers with the lowest estimated latency toward peer A to introduce to peer A. The default value for n is 8 and this value is chosen to give some variation in peer introductions to peer A but to also still introduce only peers with a low estimated latency towards peer A.

Secondly, the incremental latency estimation algorithm is run in the background of the peer discovery mechanism to update the results of the latency estimation algorithm continually. A new step in the latency estimation incremental algorithm is run every m seconds. The time interval m is chosen to be 3 seconds by default to update the latency estimation frequently but also to give the low latency overlay enough time to walk to other peers and obtain new latency information from other peers. For the "Repeat-Structured" and "RepeatTIV" algorithms the repetitive updating of previously calculated coordinates is also run in the background to save computational time. Every 1 second a previously calculated coordinate is updated by the latency estimation algorithm. Newly introduced peers P_{new} are remembered by the low latency overlay to use them as new input to the incremental algorithm. Dispersy takes a step with normal walking times every 5 seconds. This means every 5 seconds a new peer gets introduced and a new peer is

added to P_{new}. Nonetheless, the incremental algorithm can only benefit from the newly introduced peer if first latency information is obtained from that peer. Thus, the low latency overlay is designed that the incremental algorithm only uses the peers from P_{new} as input after latency information is obtained from P_{new}. The latency information of a newly introduced peer of P_{new} can be obtained with the ping-pong mechanism or the crawler after introduction. In the default setting in the low latency overlay latency information is obtained every second with the ping-pong mechanism from every peer in the neighbourhood. This means that in the worst case the latency information of peers of P_{new} is obtained one second after introduction.

At last the peer discovery mechanism is changed to prefer to take steps toward low latency peers. In the first phase of a step peer A chooses a peer from its candidate list to send an introduction to according to the node selection policy described above. In the new low latency overlay a combination of the old mechanism and a new mechanism that prefers low latency peers is build. In 50% of node selections the old mechanism is used and in the other 50% of node selections a peer with a low latency toward the selecting peer is chosen. When the overlay always selects the peer that has the lowest latency toward the selecting peer the hammering effect occurs. To prevent the hammering effect the selecting peer chooses randomly a peer from the top n peers with the lowest latency toward peer A.

5

EXPERIMENTS

In this chapter we describe the experiments that have been done to test the low latency overlay. We will first describe the performance metrics for the experiment. After that a description of a local experiment is given to test the performance of the different algorithms quickly with complete information. Next a description is given of two experiments in a decentralized Tribler setting with a P2P network consisting of 30 and 500 nodes to test the algorithms in a real world P2P network.

In all experiments, latency's are not measured in real time but are extracted from the King Dataset. The King Dataset is a latency data-set with latency's measured between computers in the real world. By using the King data-set the experiments use data input from the real world and thus are the results of the experiment applicable to the real world. The King Dataset is a NxN matrix with latency's measured between a set of 1740 DNS servers. [33]. The entry on row n and column m contains the latency measurement from DNS server n to m. This latency measurement is different of the latency measured the other way around from DNS server m to n. This latency measurement is represented by the entry of row m and column n. Because the algorithms described in the previous chapter assume that there is a single latency between two peers in the P2P network the latency of DNS server n towards DNS server m and the latency of DNS server m towards DNS server n are averaged and used for both latency measurements. In the experiments each node in the P2P network is given a unique ID. Whenever a node wants to lookup the latency measurement between two nodes with ID a and b it instead retrieves the latency measurement between DNS Server a and DNS Server b in the King Dataset. This provides the node with measured latency's from the real world.

5.1. PERFORMANCE METRICS

COMPUTATION TIME

The computation time performance metric measures how much time Tribler is blocked and computing something. It is an important performance metric because when a Tribler instance is blocked it cannot respond to communication and this increases the la-

tency of that Tribler instance. The computation time performance metric is calculated by taking the maximum amount of time a Tribler instance is computing something consecutively computation time of each incremental step of the incremental algorithm. This computation time can easily be calculated by taking the time difference of the time before and after the computation. When the computation of one incremental step is further spread in some algorithms like for instance in the Repeated algorithm each computational part that requires a separate calculation also blocks the Tribler instance and is counted as part of the computation time metric.

RELATIVE ERROR

The relative error metric measures how well an estimated latency matches the corresponding measured latency in the latency estimation algorithms. It measures the overall estimation performance of the algorithm. We will explain the relative error in a mathematical formula. In all latency estimation algorithms proposed in the previous chapter the latency estimation between two peers a and b in the P2P network is equal to the euclidean distance between two points in a geometric space. We call this distance the estimated distance or estimated latency. The measured latency is equal to the real world latency measurement between two DNS servers who represent peer a and peer b in the King data-set. For each estimated latency that can be calculated by the latency estimation algorithm between two peers a and b in the P2P network the relative error is defined as follows:

$$relative\_error = \frac{|estimated latency - measured latency|}{min(estimated latency, measured latency)}$$

A value of zero implies a perfect prediction as then the estimated latency and measured latency are equal and a value of one would imply that the estimated latency is larger by a factor of two.

RANKING ACCURACY

The ranking accuracy measures the quality of the latency estimation algorithms by comparing the lowest estimated latency's of the latency estimation algorithms with the lowest real measured latency's from the king data-set. Because the ranking accuracy measures how well the lowest latency's are estimated it is a good metric to evaluate the selective performance of the latency estimation algorithm. In other words, how good can the latency estimation algorithm make predictions on what peers have a low latency toward another peer. The ranking accuracy can be calculated at any point in time. The way this is done is as follows. Suppose we have a network with a set of P peers in the network. The latency estimation algorithm is run on every peer and can only estimate latency's between a limited set of peers B_a from the perspective of a peer $a \in P$. The set of peers B_a are all the peers that were introduced before the run of the algorithm to peer $a \in P$. The set of peers B_a becomes larger over time as more peers are introduced to peer $a \in P$. For each peer $a \in P$ the set of peer B_a is retrieved and the latency's toward all peers in B_a are estimated with the latency estimation algorithm and sorted in an ascending list of estimated latency's E_a. The list of estimated latency's E_a is a list of tuples (p, e) where p is the peer toward the latency is estimated from peer a and e is the estimated latency from peer a to peer p. The tuples in E_a are sorted in ascending order by the estimated latency

e in each tuple element (p, e). For the same peers in B_a the measured latency's of the king data-set are also sorted in a list K_a with a tuple structure (p, m) where p is the peer toward the latency is measured from peer a and m is the measured latency from peer a to peer p. From both sorted lists E_a and K_a we only compare the top 10% of lowest latency's in the list because we are only interested in the accuracy of the lowest latency's. Thus only the top 10% of the tuples of E_a and K_a with the lowest latency are saved to the lists $E10_a$ and $K10_a$ and the other 90% of tuples are deleted. The local ranking accuracy of peer a is defined as the percentage of peers that is both in the tuples of the lists $E10_a$ and $K10_a$. We call the peers in $E10_a$ the top 10% lowest estimated peers of peer a and we call the peers in $K10_a$ the top 10% lowest measured peers of peer a. The local ranking accuracy is thus what percentage of peers is both in the top 10% of lowest estimated peers and in the top 10% of lowest measured peers of peer a. The ranking accuracy of the whole network is the average of the local ranking accuracy's for every peer $a \in P$.

QUALITY OF INTRODUCTIONS

The quality of introductions metric measures how low the latency toward an introduced peer is compared to other possible introduced peers. The calculation of the metric is explained with an example calculation of an introduction. In the example peer a introduces a peer b to peer c and the metric is calculated by peer c where peer a, b and c are random peers in the P2P network. At first the row in the king data-set that contains all the latency's toward peer c is sorted by the latency in ascending order. The position of the latency that corresponds to the latency from peer b to peer c in the sorted list is equal to the quality of introductions metric.

TOP 10 LOWEST LATENCY PEERS

The top 10 lowest latency peers metric measures how well a peer knows the top 10 of peers with the lowest latency toward itself in the P2P network. To explain how the metric is calculated an example is given of how to calculate the metric for a random peer a in the P2P network. The top 10 lowest latency peers of peer a are the 10 peers in the P2P network with the lowest latency toward peer a. These peers can be calculated from the king data-set. Each row in the king data-set contains the latency's from one peer to all other peers in the P2P network. By sorting the row that contains all the latency's toward peer a the top 10 lowest latency peers toward peer a in the P2P network are calculated. Peer a also maintains a list of all the latency's it has ever measured. The metric is the percentage of peers that is both in the top 10 lowest latency's ever measured by peer a and in the top 10 of lowest latency's toward peer a in the P2P network.

5.2. CENTRAL VALIDATION EXPERIMENT WITH COMPLETE INFORMATION

The 4 algorithms described in chapter 5 have been implemented and are validated in this experiment in a central setting with complete information. The algorithms were run on a computer with a dual core 2.8 GHz processor. The experiment tries to mimic the peer discovery mechanism by incrementally adding peers to the network. In the beginning of the experiment there are 0 peers in the network. Every time a new peer is

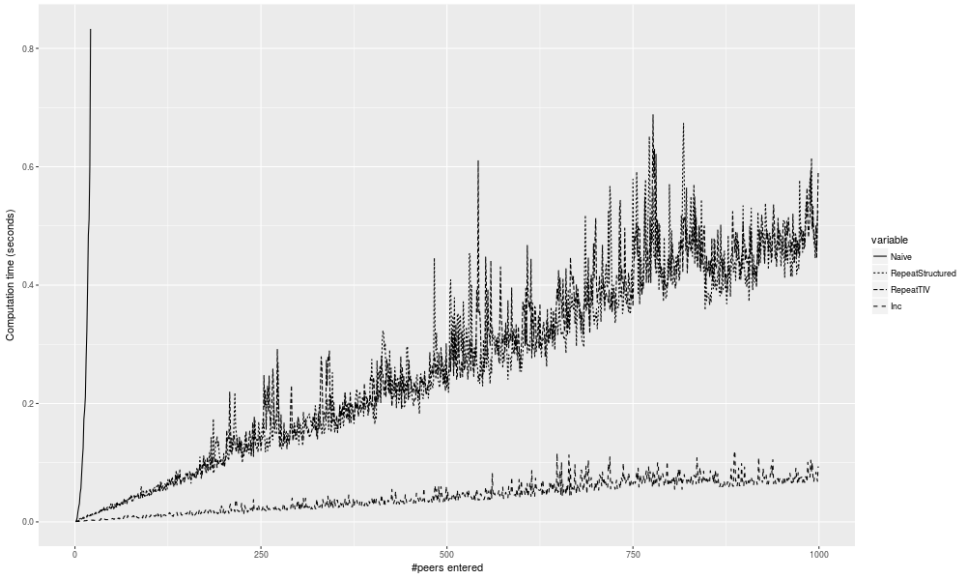
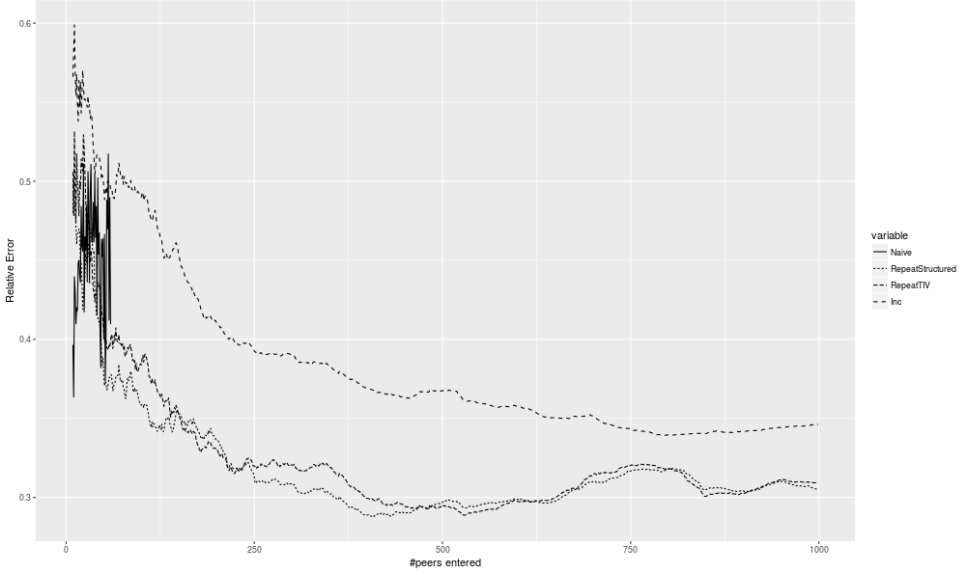


Figure 5.1: Graph of the computation time metric for latency estimation algorithms in a centralized setting. The x-axis are the number of peers that enter the system.



of the relative error metric for latency estimation algorithms in a centralized setting. The x-axis are the number of peers that enter the system.

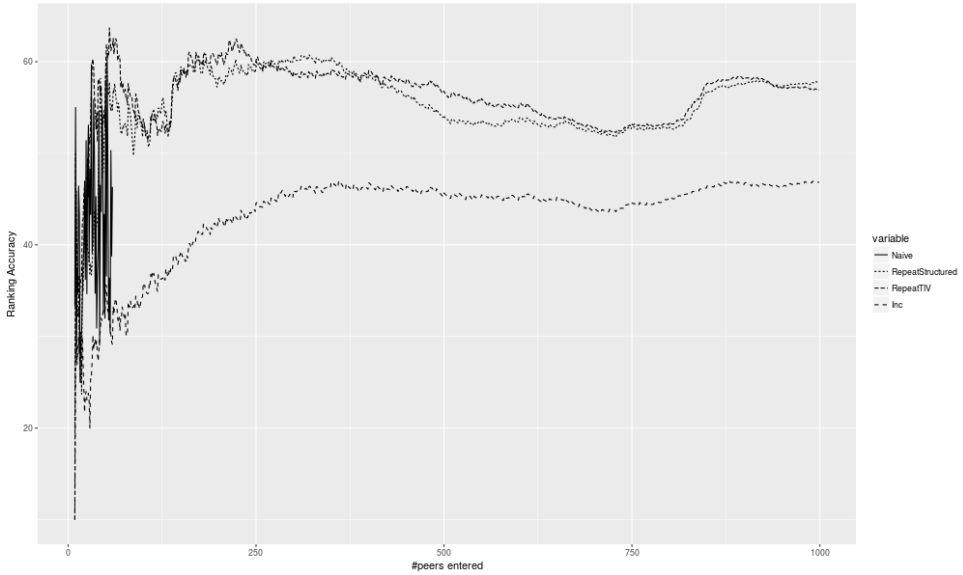


Figure 5.2: Graph of the ranking accuracy metric for latency estimation algorithms in a centralized setting. The x-axis are the number of peers that enter the system..

added to the network an incremental step of the algorithm is taken and the new peer is given complete information over all the latency's toward all other peers in the system. After every incremental step the computation time, ranking accuracy and relative error performance metrics are calculated. The computation time is the time to compute one incremental step and the ranking accuracy and relative error are measured of the whole P2P network.

The results of the computation time of the 4 algorithms can be seen in the graph of figure 5.1. The computation time of the naive algorithm grows exponentially, while the computation time of the other algorithms grow linearly larger as the number of peers entering the network increases. The computation times of the RepeatTIV and RepeatStructured algorithm eventually become larger than 0.5 seconds after 1000 peers are added to the network. Such a large computation time becomes problematic in networks with a large number of peers. For instance, in networks with millions of peers the RepeatStructured and RepeatTIV algorithm will block the application for a significant amount of time such that the latency toward a computing peer increases. This seems not to be a problem with the Inc algorithm because the linear growth of that algorithm is smaller. After 1000 incremental steps the computation time of the Inc algorithm is still smaller than 0.1 seconds. This suggests there will be less problems when scaling the Inc algorithm to larger networks with millions of peers.

The ranking accuracy and relative error performance results of the 4 algorithms are shown in the graphs of figures 5.1 and 5.2. When looking at the ranking accuracy and relative error performance metrics the RepeatTIV and RepeatStructured have the best performance and the Inc algorithm has the worst performance. The performance of the

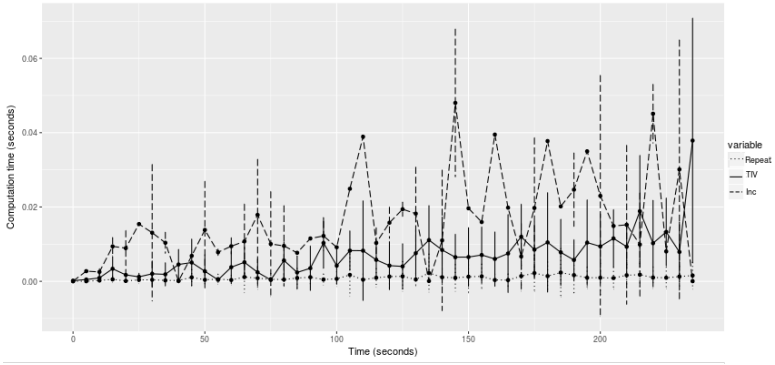


Figure 5.3: The figure shows the computation time metric in a decentralized experiment with 30 nodes. The x-axis is the elapsed time of the experiment. The dots represent the average computation time over every peer for each different algorithm at a certain point in time. The bar at every dot represents the size of the variance relative to the mean.

naive algorithm fluctuates between the Inc algorithm and the RepeatTIV and RepeatStructured algorithms. The Inc, RepeatStructured and RepeatTIV algorithms appear to have a warm up period when the first peers enter the network and the performance metrics converge to a constant value as the number of peers entering the network increases. The repetitive updating of already calculated coordinates in the RepeatTIV and RepeatStructured algorithms seems to be beneficial because the RepeatTIV and RepeatStructured algorithms converge faster and have a better overall performance in both performance metrics compared to the Inc algorithm. The overall performance of the repetitive updating algorithms is better because when the algorithms are converged the Inc algorithm has a ranking accuracy between 40% and 50% and a relative error around "0.35". The RepeatTIV and RepeatStructured algorithms have a higher performance with a ranking accuracy between 50% and 60% and a lower relative error around "0.30".

5.3. DECENTRAL VALIDATION EXPERIMENT WITH INCOMPLETE INFORMATION

The goal of the experiments is to validate the correct implementation of the low latency overlay in a decentralized setting with incomplete information. In each experiment one of the low latency estimation algorithms are validated. Only the "Inc", "RepeatStructured" and "RepeatTIV" algorithms are validated because the "Naive" algorithm was not computationally efficient enough in the centralized experiment. The decentralized experiments are run by the Distributed ASCI 5 supercomputer (DAS5). Multiple Tribler instances with the low latency overlay are run on the DAS5 and managed by the Gumby software in each experiment. Gumby allows to calculate and collect the data of the performance metrics. Because the goal is to only validate the correct implementation of the low latency overlay the experiments are run for a short period of 5 minutes. Every 10 seconds the performance metrics "Computation time", "Relative error", "Ranking accuracy" and the average latency towards peers in the neighbourhood are measured by

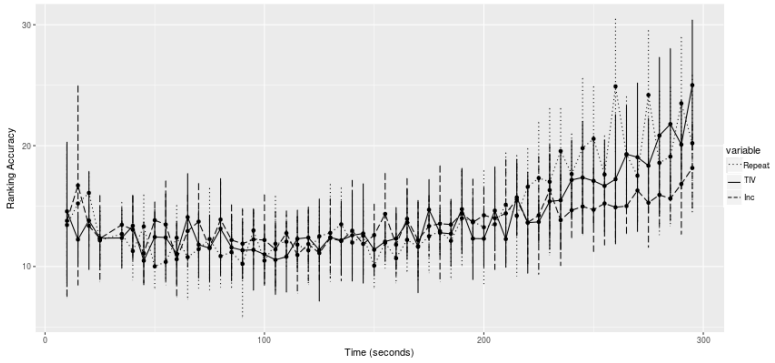


Figure 5.4: The figure shows the ranking accuracy metric calculated every 10 seconds in a decentralized experiment with 30 nodes. The dots represents the average ranking accuracy over every peer for each different algorithm at a certain point in time. The bar at every dot represents the size of the variance relative to the mean.

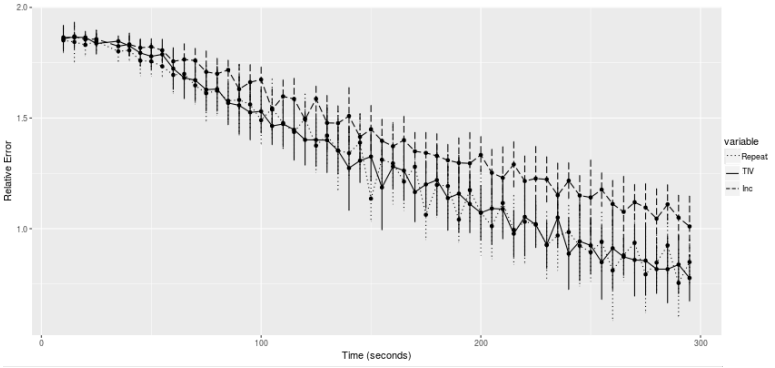


Figure 5.5: The figure shows the ranking accuracy metric calculated every 10 seconds in a decentralized experiment with 30 nodes. The dots represents the average relative error over every peer for each different algorithm at a certain point in time. The bar at every dot represents the size of the variance relative to the mean.

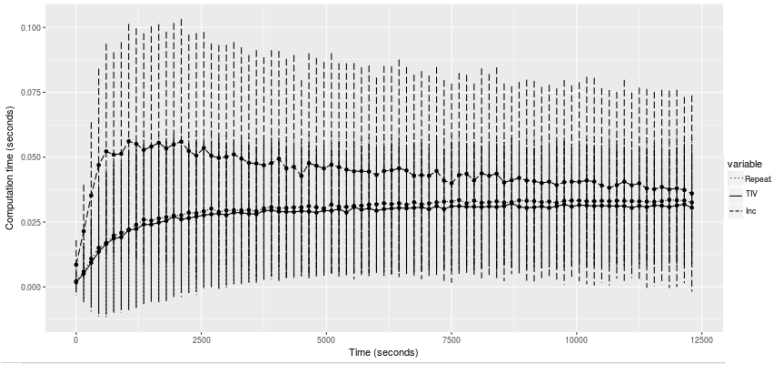


Figure 5.6: The figure shows the computation time metric in a decentralized experiment with 500 nodes. The x-axis is the elapsed time of the experiment. The dots represent the average computation time over every peer for each different algorithm at a certain point in time. The bar at every dot represents the size of the variance relative to the mean.

the low latency overlay. The resulting graphs of the performance metrics are shown in Figures 5.3, 5.4 and 5.5. [34]

The computation time performance metric of all algorithms is better than expected because all computation times are below 0,06 seconds. The computation time of the "RepeatTIV" and "RepeatStructured" algorithms are higher than the "Inc" algorithm because these algorithms recalculate the geo-location of past calculated peers. The "RepeatTIV" algorithm has a bit higher computation than the "Inc" algorithm because some extra computation is used 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 looking at the performance metrics the "RepeatTIV" algorithm seems to perform the best, followed by the "RepeatStructured" algorithm and at last the "Inc" algorithm. The relative error of all latency estimation algorithms decrease but are unable to converge in 5 minutes. The relative error of the "Inc" algorithm decreases slower compared to the the "RepeatedStructured" and "RepeatTIV" algorithm. The "RepeatTIV" and "RepeatStructured" algorithm have around the same progression in reduction of the relative error. After 5 minutes the ranking accuracy of the "RepeatTIV" and "RepeatStructured" algorithm is around 25% while the ranking accuracy of the "Inc" algorithm is around 18%. Past calculated geo-locations are updated in the "RepeatTIV" and "RepeatStructured" latency estimation algorithms which explains the better performance.

5.4. ACCURACY EXPERIMENT WITH INCOMPLETE INFORMATION

The goal of the experiment is to measure the performance of the latency estimation algorithms in a setting with incomplete information. The same measurement methods as with the validation experiment in decentralized setting are used. The experiment is executed over a period 12000 seconds which are 3 hours and 20 minutes. The graphs of the performance metrics and computation time are shown in Figures 5.6, 5.7 and 5.8.

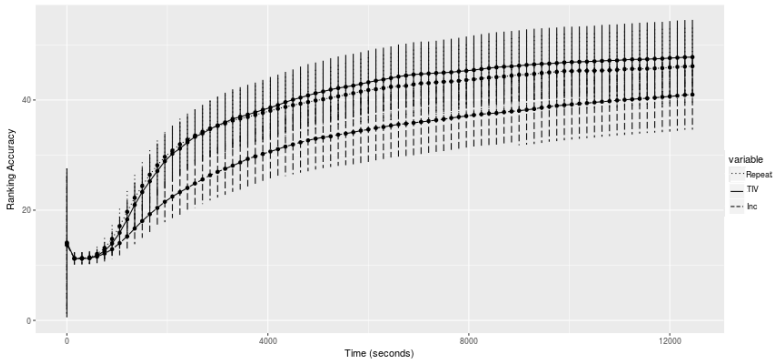


Figure 5.7: The figure shows the ranking accuracy metric calculated every 10 seconds in a decentralized experiment with 500 nodes. The dots represents the average ranking accuracy over every peer for each different algorithm at a certain point in time. The bar at every dot represents the size of the variance relative to the mean.

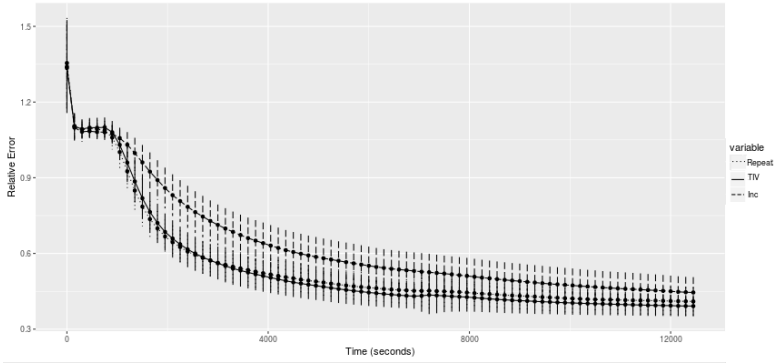


Figure 5.8: The figure shows the ranking accuracy metric calculated every 10 seconds in a decentralized experiment with 500 nodes. The dots represents the average relative error over every peer for each different algorithm at a certain point in time. The bar at every dot represents the size of the variance relative to the mean.

All algorithms show good results for computation time, with all computation times below 0.1 seconds. The "Inc" algorithm unexpectedly has a larger computation time than the "RepeatStructured" and "RepeatTIV" algorithm. An explanation for this difference could be that the coordinates representing the peers are already at a relatively good position in the "RepeatStructured" and "RepeatTIV" algorithm and require not so much change. Therefore the average computation time is lower in the "RepeatStructured" and "RepeatTIV" compared to the "Inc" algorithm. Eventually the computation time of all algorithm converges to an average value around 0.03 seconds and also the computation time of the "Inc" algorithm eventually converges to the same average value of the "RepeatStructured" and "RepeatTIV" algorithm.

The accuracy performance metrics give good results for the latency estimation algorithms but the convergence time is slow. All the algorithms converge to a value of around 0.45 for the relative error after 12000 seconds with a bit faster convergence for the "RepeatTIV" and "RepeatStructured" algorithm compared to the "Inc" algorithm. In 12000 seconds 2400 steps are taken by the peer discovery mechanism and the 500 nodes in the P2P network are visited multiple times. The latency estimation algorithms seem to react bad to incomplete information. In the first 100 steps taken by the peer discovery mechanism the relative error and ranking accuracy do not improve and remain at the same level for all algorithms. After 15 minutes the relative error starts to decrease and eventually converges. There are two explanations for the bad convergence. The first is that the latency estimation algorithms have to have more information on measured latency's before coordinate positions can be updated adequately to decrease the relative error. The second is that the latency estimation algorithms need more computation time to update all the measured latency's. This seems unlikely because there should be enough computation time available in 100 steps of the peer discovery mechanism.

The results for the ranking accuracy also show that the latency estimation algorithm give good results with a bad convergence time. Throughout the experiment the "RepeatTIV" and "RepeatStructured" have around the same ranking accuracy and the "RepeatTIV" algorithm only performs a little bit better for the ranking accuracy as time elapses in the experiment. Eventually the ranking accuracy of the "RepeatTIV" and "RepeatStructured" algorithm converges to a value of around 45% and the variance of the ranking accuracy for all the algorithms increases with time. Compared to the "RepeatTIV" and "RepeatStructured" algorithm the "Inc" algorithm has a worse performance but a good enough performance to let the ranking accuracy of the "Inc" algorithm eventually converge to an average value of 40%. At the start of the experiment the geo-locations of peers are randomly assigned which explain the large fluctuation in the performance at the start of the experiment. When comparing the converged ranking accuracy performance with the converged values of the centralized experiment the ranking accuracy performs around 10% worse. Most likely this is the result of the incomplete information in the decentralized experiment compared to the centralized implementation. In the decentralized experiment not all the measured latency's between peers are immediately available for the latency estimation algorithms.

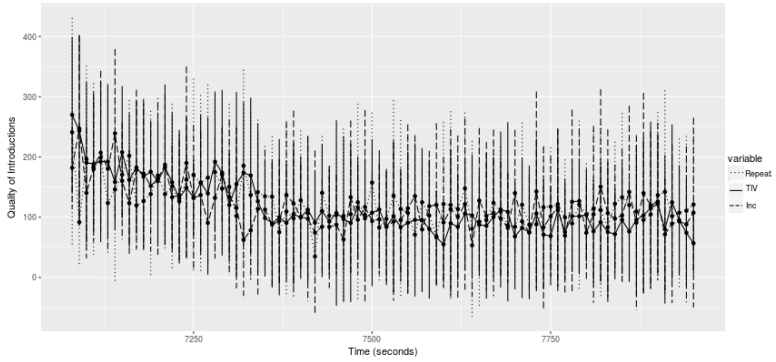


Figure 5.9: The figure shows the quality of introductions metric of the 10 new entering peers in the bootstrap experiment for the Inc, RepeatTIV and RepeatStructured algorithms. The x-axis shows the time in seconds. The dots represent the average quality of introductions metric. The bar at every dot represents the size of the variance of the quality of introductions metric relative to the mean.

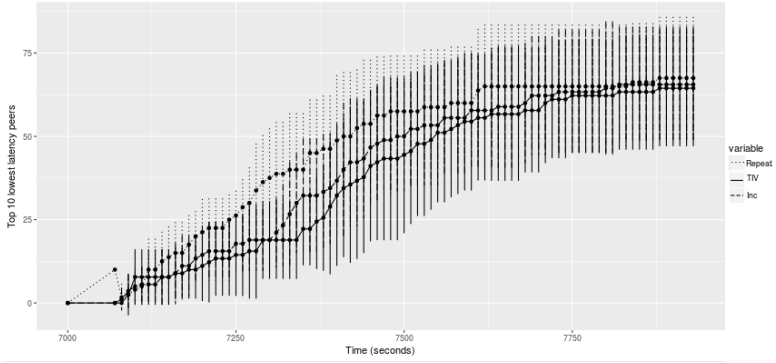


Figure 5.10: The figure shows the top 10 lowest latency peers metric of the 10 new entering peers in the bootstrap experiment for the Inc, RepeatTIV and RepeatStructured algorithms. The x-axis shows the time in seconds. The dots represent the average of the top 10 lowest latency peers metric. The bar at every dot represents the size of the variance of top 10 lowest latency peers metric relative to the mean.

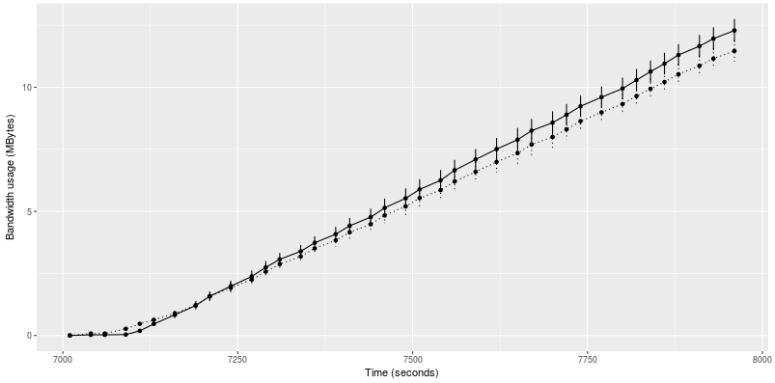


Figure 5.11: The figure shows the total upload and download for the 10 new entering peers in the bootstrap experiment. The x-axis shows the time in seconds. The dots represent the average total upload and the bar represents the size of the variance of the total upload relative to the mean. The solid line represents the upload and the dotted line represents the download.

5.5. BOOTSTRAP EXPERIMENT

The goal of the bootstrap experiment is to measure how well the low latency overlay reacts to new entering peers in the P2P network. It is expected that the 10 added nodes quickly know their top 10 latency peers and should receive high quality introductions quickly. In the experiment the low latency overlay is executed on 500 nodes for 133 minutes and 20 seconds (8000 seconds). In the first 116 minutes and 40 seconds (7000 seconds) 490 nodes are run normally with the low latency overlay. After the 7000 seconds have elapsed 10 nodes are added to the network that run the low latency overlay called the new entering peers. In this experiment only the performance of the 10 nodes that were added after 7000 seconds is measured. As shown in the previous experiment with 500 nodes the accuracy of the latency estimation algorithm after 7000 seconds is high and converged. This means the 490 nodes in the P2P network estimate latency's toward each other with high quality and these estimation can be used by the 10 new entering peers.

After 375 seconds the quality of introductions converges to a mean value of 100 for every algorithm. Figure 5.9 shows the quality of introductions for the new entering peers. There is no difference in introduction quality with different latency estimation algorithms. In the beginning of the experiment the quality of introductions is averaged 250, half the number of peers in the P2P network, indicating a random introduction of peers.

It takes 1000 seconds for 10 peers to find 64% of their top-10 low-latency peers. Figure 5.10 shows the correctness of the top 10 lowest latency peers for the 10 new entering peers. The correctness increases as time progresses. After 500 seconds most algorithms got 50% of their top 10 lowest latency peers correct and after a few steps some new entering peers already know some of their top 10 lowest latency peers.

The results of the experiment are worse than expected. It takes more time than expected before the new entering peers receive high quality introductions and before the new entering peers know their top 10 lowest latency peers. With limited knowledge on

latency's toward new entering peers the latency estimation algorithm of other peers cannot estimate latency's correctly. Whenever a new entering peer enters the P2P network it has not yet measured any latency toward another peer. Therefore it is impossible for the other 490 peers to calculate the geo-location of the new entering peer in the beginning of the experiment with high accuracy. The latency estimation algorithm can only guess where the geo-location of the new entering peer is. As more latency's are measured between the new entering peers and other peers in the P2P network the latency estimation algorithm can calculate the geo-location of the new entering peer with higher accuracy. New latency's are measured immediately after a new peer is discovered by the new entering peers. It requires 75 latency measurements and peer discoveries before the introduction quality converges because after 375 seconds 75 peers are discovered.

The bandwidth usage of the new entering peers is low as expected. Figure 5.11 shows the bandwidth usage in bytes for the 10 new entering peers in the bootstrap experiment. The upload and download speed are throughout the experiment about the same and show a linear growth with a slower growth in the beginning of the experiment. The slower growth in the beginning is normal because when the peers just entered no latency's are yet measured and less than 10 latency's are send in the ping message and thus is the message size smaller. After 1000 seconds the total byte usage is around 24000 Kbyte, 12000 Kbyte total upload and 12000 total Kbyte download. 1 KByte is assumed to be 1024 bytes. The average upload and download speed throughout the experiment is 12 KByte. The expected bandwidth usage is the addition of the byte cost of the ping-pong mechanism and the peer discovery mechanism. Every 2 seconds ping messages are send to each peer in the neighbourhood. The neighbourhood consists of 35 peers on average and one ping message is around 350 bytes. After 1000 seconds the byte cost of the ping message is around 350*35*(1000/2) = 6125000 bytes = 5981.44 KByte. The other 6 KByte is most likely used by the pong message and the peer discovery mechanism. The total byte usage to converge towards high quality introductions is around 9000 KByte per peer because convergence is reached after about 375 seconds. In those 375 seconds 188 ping and pong messages are send by the peers and 75 peer steps are taken in the peer discovery mechanism.

6

Future Work

Finding better algorithms to solve the convergence problems and provide latency estimation with higher accuracy should have the highest priority for future work. The latency estimation algorithms need to be of high quality to let the low-latency overlay fully rely on them. The current low-latency overlay takes a long time to converge and when new peers enter the P2P network the overlay does not provide them with low latency peers fast. On the positive side does the low-latency overlay converge at some point and are incremental algorithms used with a low computation cost. Algorithms that handle lack of information well and Triangle inequality violations (TIVs) could be further investigated to improve the accuracy because the algorithm that tried to counter triangle inequality violations worked the best.

Experiments with the low latency overlay on large networks should be done to test and develop a low latency overlay suitable for 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. It should be tested whether the latency estimation algorithms converge towards a result with high accuracy in a large P2P network. The algorithm should also be able to handle large latency inputs with more than 100000 latency's at each step of the incremental algorithm. One incremental step should be 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 algorithms that deal with lack of information and how to counter Triangle Inequality Violations (TIVs) can further improve the accuracy and convergence of the latency estimation algorithms. The experiments so far have shown that algorithms with TIV prevention and algorithms that repeat past calculated coordinates are useful. Accuracy of latency estimation algorithms 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.

An algorithm could be created that handles lack of information more efficiently or more latency information could be gained via other ways with for instance ICMP messages. Other possibilities to detect TIVs and prevent them could be further investigated to improve the accuracy of the overlay.

7

Conclusion

Building a low latency overlay to give low latency connections between peers is a hard problem. To make good decisions on what peer to introduce to another peer the latency's between two arbitrary peers in the P2P network have to be estimated. Finding good algorithms that estimate latency's between peers computationally and memory efficient is hard. The current state of the art latency estimation algorithms provide enough accuracy to build a low latency overlay but lack the accuracy to quickly provide new entering peers with low latency peers. Incremental algorithms are used to divide the computation over time but methods that use more computational power create a more accurate latency overlay. These methods require a very frequent updating of previously estimated latency's. It is beneficial to counter peers who cause Triangle Inequality Violations (TIVs) in the latency estimation algorithm because algorithms that deal with TIVs perform better and have a better accuracy. The low latency connections between peers has various benefits to various applications. It provides faster trading and faster onion routing.

Low latency peers cannot be introduced to other peers if there is no peer discovery mechanism that traverses the NAT boxes which enable peers to connect to the internet. A NAT traversal mechanism is required because most computers used by peers are not directly connected to the internet but behind a NAT box in a local network. In the overlay NAT boxes are punctured by previously discovered peers to enable good connections between peers.

Eclipse attacks are a very generic attack and powerful attack on the low latency overlay and is countered in the design of the overlay. The 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 the eclipse attack, nodes could be controlled by adversaries or nodes could receive false information about other peers or about things from a P2P application. This gives very powerful attacks on cryptocurrency applications with direct financial consequences and thus are eclipse attack prevention methods important.

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

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

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

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

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

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

[26] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[27] Qiao Lian, Zheng Zhang, Mao Yang, Ben Y Zhao, Yafei Dai, and Xiaoming Li. An empirical study of collusion behavior in the maze p2p file-sharing system. In *Distributed Computing Systems, 2007. ICDCS'07. 27th International Conference on*, pages 56–56. IEEE, 2007.

[28] Pim Otte, Martijn de Vos, and Johan Pouwelse. Trustchain: A sybil-resistant scalable blockchain. *Future Generation Computer Systems*, 2017.

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

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

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

[34] Henri Bal, Dick Epema, Cees de Laat, Rob van Nieuwpoort, John Romein, Frank Seinstra, Cees Snoek, and Harry Wijshoff. A medium-scale distributed system for computer science research: Infrastructure for the long term. *Computer*, 49(5):54–63, 2016.