Communicating with low latency peers

Building a low latency overlay in P2P networks

by

Bastiaan van IJzendoorn

to obtain the degree of Master of Science
at the Delft University of Technology,

Student number:	4024850	
Thesis committee:	Dr. ir. J.A. Johan Pouwelse,	TU Delft, supervisor
	Dr. J.S. Rellermeyer,	TU Delft
	Dr. M.T.J. Spaan,	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

PREFACE

This thesis would not be possible without the help and support of various people. At first, I would like to thank my supervisor Johan Pouwelse for this feedback, comments and suggestions. Furthermore, I would like to thank Jan Rellermeyer for his feedback and comments on the thesis. Thanks to Martijn de Vos, Quinten Stokkink and Egbert Bouman for helping me out with Tribler. Finally thanks to Milan Lopuhaa for providing me with comments.

A special thanks goes to my parents who gave their unconditionally support when creating this thesis. I would also like to thank my friends.

Bas van IJzendoorn
Leiden, June 2018

ABSTRACT

In decentralized applications requirements on latency are high. For instance, in trading applications a millisecond latency advantage can save companies millions of dollars. In this thesis a design is proposed for an overlay in a decentralized peer-to-peer system that lowers the latency between connected peers. The overlay is designed and implemented on top of Tribler, a BitTorrent client developed at Delft University of Technology. The client continually connects toward the peers with the lowest latency to keep an established connection towards them. Next to that, the introductions in the peer discovery mechanism are chosen in such a way that introduced peers have a low latency toward each other. To achieve this latency estimation algorithms are used that estimate the latency between arbitrary peers in the network. These algorithms require a limited amount of measured latencies between peers in the network and can estimate the others. Latencies are continually measured toward peers that have an established connection to a client. These measured latencies are shared to other peers for further improvements on the quality of the latency estimation algorithms. The overlay design is evaluated on the accuracy of the latency estimation algorithms, computation time and to what extent the connections of a peer are low latency. The results show that the peers in the overlay are able to connect toward their lowest latency peers despite low accuracy of the latency estimation algorithms.

CONTENTS

1

INTRODUCTION

In the field of distributed systems new applications are created such as cryptocurrencies, online contracts and match making in multiplayer gaming. The requirements on latency are high in these new systems. In distributed systems multiple processes are executed. Pairs of processes can communicate to each other by passing messages. Some applications make use of overlay networks built on top of the internet for efficient communication. Widely used distributed systems are peer-to-peer (P2P) networks with no central administration element. Unlike the traditional client-server model, computers in P2P networks are called nodes or peers. In this thesis we try to lower the response time (latency) between nodes in P2P networks. This enables communication to go faster between P2P applications.

1.1. THE IMPORTANCE OF LATENCY

To show that a low latency is beneficial in a P2P network the latency requirements for trading and anonymous communication applications are discussed. Latency is a bottleneck for efficient communication and latency sometimes also effects the logic of a system when, for instance, resources in a trading system are given faster to nodes with a low latency.

LATENCY IN TRADING

In the past 30 years, trading has become faster. Due to large competition and investments into technology the time it takes to process a trade has gone from minutes to seconds to milliseconds. In financial technology latencies are at wire speed with definitions of a "Low latency" under 10 milliseconds and an "Ultra-Low latency" under one millisecond. Around 50% of trades in the U.S. are done with an "Ultra-low latency". Some firms state that 1 millisecond advantage can save 100 million U.S. dollars [1]. Traders have the following advantages in low latency systems [2]:

1. Better decision making: Whenever communication on trades arrives later, the decisions of traders are different. Traders continually communicate with each other

1

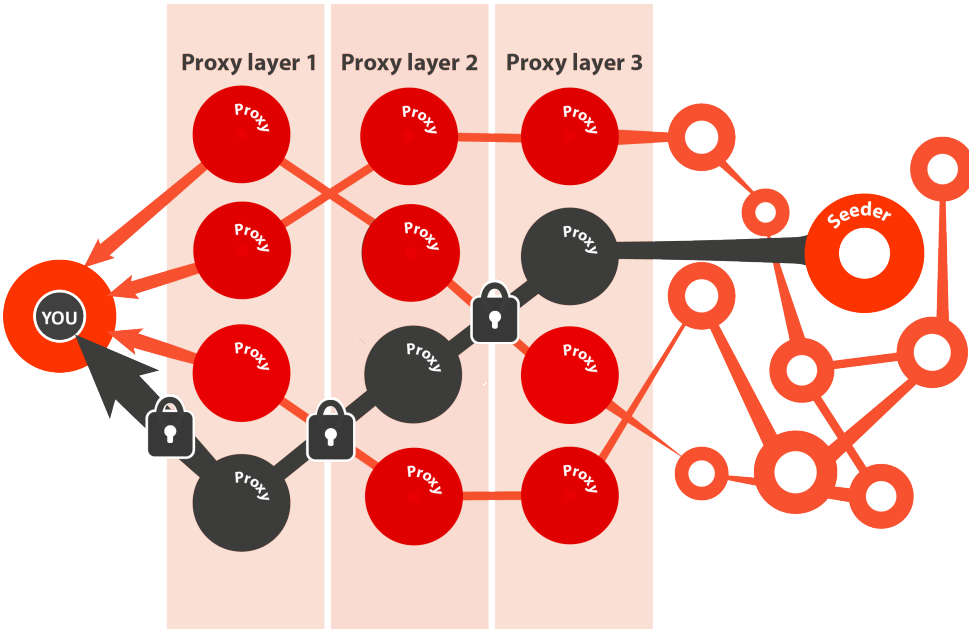

Figure 1.1: Anonymization techniques used in Tribler. There are three layers of the TOR protocol that make anonymous communication between peers.

by offering bid and ask prices for specific quantities of goods. They also communicate with each other to cancel trades. When this information is processed later there is a different overview of the market.

2. Competitive advantage towards other traders: A trader has competitive advantage when it can respond faster to trades. If a price differentiation takes place, nodes with a lower latency can act earlier. The goods can then be bought before a price correction takes place.

3. Higher priority for nodes with a low latency: Offers that arrive first are served first. To compensate, traders with low latency can lower their price to get a higher priority[3] [4].

LATENCY IN ANONYMIZATION TECHNIQUES

Another example of an applications with high latency requirements are anonymization techniques. The time required for data transfers in anonymization techniques is highly dependent on the latency between nodes. Data is communicated and encrypted via a chain of nodes to ensure the sender and receiver are unlinkable. The total latency of one data transfer is greater or equal than the addition of the latency between the chain of nodes. Especially in the Mix network anonymization technique proposed by Chaum in 1981 latency has a large effect[5]. In mix networks messages are batched at nodes and batches are forwarded when a certain amount of messages are received. Messages have

Figure 1.2: Graph of earth in a geometric space with the physical location of peers represented by dots in the space. The distance between peers estimates the latency.

to wait at nodes before others arrive to be reordered and forwarded. Therefore the total latency between the sending and receiving node becomes very high. To reduce the total latency Dingledine *et al* proposed in 2004 TOR onion routing[6]. In TOR Messages are not batched but forwarded in real time between a chain of nodes to reduce the latency between sender and receiver. With TOR, only the latencies between nodes are the bottleneck. Figure 1.1 shows an overview of onion routing.

1.2. LOW LATENCY OVERLAY

A low latency overlay is constructed in this thesis that provides nodes with low latency connections. Latencies between nodes are estimated with algorithms to make better decisions when making connections. Various algorithms in the past 15 years have been proposed that model peers as dots in a geometric space where the distance between the dots estimates the latency[7] [8] [9] [10] [11]. By changing the coordinates of the dots, the algorithms improve the latency estimation. Dots in the geometric space can be interpreted as the physical location of peers on earth. Such interpretations reside on the assumption that the latency estimation algorithms correctly provide an earth like map. Figure 1.2 shows the idea of a location space similar to earth with dots representing peers.

In this thesis the focus is on creating a latency overlay that is computational efficient and provides peers with low latency connections. The following research question is answered:

How can a computational efficient overlay be created 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 latencies between computers 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 algorithms have been proposed to estimate latencies between computers on the internet. By combining the research to the latency estimators with the state-of-the-art P2P technology a new low latency overlay is designed. This chapter describes the state-of-the-art P2P technology in section 2.1 and the past research to latency estimation algorithms in section 2.3. Section 2.2 describes the optimization functions that are used by the latency estimation algorithms and in section 2.4 some previously designed overlay systems are described.

2.1. Tribler

Tribler is an extension of BitTorrent and includes social phenomena such as friendships and the existence of communities of users with similar tastes or interests. The social phenomena are exploited in content discovery, content recommendation and uploading and downloading of files to increase usability and performance. Tribler has the following Vision and Mission:

"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[12]."

Since the founding of Tribler in 2005, about 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 that gives superior protection to VPN, but no protection against resourceful spying agencies. A reputation system is included that provides incentives for users to upload resources instead of only downloading resources from the network[12]. A screenshot of Tribler is given in figure 2.1.

Dispersy Overlay

Dispersy allows nodes to communicate with each other in Tribler using one or more overlays. Each overlay is called a community and consists of a group of peers that communicate with each other according to specific design criteria in line with the purpose

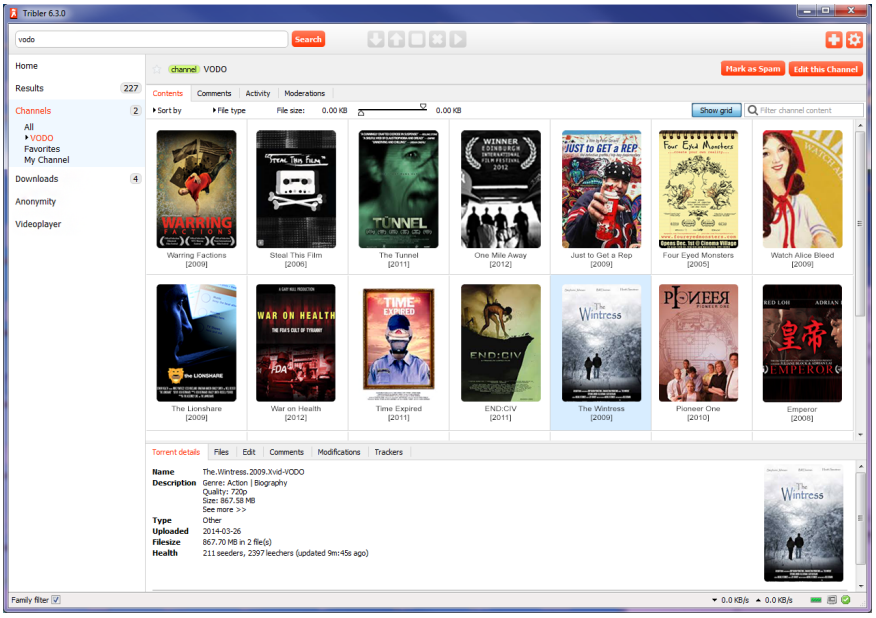

Figure 2.1: A screenshot of the Tribler application[14]

of the community. There are communities for P2P file sharing, TOR-like anonymity tunnels and market exchanges. In every community each node maintains a list of peers called the neighbourhood with active connections toward the node. Whenever a node connects to the overlay, the neighbourhood is initially empty. To fill the neighbourhood with active connections Dispersy has a peer discovery mechanism that introduces peers to each other and punctures the firewalls protecting the peers[13].

2.2. OPTIMIZATION FUNCTIONS

In this paragraph we discuss the Simplex Downhill and the L-BFGS-B optimization algorithms. The algorithms approximate the minimum of a multi-dimensional objective function in different ways.

SIMPLEX DOWNHILL ALGORITHM

A widely used optimization function is the simplex downhill algorithm[7] [8] [9] [10]. 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 an n dimensional input space the algorithm 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. At each step for each test point it is decided whether increasing or decreasing the test point would give a better result for the objective function

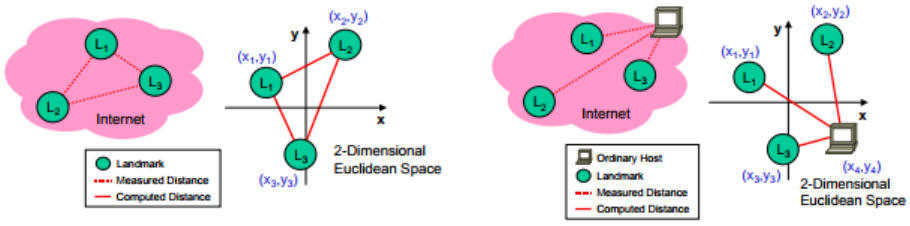

Figure 2.2: Part 1 and 2 of GNP algorithm. The left picture shows the first step of the GNP algorithm with landmark computation. The right picture shows ordinary host computation with ordinary hosts positioning themselves next to landmarks[7].

and the test point is updated accordingly. When after several steps the objective function is converged towards a minimum, the algorithm terminates. The algorithm always gives an approximation of the minimum of the objective function because sometimes the algorithm converges toward a local minimum instead of a global minimum[15].

L-BFGS-B

The Limited-Memory Broyden-Fletcher-Goldfarb Algorithm (L-BFGS-B) improves on the simplex downhill algorithm with mathematics and is especially suited for problems with large amount of input variables. The basis of the algorithm is similar to other optimization techniques in that it tries to optimize a set of 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[16] [17].

2.3. LATENCY ESTIMATION ALGORITHMS

The latency estimation algorithms that are described in this section are coordinate-based. Each host is represented by a position in a space. The distance between the hosts represents the two-directional estimated latency. It is difficult and computationally expensive to find coordinates that provide good estimations. If this has happened, the estimations are calculated quickly with one euclidean distance calculation.

GNP ALGORITHM

In 2002, Global Network Positioning (GNP), the first latency estimation algorithm by Zhang *et al* was [7]. It consists of two steps, Figure 2.2 shows the two steps of the GNP algorithm. In the first step a subset of landmarks L from all hosts H are chosen as points of reference. It is normal to have around 20 landmarks. Coordinates are found by minimizing the difference between the real measured latencies and computed distances between the landmarks. In the second step the coordinates of ordinary hosts are determined. To find the coordinates of a host H the sum of the differences between the measured and estimated latencies to all landmarks L is minimized. In both steps the minimization is done with the simplex downhill algorithm. Any other optimization algorithm can also be used.

The complexity of the number of distances that have to be minimized in the first step is $O(|L|^2)$ and the complexity of the second step is $O(|H| * |L|)$. However, in most cases there are only a small number of landmarks. This means becomes $|L|$ becomes negligible in the complexity. The complexity of the whole algorithm will then only depend on step 2 and is $O(|H|)$. Whenever the number of landmarks L is larger, the algorithm might provide higher accuracy, but takes more time to compute. Therefore, a trade-off between the number of landmarks and accuracy has to be made.

NPS ALGORITHM

The Network Positioning System (NPS) latency estimation algorithm is shortly published after the GNP algorithm in 2004 and improves GNP by introducing decentralization[8]. Hosts can serve as landmarks for a base of reference points. This makes landmarks much less critical and less of a bottleneck to the system. If a landmark or connections toward it fail, the system can recover.

In NPS the two steps method of GNP is used but the landmark calculation is changed. Each landmark node computes its own coordinates in multiple steps. In a single step the difference between the measured and computed distance toward other landmarks is minimized. The computed coordinates are then shared with other landmarks. After one second, a new step is started and this repeats until convergence is met. Convergence is achieved if after 3 consecutive iterations a landmark position has not moved by more than one millisecond. The approach can compute the coordinates of 20 landmarks in approximately one minute. The resulting positions are just as accurate as the centralized approach.

VIVALDI ALGORITHM

Vivaldi was published shortly after the GNP algorithm in 2004[9]. It conceptually differs from GNP in that corrections on coordinates are done with Newtonian physics. Nodes move step-wise towards other locations in the map according to the force that is exerted on them. In the model there is a spring placed between each pair of nodes with a rest length equal to the measured latency between the nodes. Every pair of nodes exert a force on each other with a strength equal to the displacement of the spring from rest. The displacement is equal to the difference between the measured and computed latency. The total force on a single node is the sum of all pair forces.

In the simple decentralized Vivaldi algorithm each node simulates only its own movements. In every step nodes obtain the coordinates from- and measures the latency toward other peers. After receiving this information from a single peer, the node directly moves in the direction of the node it communicated with. The size of the move is dependent on the time-step. The time-step size δ determines how long the force exerts on a node.

Because the node moves directly after communication with a single peer, only the displacement of the spring from that particular peer is reduced. To converge toward the right coordinate, nodes continually communicate with multiple peers . Figure 2.3 shows an example of coordinate results after convergence. Nodes that are contacted a lot are more frequently updated because a node updates itself directly after communication. To resolve this a node can favor not recently contacted peers and peers that aren't contacted frequently. Nodes can do this by maintaining a communication history.

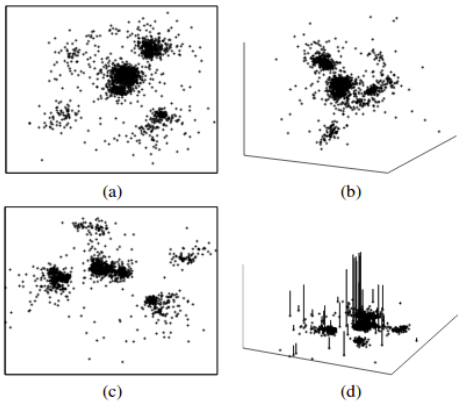

Figure 2.3: The node coordinates results by Vivaldi with the King data set (a) in two dimensions, (b) in three dimensions, (c) with height vectors projected on the xy plane, and (d) with height vectors rotated to show the heights[9].

The main difficulty in Vivaldi is choosing a right time-step size δ. A large δ often results in oscillation with no convergence. A small δ leads to convergence, but this can happen very slowly. In order to solve both problems, Vivaldi varies δ depending on how certain the node is about its coordinates. Large δ values will help the node to go quickly to a position, while small δ values allows it to refine itself. Vivaldi also takes into account the error of the opposing node when changing δ. When the error of the opposing node is high, δ will be lower. With the approach of changing δ during the run of the algorithm, there is quick convergence, low oscillation and nodes with high error have lower impact.

PIC

The Practical Internet Coordinates for Distance Estimation (PIC) published in 2004 is a solution that scales well and does not rely on centralized infrastructure nodes[10]. Any node in the system can act as a landmark if its 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.

New entering nodes determine their coordinates by minimizing the errors in predicted distances to all landmarks. The authors of the paper experimented with several error functions to minimize. The one that performed the best was the sum of the squares of the relative errors. For this error function, the new entering node has to obtain the latencies toward all landmarks. This is done by probing all landmarks before computation.

Three different strategies have been tested to choose a subset of landmarks out of all nodes. Each strategy is tested in different environments with a variable amount of routers. Choosing some landmarks close to the new entering nodes and some landmarks randomly gives the best performance.

To make PIC more secure triangle inequality tests are introduced. In mathematics, the triangle inequality states that the sum of any two sides must be greater than or equal to the length of the remaining side. If an attacker lies about its coordinates or its distance

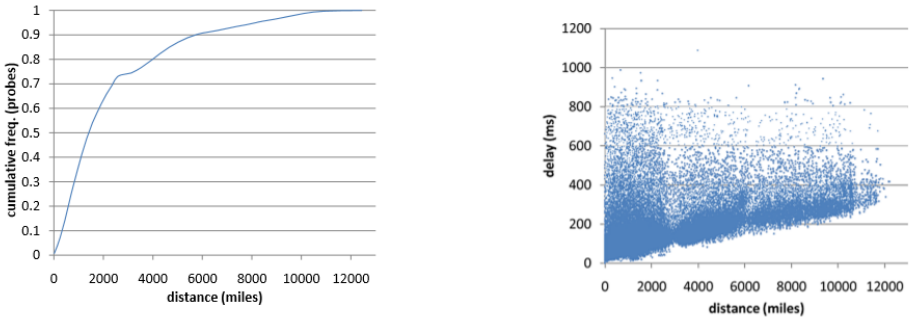

Figure 2.4: Both figures are from the experiments performed on the data from Xbox game consoles by Lee *et al* in 2008. [18] The figure on the left shows the cumulative distribution of the distance between consoles. The figure on the right shows the latencies measured for each distance between two nodes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

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.

LATENCY ESTIMATION WITH GEO-LOCATION

In 2008, Lee *et al* tried to do latency estimation with location data of peers from earth[18]. The IP-addresses of Xbox live game session information for Halo 3 are retrieved and translated towards locations. Using the commercial MaxMind GeoIP City database from June 2007, the authors were able to provide the latitude and longitude for over 98% of the IP addresses. The data set covers over 126 million latency measurements from 5.6 million IP addresses.

It is hypothesized that the geographic distance has a strong correlation with the measured latency between two consoles. The great-circle distance, the shortest distance between two points on a sphere, is used to calculate the distances between two consoles at different geolocations on earth. These distances vary between 0 and 12000 miles and Figure 2.4 shows a cumulative distribution function for these distances. It can be seen that about 14% of the console pairs traversed over 5000 miles.

In the right graph of figure 2.4 the relation between the distance and delay is shown. There is a very strong correlation between the geographic distance and the minimum latency measured between two consoles. Therefore the geography of IP addresses is a useful predictor for filtering out node pairs that have a low latency, but are too far apart for this. Above the minimum value there is a lot of noise. This means there is no strong correlation between the latency and the geographic distance between node pairs on earth.

HTRAE LATENCY ESTIMATION SYSTEM

Htrae is a latency prediction method published by Microsoft in 2009 merging both network coordinate systems (NCS) and earth geo-location approaches. The way it combines both methods works is by geographic bootstrapping. NCS coordinates are initialized in

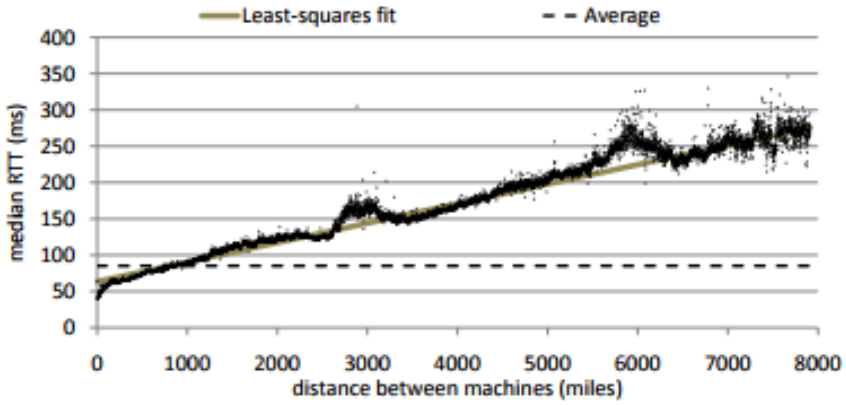

Figure 2.5: The correlation between the distance and latency. The latency data is the median of the data from the Halo 3 players database. The distance data is from MaxMind's IP-to-geo database. There is a clear linear relation between the distance and the median. The slope of the line is 0.0269 ms/mile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

such a way that they correspond to the locations of the nodes in space. With better initial positions, internet latencies can be better predicted.

Figure 2.5 shows the relation between the distance in miles and latencies from the Halo 3 database. In contrast to figure 2.4, the median is taken at each distance which show a linear relation between the distance and median latency. The least-squares fit line is also drawn in figure 2.5. The explained variance percentage of this fit is 97,6%. This is a high explained variance, suggesting 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 a sphere space with spherical coordinates instead of a linear euclidean space to better model the shape of the earth. An uncertainty model is also added that is used to calculate the magnitude of the force to apply when updating coordinates. Uncertainty is defined as the difference between the observed and estimated latencies. The greater the uncertainty, the stronger the force will be.

The Htrae system implements additional things to improve on NCS systems like Triangle Inequality Violation (TIV) avoidance and autonomous systems (AS) correction. Triangle Inequality Violations (TIVs) have an impact on the performance of neighbour selection in P2P systems. TIVs exist due to routing policies and the structure of the internet and will therefore remain in the future. TIVs are removed by skipping the coordinate update if the measured latency is different from the predicted latency by some number δ. In AS correction, inefficient routing paths are corrected. When connecting two random computers via the internet in as many as 40% of times these computers have an alternative shorter routing path. Inefficient routing causes a large delay between two

nodes compared to a more efficient route[11].

2.4. Internet Overlays

In this section we describe internet overlays as examples of existing systems. The literature provides theoretically concepts.

DHash++

DHash++ is a distributed hash table (DHT) that provides network storage [19]. It has high requirements on latency because it is designed for applications that read data most of time. Dhash is built as a layer over the Chord distributed lookup system. Chord provides a scalable lookup service that maps keys of the DHT to nodes. A node will have the ability to choose from a set of node to complete a lookup operation. Therefore it is prefered to choose the node with the lowest latency. For instance, a data item might be replicated over multiple nodes to create redundancy. It is beneficial retrieve the data from the node with the lowest latency to save time. To know the latencies between nodes the Vivaldi latency estimation algorithm is used.

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 [20]. 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. The measured latencies are used to select a bin. The latencies 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 reduces the latency and performance in the network overlay construction.

3

Problem Description

This chapter describes the problems faced when creating a low latency overlay. The main problem is to make efficient latency estimation algorithms. In order to give low latency peer introductions to each other, peers should be able to estimate latencies well. Furthermore, the low latency overlay should not be able to be taken down easily. Finally, the low latency overlay should be resilient against attacks like the eclipse and sybil attack.

3.1. Performance of latency estimation algorithms

The first requirement of the low latency overlay is that the latency estimation algorithms should be computationally and memory efficient with a large number of peers N in the P2P network. Nodes can run other programs in the background and might not have the computation power available to run the P2P application with an inefficient algorithm. If the algorithm computation takes too long, the P2P application maybe blocked. The node will not be able to respond to communication and the user experience will become bad. With no response to communication, the latency of a peer toward the blocking node increases.

The algorithms should also be efficient in memory- and bandwidth usage. The number of latencies stored in memory and sent over the internet can become large as more are obtained from other peers. If peers maintain and share all the latency information they ever received, the space complexity is $O(N^2)$ where N is the number of peers in the network. With millions of peers N in the network, the memory usage can become Gigabytes. The GNP algorithm requires such amounts of memory because it also has space complexity $O(N^2)$. Therefore, a latency estimation algorithm has to be used that reduces the amount of latencies sent to other peers and store in memory. Also, choices have to be made about which latencies to remember and to send toward other peers. Such choices imply an information loss that could decrease the quality of predictions.

3.2. PEER DISCOVERY

A second requirement is that peers should be able to communicate with each other in both ways even though some peers are behind a Network Address Translation (NAT) box. Figure 3.1 gives an overview of NAT. To enable communication in both ways, NAT boxes should be punctured and a peer discovery mechanism is required. In a peer discovery mechanism peers are introduced to each other and new connections are set up. Low latency peers should be introduced. In this way peers will automatically make connections toward low latency peers. 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.

Connections between peers can terminate because punctured holes in NAT boxes are closed when communication does not occur anymore. Therefore connections toward low latency peers can also terminate. The peer discovery mechanism should try to maintain the connection toward low latency peers and reconnect if needed.

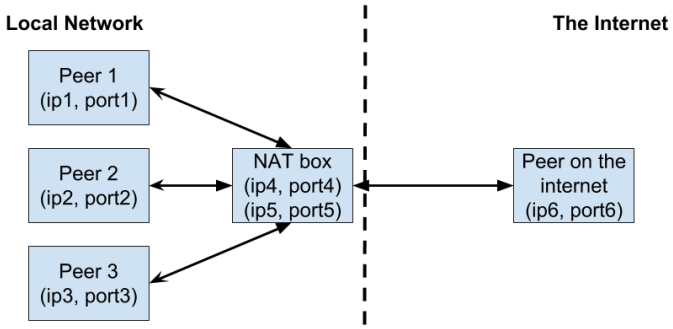

Figure 3.1: Network Address Translation (NAT). The NAT box has two IP, port combinations. $(ip4, port4$ is available on the local network and $ip5, port5$ is available on the internet.

3.3. SECURITY REQUIREMENTS

SYBIL ATTACK

In the Sybil attack an adversary creates multiple pseudonym peers that flood or spam the P2P network with false information. It is hard to solve the sybil attack 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 using pseudonyms. An adversary can order multiple pseudonyms to send a lot of peer introduction requests to a target peer. The target peer is then completely occupied by handling all the introduction requests and is taken down. It is unable to respond and send messages to normal non-pseudonym peers.

By letting pseudonyms collude with each other, an adversary can subvert the reputation system of P2P applications. Peers could gain a false high reputation. Reputation systems are important in a lot of P2P applications. P2P file sharing applications make use of reputation systems to incentivize peers to share files with other peers instead of

just using the system. The reputation is then defined as the amount of data shared with other peers. In the TrustChain online currency application a peer will be denied service when its reputation is too low [21] [22].

ECLIPSE ATTACK

Eclipse attacks have large implications on P2P networks. An attacker can gain partly or complete control over the data that is received by a victim node. This is achieved by manipulating the connections of the victim and its neighbours. Adversaries can introduce attacking nodes to peers. If attacker nodes control a large part of the connections they can "eclipse" victims. When this happens, messages that attempt to reach the victim are dropped or rerouted. If all connections of a peer are from attackers, they will gain full control over all traffic toward the victim[23].

The eclipse attack is a very powerful and generic attack. We will provide several examples in cryptocurrency applications where eclipse attacks are used and have direct financial consequences. In most cryptocurrency systems a decentralized blockchain is used where transactions are stored in blocks. Nodes use computational proof-of-work to reach consensus on transactions. These nodes are called miners and by providing computational power to discover blocks they receive a mining reward. The following attacks on cryptocurrencies use eclipsed nodes as a powerful building block[24].

1) Engineering block races

A block race occurs when two miners discover blocks at the same time. One of these miners receives the reward for that block and it will become part of the blockchain. The other miner will be ignored, create an "orphan" block and receives no reward. Attackers can forge block races by holding back blocks mined by eclipsed miners. When holding back blocks, a non-eclipsed miner will discover them first and receive the reward.

2) Splitting mining power

By eclipsing a large part of miners from the rest of the network, the 51 % attack becomes easier. In this attack, the attacker gains control over 51 % of the mining power in the network. The attacker is therefore able to create a separate blockchain. The mining power reduction of eclipsed miners can be made less detectable by eclipsing them gradually or intermittently.

3) Selfish mining

The attacker can decide to eclipse certain miners and withhold them from discovering blocks. Other miners controlled by the attacker have then higher chances of discovering a block with less competition. In this attack, a is the fraction of nodes used to eclipse other miners. The fraction of nodes b is used for honest mining. When a increases, mining becomes easier for the fraction b. Another way to obstruct eclipsed miners is to only give them a limited view on the blockchain.

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. This happens in systems where it is inappropriate to wait 5-10 minutes before a transaction gets confirmed. For instance, the retail service BitPay or gambling sites like Betcoin a transaction needs to be confirmed immediately. In the attack, coins are double spend

to the merchant. The attacker first eclipses the merchant and starts a transaction T. Because the merchant is eclipsed he can never tell the blockchain about T. For the outside world the transaction T did not happen. The attacker double spends the bitcoins with another transaction T' and rewires the money back to himself.

5) N-confirmation double spend

In a system with N-confirmation transactions, the attacker can also double spend coins from a merchant. 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 starts a transaction T with the merchant but only sends T 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. After this, the blockchain view V of the non-eclipsed miners is send toward the merchant and the eclipsed miners. Therefore, the block-chain view V' containing T is orphaned and T never happened.

4

Overlay Design

In this chapter we will focus on how the low latency overlay is designed. First the algorithms are described that estimate latencies between peers. The algorithms enable low latency introductions. Incremental algorithms make the latency estimators computationally and memory efficient. How the low latency overlay is designed into Tribler is described in the second section.

Dispersy

The low latency overlay is build on top of Dispersy. It consists of 1200 lines of code and can be downloaded open source from Github [1]. 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%. Dispersy allows implementing communication protocols on various machines in a distributed system.

The most important terminologies are explained to clarify how Dispersy is implemented[25] [13] .

Peer A peer is a device running a Dispersy instance.

Node A node is equal to a peer.

P2P network A Peer-to-peer (P2P) network is a network consisting of peers.

Message A message is sent between peers for communication.

Payload The payload defines the content of a message. Every message has a unique payload design. It defines what is sent in the message.

Latency A latency is the round-trip time of a message between two peers.

Community A community is a subclass of Dispersy with the code implementation of an overlay. The low latency overlay has its own community class called LatencyCommunity.

[1] https://github.com/basvijzendoorn/tribler/

Neighbourhood Each node maintains a list of peers to which it has an active connection. This list is called the neighbourhood. The node can communicate with peers in the neighbourhood and vice versa.

Peer discovery Procedure where nodes find other peers to connect to. When a new connection is set-up between two peers they add each other to their neighbourhood.

4.1. LATENCY ESTIMATION ALGORITHMS

The focus is on online incremental algorithms to create a computationally and memory efficient latency estimator (figure 4.1). An online incremental algorithm does not require the total input of all the measured latencies at once. Instead, the input is given over time in steps. At each new step, new input e_n is given and a new intermediate solution s_n is calculated. The variable n denotes the step number. One step should not require much computational power. Eventually, the incremental algorithm will converge to a final solution[26] [27].

The 5 latency estimation algorithms are explained in the following paragraphs. Each algorithm is given a unique name. Using incremental algorithms gives a computational benefit. Splitting the problem into different parts may reduce the accuracy. Incremental algorithms have incomplete information because future measured latencies cannot be taken into account. To cope with this, the relation between past- and newly added information has to be analyzed. The latency estimation algorithm may reuse information added in the past.

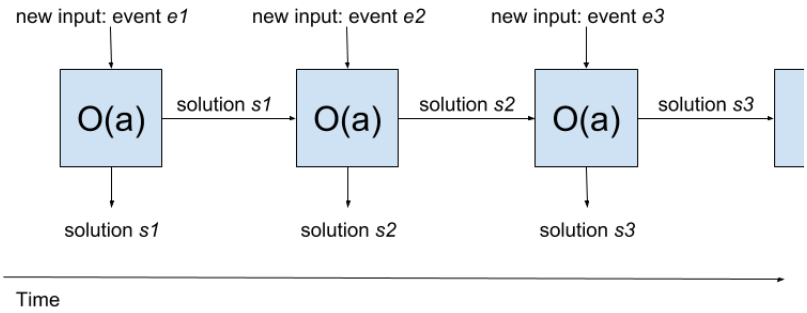

Figure 4.1: Overview of an online incremental algorithm. At each step a new input event e is added to the algorithm. A small computation with $O(a)$ complexity is used to calculate a new solution s. The new solution is used in the next step of the algorithm.

NAIVE ALGORITHM

The naive algorithm is coordinate-based and an error function is minimized. The error function is equal to the difference between the estimated- and real measured latencies. It assumes that there are n hosts in the system. These hosts H are coordinates in a two-dimensional geometric space S. Hosts are equivalent to peers and nodes. Every host $H_i \in H$ has its own coordinate C_i^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 latencies are measured and collected by hosts. The function $md(H_1, H_2)$ equals the measured latency between hosts $H_1, H_2 \in H$.

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

$$f_{obj}(C_1^S, C_2^S, ..., C_n^S) = \sum_{i,j \in \{1,2,..,n\} | i>j} = \epsilon(d(C_i^S, C_j^S), md(H_i, H_j))$$

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

$$\epsilon(d(.), md(.)) = (d(.) - md(.))^2$$

The minimization algorithm used is L-BFGS-B. It uses less time compared to the simplex downhill algorithm while the accuracy is remained. L-BFGS-B is able to handle large number of peer coordinates since it is especially suited for large input spaces. The number of iterations in L-BFGS-B can vary. With more iterations, L-BFGS-B might have an improved minimization, but more computation time is used.

The complexity of L-BFGS-B is $O(m * n^2)$ where m is the number of error function calls and n the number of hosts in the system. P2P applications are not allowed to distinguish between landmark and non-landmark nodes used in the GNP algorithm. Therefore, no computational optimizations based on central components can be applied. Since every pair of coordinates and their representing hosts are used in the objective function, the complexity of one call is $O(n^2)$. The squared relationship implies that with large n, the algorithm may become computationally too expensive. In large P2P networks, n can easily be in the order of 100 000 nodes.

SIMPLE INCREMENTAL ALGORITHM

The simple incremental algorithm (Inc) only updates the coordinates of new entered peers P_{new} to the neighbourhood. It is similar to the naive algorithm: It also contains a two-dimensional geometric space S where every host $H_i \in H$ has its own coordinate $C_i^S \in C$. The distance function $md(H_a, H_b)$ equals the measured latency between two hosts $a, b \in H$. The function $d(C_a^S, C_n^S)$ equals the euclidean distance between the two coordinates representing hosts $a, b \in H$. These definitions apply to all other incremental algorithms described in this chapter. The way the coordinates are calculated differs in each algorithm.

In the simple incremental algorithm "Inc", only the corresponding coordinates C_a^S of each peer $p_a \in P_{new}$ is updated by minimizing its error function. Peers measure the latencies toward their neighbours and remember the latencies measured toward past neighbours. The set L contains all the measured latencies between peer a and its neighbours and past neighbours. For each latency $l \in L$ there are two peers p_1 and p_2 between which the latency l is measured. The collection of all these peers minus peer $p_a \in P_{new}$ is called P_{sub} with corresponding coordinates C_{sub}. For each of the peers $p_n \in P_{sub}$ its corresponding coordinate $C_n^S \in C_{sub}$ is looked up from memory or created. Whenever there is a peer $p_n \in P_{sub}$ with no corresponding coordinate in C_{sub}, its initial coordinates $C_n^S \in C$ are created by taking two draws from a uniform distribution from 0 to 1.

The corresponding coordinate $C_a^S \in C$ of the peer $p_a \in P_{new}$ 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:

$$\epsilon(d(.), md(.)) = (d(.) - md(.))^2$$

The minimization is done with the L-BFGS-B algorithm like in the naive algorithm. The total complexity of one step in the "Inc" algorithm is $O(m * |L|)$ where $|L|$ is the size of the number of latencies measured by one peer and m is the number of error function calls. The complexity of one minimization function call is $O(|L|)$. As time progresses, $|L|$ increases because more peers are contacted and latencies are measured. The minimization function is called for each $p \in P_{new}$ every time the "Inc" algorithm is started. However, the size of P_{new} is negligible and thus not added to the total complexity.

4

INCREMENTAL ALGORITHM WITH R RANDOM REPEAT

The Incremental algorithm with R random repeat extends the "Inc" algorithm by updating the coordinates of past calculated peers. This algorithm is called "RandomRepeat". When the calculating the extension, 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 in the "Inc" algorithm. The algorithm of the extension equals:

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:

$$\epsilon(d(.), md(.)) = (d(.) - md(.))^2$$

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

The complexity of the extension is $O(m * R * |L|)$. The minimization function is called for R times extra. Various numbers for R can be chosen to observe the impact on the computation time and accuracy of the latency estimator.

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 is implemented to improve the accuracy of the R random repeat algorithm. This algorithm is called "RepeatStructured" or "Repeat". 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 and none are updated more frequently. Each coordinate of C is numbered. When C increases the new coordinates are given a new number in incremental order. 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 $R \subseteq C$ 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 numbers of C. The complexity of this algorithm is the same as the R random repeat version as the coordinates of R nodes are updated with the same minimization function. Thus the complexity is $O(m * R * |L|)$.

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. TIVs are described in chapter 2. This algorithm is called "TIVPrev". In the extension algorithm peers that are part of a TIV are ignored. Therefore the coordinates and latencies towards these peers are ignored in the minimization functions of both the "Inc" and "Repeat" part.

To estimate which latencies are part of TIVs, the "prediction error" is calculated for every latency that is measured in the past. For every latency $l \in L$ and peer pair H_1, H_2 of l the following prediction error is calculated:

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

TIVs are prevented by ignoring the three latencies with the largest prediction error. These latencies are not used in the minimization calculations.

The total complexity of the extension algorithm remains $O(m * R * |L|)$. The TIV prevention requires a sorting of the latencies with merge sort. This has complexity $O(|L|log(|L|))$. However, the sorting happens before the calculation of the minimization function and is therefore negligible.

4.2. IMPLEMENTATION INTO TRIBLER

This section describes how the low latency overlay is implemented into Tribler. How the peer discovery mechanism works is described first. Folowed by how the low latency overlay obtains latency information. Finally is described how the low latency overlay introduces peers to other peers. An overview of the settings of the overlay is presented in table 4.1 at the end of the chapter.

DISPERSY PEER DISCOVERY AND NAT PUNCTURING

The design of the current peer discovery mechanism is explained[25] [13]. In the current implementation a peer introduction request- and response mechanism is build. This mechanism requests another peer for an introduction and then a response is given. The result is a list of peers called the candidate list or neighbourhood that each node maintains. The peers in the candidate list are called the neighbours of a peer. Data can always be exchanged between two peers in the candidate list. This implies that if peer A has

peer *B* in its candidate list then peer *B* also has peer *A* in its candidate list. Both peers *A* and *B* assume the role of client and server and therefore the NAT firewall of one of the peers has to be punctured.

There are four phases in the current peer discovery mechanism of Dispersy. These four phases represent one step. A new step is taken in an interval of seconds. The number of seconds of the interval can be set in the TAKE_STEP_INTERVAL setting. Multiple steps are called a walk, and by walking each peer discovers its neighbourhood. The TAKE_STEP_INTERVAL setting is chosen to be 5 seconds by default in Dispersy. The four phases are shown in an overview in figure 4.2.

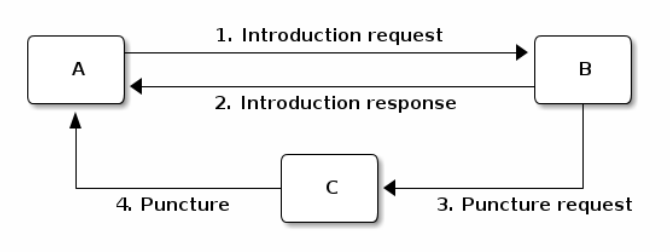

Figure 4.2: Overview of a step in peer discovery of Dispsery.

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.

The NAT puncturing is integrated in the peer discovery mechanism. It works by sending puncture messages to other peers. 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 puncture message toward peer *A*. When doing this peer *C* opens a port in its own firewall such that peer *A* can send a response. The two-way communication requirement is complete if peer *A* can also communicate to peer *C*. In the second step of a peer discovery step peer *B* also sends the address of peer *C* to peer *A*. By knowing this address peer *A* can send messages toward peer *C*. Peer *C* receives these messages through its punctured hole. The NAT puncturing enables peers to communicate with each other without having to worry about the NAT firewalls.

DISPERSY NODE SELECTION

To prevent against eclipse attacks a node selection policy for step 1 is implemented in Dispersy[25] [13] . The policy selects a node from the neighbourhood to send an introduction request. The candidate list is divided into categories and Nodes are selected with pre-defined rules. These categories are:

I) Trusted nodes

II) Nodes we have successfully contacted in the past

IIIa) Nodes who have contacted us in the past by sending an introduction-request.

IIIb) Nodes that have been introduced by another node.

A node which responded to an introduction request is put in category II. The introduced node that was included in the introduction response is put in category IIIb. Whenever a node receives an introduction request the sender is put in category IIIa. The trusted nodes category consists of a special list of pre-defined nodes.

When selecting a node, a choice is made from the categories with pre-defined probabilities. The trusted node category I is chosen with a probability of 1%. Category II is chosen for 49.5% of cases and category IIIa and IIIb are both chosen 24.75%. Some firewalls close inactive connections after a certain timeout. When a connection is closed, both nodes cannot communicate with each other anymore. Therefore, after a category is chosen, the node with the most recent interactions is selected.

Nodes in the neighbourhood are removed if the probability of a closed punctured hole is high. Most NAT boxes close punctured holes after a certain period of inactivity. NAT boxes can close punctured holes after about 60 seconds of no communication. Therefore peers in category II and IIIa are removed from the neighbourhood after 55 seconds and introduced nodes from category IIIb are removed after 25 seconds. This is a problem becuase low latency peers could be removed from the neighbourhood. Countermeasures should be taken to keep them.

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, they are only added to category III and not to category II. Because category II has a 49,5% selection probability the adversary nodes will not always be selected. Therefore, the node selection policy mitigates an eclipse attack. It is for an adversary with a lot of resources still possible to do an eclipse attack. The trusted node group, with a selection probability of 1%, is added to give additional protection. Whenever a node selects the trusted group the neighbourhood is completely reset and all adversary nodes in the candidate list are automatically removed.

4.2.1. OBTAINING LATENCY INFORMATION

In this section two mechanisms are explained to measure and obtain latency information: the ping-pong mechanism and the crawling mechanism. The ping-pong mechanism measures and shares previously measured latency information. The crawling

mechanism is an extra feature added to the overlay in order to rapidly share latency information between peers. In contrast to the ping-pong method this mechanism is bandwidth inefficient and not enabled by default.

To make the overlay more memory and computationally efficient only a selected number of latencies are stored. Latencies toward peers that are closest to the own node will be remembered. The own node is the peer obtaining the latency information. After a latency from peer $a \in P$ to $b \in P$ is obtained, all peers Q with stored latencies toward $a \in P$ are determined. The list Q is sorted in ascending order of the peers that are closest to the own node in the map of the latency estimation algorithm. The latencies toward a top number of the closest peers to the own node are stored. This number can be set in the REMEMBER_LATENCIES setting.

PING-PONG MECHANISM

The ping-pong mechanism has two purposes. At first, the latency between peers is measured. Secondly, previously measured latencies stored by the Tribler instance are shared to other peers. The ping-pong mechanism starts in an interval number of seconds by every peer. The interval can be set in the PING_TIME_INTERVAL setting. By default the value of the PING_TIME_INTERVAL setting is set to 2 seconds. This setting frequently updates the latency information of peers with limited bandwidth usage. Frequent updating is important for two reasons. First, the latency to newly entered peers in the neighbourhood should quickly be measured to use this information in the latency estimation algorithms. 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 too much.

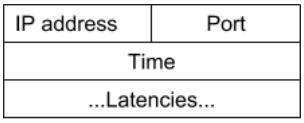

Figure 4.3: Ping payload.

Latency measuring and sharing works as follows. When the ping-pong mechanism is activated every peer sends all its neighbours a ping message. The peers which receive the ping message return a pong message. The time when the ping message was send is stored to compare later with the arrival time of the pong. After the pong has arrived, the difference between the send and return time is calculated and the latency toward the neighbour is obtained. The payload format of the ping message is shown in figure 4.3. The ping message contains the IP and port of the peer sending the ping and the time of sending plus 10 previously measured latencies. These latencies always contain the peer who activates the ping-pong mechanism. They are are not between two arbitrary peers. In 30 seconds with 15 ping messages and default settings are $10 * 15 = 150$ measured latencies send toward the other peer. In practice, the ping-pong mechanism will first send the first 10 measured latencies, then the second 10 etc. When all measured latencies are send the ping-pong mechanism will start again from the beginning and send the first 10 measured latencies again.

The ping-pong mechanism also shares latencies from two arbitrary peers. These are added to the 10 previously measured latencies. In a ping message, from $share\_p$ peers are $share\_l$ latencies randomly chosen to send. Thus, in total $share\_p$ times $share\_l$ latencies are additionally send in a ping message. By default $share\_p$ is equal to 2 peers and $share\_l$ is 10 latencies. With these values bandwidth is saved, but over a longer period, still a significant amount of latencies are shared. The variables $share\_p$ and $share\_l$ can vary to share more or less latencies in the ping message.

With default settings the size of a ping message equals $30 * 70 + 70 = 2100$ bytes. The bandwidth consumption of ping messages cannot be exactly calculated. It depends on the amount of latencies send in each ping message and the byte cost of one latency (see figure 4.3). A latency is transferred as a string and therefore varies in size. On average one latency has a size of around 70 bytes. With default settings 30 latencies are send in each ping payload. Another 70 bytes are reserved in the message for the IP address, port and time that are also send in the ping message.

IP address	Port	Time

Figure 4.4: Pong payload.

After receiving a ping message, a pong response is given back. Figure 4.4 shows the payload format of the pong response message. The pong payload contains the IP and port of the responder. The time from the ping message is copied to the pong payload. On average the size of the pong payload equals 70 bytes.

CRAWLING MECHANISM

When the crawling mechanism is active, in an interval of seconds a crawl request is sent to all neighbours of a peer. The interval can be set in the CRAWL_TIME_INTERVAL setting. The standard CRAWL_TIME_INTERVAL is 15 seconds. Figure 4.5 shows what happens when the crawl request is sent. The mechanism consumes a vast amount of bandwidth and is therefore not activated by default. Each peer that receives a crawl request forwards it to other peers. It also sends all its obtained latencies back to the requesting peer. By forwarding latency requests more peers are reached. After a latency request message is received, the receiver returns its obtained latencies.

The forwarding construction is built, because peers can only communicate with their neighbours. Peers cannot directly send back the latency information to the initiator of the crawl request. This would require puncturing the NAT firewall of the initiator.

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. It is increased each time the message is forwarded. The peer that sends the first crawl message sets the hop variable to 0. If the hop count exceeds the MAXIMUM_HOP_COUNT setting variable the message is not forwarded anymore. This limits the traffic on the P2P network. The relay list contains a list of unique variables that is used when sending the response latency message. It is used in the forwarding mechanism in order to know to which peer the latency response should be forwarded back. The byte cost of one message varies and depends on the size of the relay list. In default settings MAXIMUM_HOP_COUNT is equal to two

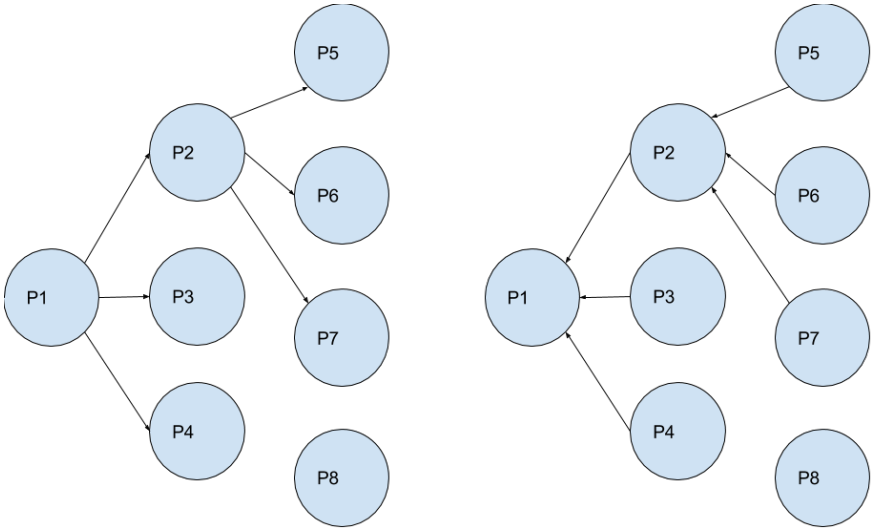

Figure 4.5: The left figure shows what happens when peer P1 sends a crawl request. The crawl request is forwarded to its neighbours P2, P3 and P4. These peers forward the crawl request to their neighbours. In the right figure the latency response message is shown. All peers send back their latency information to the peer from who they received the crawl request message. These peers forward this response message back until the original crawler P1 is reached.

and the relay list has a maximum size of around 10 bytes. In default settings, another 50 bytes are added to the byte cost for the IP address, port and hops variables. Therefore the total byte cost is around $10 + 50 = 60$ bytes.

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

Figure 4.6: Overview of crawl request payload.

The latency response message payload is shown in figure 4.7. The IP address and port contain the address of the peer returning the latency response message. The relay list is used by the forwarding mechanism. The latencies in the payload are all the latencies a peer knows. They are send backward toward the initiator of the crawl. The latencies are stored in a dictionary for transfer. The two addresses of the peers of the latency are keys and the latency between the two peers is the look-up value of the key. The dictionary is serialized to a string for easy transfer in the payload. Since all known latencies of a peer are send in the response payload its byte cost is impossible to estimate. The byte cost can become large as the number of obtained latencies increases.

THE FORWARDING MECHANISM

The forwarding mechanism consists of two parts. In the first part crawl request messages are forwarded to reach many peers. In the second part latencies are returned by all peers that received a request message. The returned latencies are sent back to the original

IP address	Port
...Relay list...	
...Latencies...	

Figure 4.7: Overview of latency response payload.

requester with the same route as the requests were send. Both the crawl request and latency response message contain a relay list that is used in the forwarding mechanism.

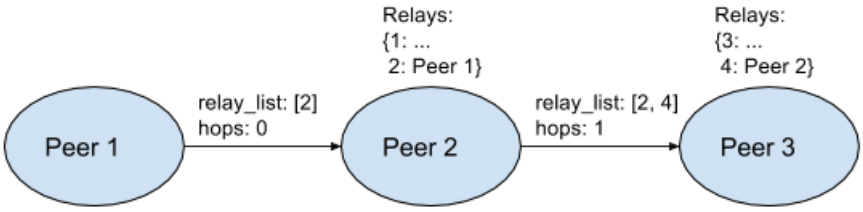

Figure 4.8: Peer forwarding scheme. In each communication line the *relay_list* is given. Each peer adds a new relay id to the *relay_list*. When a peer receives a message the *relays* dictionary is updated with the last added *relay_id* as key and the peer which sends the message as result. The hop count is also increased at each forward.

In the first part of the mechanism the crawl request messages are forwarded to other peers (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 *relays* dictionary. The last *relay_id* on the relay list is used as a key in the *relays* dictionary. With the *relay_id* key, the peer can know to which address the response with latencies has to be sent back in the second part. The unique *relay_id* is created using the global time variable in dispersy plus the address of the peer. 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 in combination with the node which sends the message and the community id. If *relay_id* is not unique the key in the *relays* dictionary might be overwritten and the response message could arrive at the wrong peer.

In the second part the latency responses are sent back to the peer that initiated the crawl (figure 4.9). After arrival of a latency response message, the last *relay_id* of the *relay_list* in the message is popped and used as a key in the *relays* dictionary. As can be seen in figure 4.9, the key returns the address of the next peer in the forwarding chain from the *relays* dictionary. The item corresponding to the dictionary key is deleted when the latency response is forwarded back. It is of no further use and by deleting it the crawl mechanism stays memory efficient.

Sometimes the peer to which the latency response has to be forwarded back is no longer in the neighbourhood. In that case the crawl initiator will never retrieve the latencies and has to wait before the crawler is actived again.

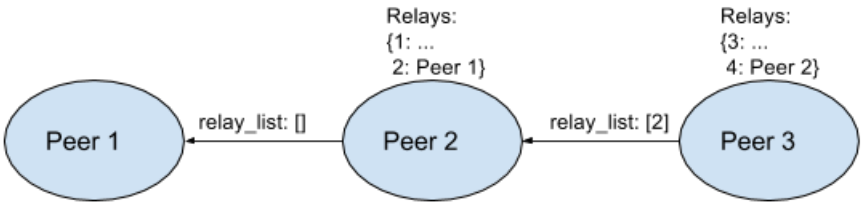

Figure 4.9: Peer forwarding scheme upon return. When the hop count exceeds the hop count limit the latencies are returned. The peer pops the last $relay\_id$ from the $relay\_list$ and uses this id to lookup the peer to backward the latencies to in the $relays$ dictionary.

4.2.2. LOW LATENCY OVERLAY

A few changes are made in the peer discovery mechanism to enable low latency introductions. In the second phase of a step the introduction peer C is chosen from the neighbourhood of peer B to introduce to peer A. It is chosen in such a way that the latency between peer A and the introduction peer C is low. Peer B uses the latency estimation algorithm to know which neighbours of B have a low latency with A. Peer B knows this by calculating the distances between his neighbours and A in the map that was constructed by the algorithm. Load balancing is set up to avoid that the same peer is introduced multiple times. This is important, otherwise the introduction has no extra value and becomes meaningless. The load balancing is implemented by choosing to introduce one of the top BALANCE_INTRODUCTIONS peers with the lowest estimated latency toward A. By default the BALANCE_INTRODUCTIONS setting variable is equal to 3.

Due to NAT timeouts, peers are removed from the neighbourhood. This has a large implication on the quality of introductions and walking requirements. It is not possible to introduce peers that are not in the neighbourhood because their firewalls are not punctured. When not all peers in the network can be introduced, it is inevitable that low quality peers are introduced with high latency.

The peer discovery mechanism is changed to keep low latency peers in the neighbourhood. It is changed to always prefer to take steps toward one of the top 10 lowest latency peers in the neighbourhood. NAT-timeouts do not occur because there is still communication between the peers. The TAKE_STEP_INTERVAL setting is lowered to 1 second to further incentivize preservation of the low latency peers. Load balancing is also set up in the walking mechanism by repetitively choosing one of the top 10 lowest latency peers. By default the above described new walking mechanism is set up, but the old mechanism can also be used to add additional randomness. The old mechanism is described in the Dispersy node selection paragraph. In the "OldMechanism" setting, in 50% of node selections the old mechanism is used and in the other 50% one of the top 10 lowest latency peers is chosen. The randomness of node selection in this setting dampens the eclipse attack.

The quality of introductions can increase with the cluster effect. In the map of the latency estimation algorithm peers can become clustered. This means these peers are close together and have a low latency toward each other (Figure 4.10). Probabilities are high that peers in such clusters have each other in their neighbourhood. Therefore,

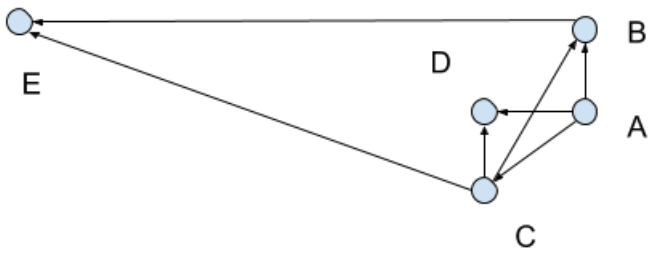

Figure 4.10: Schematic overview of the cluster effect with a map result of the latency estimation algorithm. Peers A, B, C and D are clustered together and have each other in their neighbourhood. They will introduce members of the cluster to each other. Peer E is less likely to be introduced because it is further away from peers A, B, C and D and it does not belong to the top lowest latency peers of cluster members.

members of the cluster are more likely to be introduced and stepped forward to. The clusters can increase the quality of introductions if peers already know their lowest latency neighbours in the network. They can compensate for the negative effects of the limited available peers in the neighbourhood due to NAT timeouts.

Newly introduced peers are stored in P_{new}. They are used as input to the latency estimation algorithm. The incremental algorithm can only benefit from the newly introduced peer if latencies toward that peer are obtained. The latency information is obtained after introduction with the ping-pong mechanism or the crawling mechanism. In the default setting the ping-pong mechanism is activated every 2 seconds. This means that in the worst case scenario the latencies toward peers of P_{new} are obtained two seconds after introduction.

One of the latency estimation algorithms is run in the background. This is implemented by calling two functions in intervals. The first function call updates the coordinates of new discovered peers. The coordinates of the peers of P_{new} are computed in an interval amount of seconds. The interval can be set with the COORDINATE_TIME_INTERVAL setting. When no latencies were obtained from a peer of P_{new}, no coordinate calculation can be done. These peers will stay in P_{new} until latencies are obtained. Other peers are removed from P_{new}. By default COORDINATE_TIME_INTERVAL is chosen to be 3 seconds. With an interval of 3 seconds, the overlay should have enough time to obtain latencies of a peer that was recently added to P_{new}.

The second interval updates previously calculated coordinates when the "Repeat-Structured" and "TIVPrev" algorithms are used. Every second a number of previously calculated coordinates are updated. The number can be set with the NUMBER_OF_REPEATS setting. By default NUMBER_OF_REPEATS is set to 1 coordinate. This is provisional, because the repetition is done every second. With default settings, 10 coordinates are updated after 10 seconds.

Name	Description	Options	Default setting
Latency estimation algorithm	The latency estimation algorithm used in the overlay.	Inc, Repeat, TIV	TIV
TAKE STEP INTERVAL	Interval time between steps in the peer discovery mechanism.	Integer (seconds)	1
COORDINATE TIME INTERVAL	Interval time at which a new incremental step in the latency estimation algorithm is taken.	Integer (seconds)	3
NUMBER OF REPEATS	Number of coordinates that are updated every second for the "TIV" and "Repeat" algorithms.	Integer (coordinates)	1
Remember latencies	Number of latencies to remember for one peer after new latencies are obtained.	Integer (latencies)	100
Ping-pong mechanism	Mechanism to measure and share latencies.	On/off	On
Share extra latencies in ping-pong mechanism	Share extra latencies obtained from other peers in ping-pong mechanism	On/off	On
PING TIME INTERVAL	Interval at which the ping-pong mechanism is activated	Integer (seconds)	2
Crawling mechanism	Mechanism to obtain latencies from other peers.	On/Off	Off
CRAWL TIME INTERVAL	Interval at which the crawling mechanism is activated	Integer (seconds)	15
MAXIMUM HOP COUNT	Maximum number of times crawler request is forwarded	Integer	1
BALANCE INTRODUCTIONS	Number of peers to load balance in introductions.	Integer	4
Step toward low latency peers	If on, always take steps toward the top 10 low latency peers in the neighbourhood. If off 50% of times the normal dispersy mechanism is used.	On/Off	On

Table 4.1: Overview of low latency overlay settings.

5

EXPERIMENTS

In this chapter a description is given of the experiments that test the low latency overlay. We will first describe the performance metrics used in the experiments. After that the experiments are described.

5.1. PERFORMANCE METRICS

COMPUTATION TIME

The *computation time* performance metric measures how much time Tribler is consecutively computing a part of the latency estimation algorithm. It can be easily calculated by taking the time difference of the time before and after the computation. When the computation is spread in some algorithms, each computational part that requires a separate calculation is measured as such single consecutive computation. This happens in the "Repeat" and "TIVPrev" algorithms where coordinates are updated.

RELATIVE ERROR

The *relative error* metric measures the overall estimation performance of the algorithm. In all algorithms the estimated latency between peers a and b is equal to the euclidean distance between the two corresponding coordinates in a geometrical space S. For each estimated latency the local 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 the estimated latency and measured latency are equal while a value of one would imply that the estimated latency is larger by a factor of two. The relative error performance metric is the average of all local relative errors between every peer pair in the map. If the map contains n peers, there are n^2 peer pairs and local relative errors. Thus, if the map contains 500 peers, the average of $500 * 500 = 250000$ local relative errors is taken.

RANKING ACCURACY

The *ranking accuracy* metric measures the predictive quality of the latency estimation algorithms. It is a good metric to evaluate the selective performance. In other words, how well can the algorithm make predictions on what peers have a low latency toward one other. Suppose we have a set of P peers that form a P2P network. The ranking accuracy metric is the average of the local ranking accuracies of all peers of P.

The local ranking accuracy of a peer $a \in P$ is calculated as follows. The subset $B_a \subseteq P$ are all the peers that were introduced to $a \in P$. The latencies from a to all peers of B_a are estimated and sorted in the ascending list E_a. Each element of E_a is a tuple (b, e) where $b \in B_a$ and e is the estimated latency from peer a to $b \in B_a$. The measured latencies from peer a to all peers of B_a are also sorted in an ascending list M_a. Each element of M_a is a tuple (b, m) where $b \in B_a$ and m is the measured latency from peer $a \in P$ to $b \in B_a$. Because we are only interested in the accuracy of the lowest latencies, only the top 10% lowest latency tuples in the sorted lists E_a and M_a are used. These tuples are stored in the lists $E10_a$ and $K10_a$. The local ranking accuracy of peer $a \in P$ is defined as the percentage of peers that is both in the tuples of $E10_a$ and $K10_a$.

5

TOP 10 LOWEST LATENCY PEERS

The *top 10 lowest latency peers* metric measures the quality of the neighbourhood. The metric is determined for a node $a \in P$ in the following way. First is calculated which 10 peers have the lowest latency toward node a. This is done by using the latency data-set. The number of these peers that is in the neighbourhood of node a equals the performance metric.

QUALITY OF INTRODUCTIONS

The *quality of introductions* metric measures how well introductions provide low latency peers for the node receiving the introduction. The calculation of the metric is explained with an example. Suppose there is a P2P network of P peers where node a introduces peer b to c. The peers a, b and c are random chosen peers of P. The metric is calculated by peer c. The latencies between all peers P and node c are sorted in ascending order. These latencies are extracted from the data-set. The quality of introductions is equal to the position in the sorted list that corresponds to the latency from peer b to peer c. With this definition, a value of 1 would imply that the lowest latency peer is introduced to node c. A value of 20 would imply that the twentieth lowest latency peer is introduced to node c.

5.2. ALGORITHM VALIDATION EXPERIMENT

The goal of the algorithm validation experiment is to test the behaviour of the algorithms. The experiment is not executed on a distributed system and is not meant to test the overlay.

SETUP

The algorithms, "Naive", "Repeat", "TIVPrev" and "Inc" described in chapter 4 have been implemented and are validated on a single machine. The algorithms were run on a computer with a dual core 2.8 GHz processor. In the experiment, the peer discovery mechanism is simulated by incrementally adding peers. In the beginning of the experiment

there are 0 peers in the network. Every time a new peer is added an incremental step of the algorithm is taken. Complete information over the latencies toward all other peers is provided. After every incremental step the computation time of one incremental step the ranking accuracy, and the relative error are calculated. The experiment stops after 1000 peers entered the system. The value for R in the Repeat and TIVPrev algorithm is equal to 20.

Latencies are extracted from the King Dataset. The King data-set contains is a 1740x1740 matrix with latencies measured between 1740 DNS servers[28]. The entry on row n and column m contains the latency measurement from DNS server n to m. An entry is calculated as follows. At first the latency of two DNS lookups toward the severs n and m are measured 5 times and averaged. The DNS lookup of the second server m is routed via the first. Therefore the difference between the latencies of the DNS lookups is equal to the latency from n to m. The latencies are measured multiple times to reduce potential inaccuracies due to variatons by transient network conditions. Gummadi and authors show that the latencies measured in the king dataset are indistinguishable from ICMP ping times.

The latencies from the King Dataset are between DNS servers and are not representative for the environment in which Tribler typically operates. Tribler instances are run on end-user devices such as mobile phones and personal computers with more volatile network conditions. Therefore the validity in real-world conditions is limited. However, latencies are still measured between real-world computers. The relation between the distance and latency (see figure 2.4) is still encapsulated in the experiment.

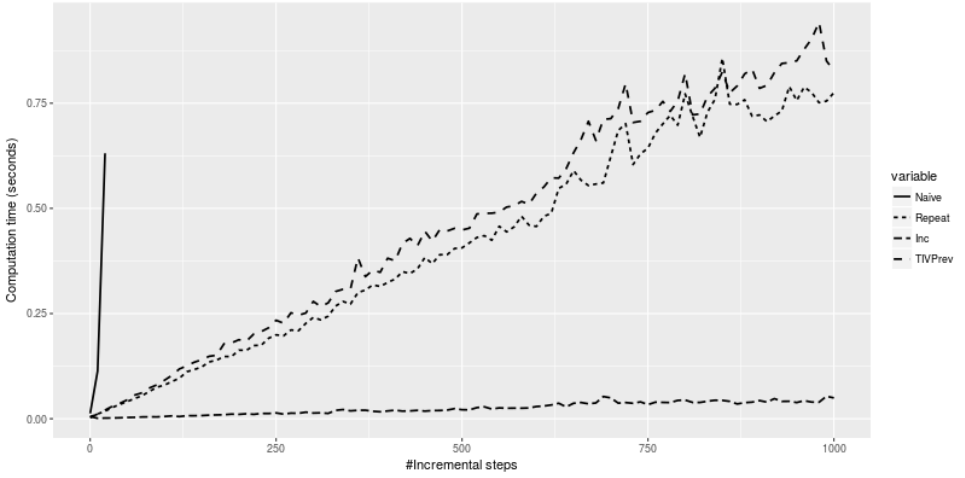

Figure 5.1: Graph of the computation time metric for different settings in the algorithm validation experiment. The x-axis are the number of peers that enter the system. The metric is the average of 10 x-axis values.

RESULTS

The computation time of the naive algorithm grows quadratic, while the computation time of the other algorithms grow linearly (figure 5.1). These behaviours can be ex-

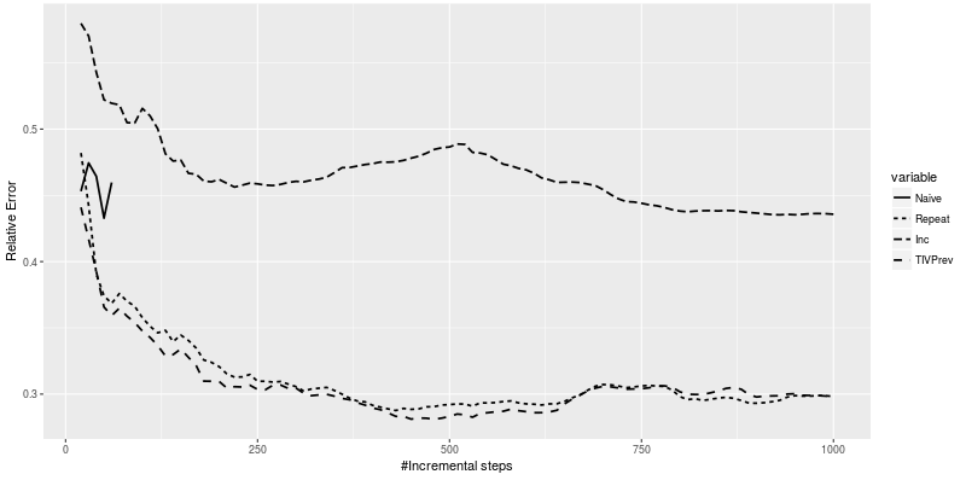

Figure 5.2: Graph of the relative error metric for different settings in the algorithm validation experiment. The x-axis are the number of peers that enter the system. The metric is the average of 10 x-axis values.

plained by the complexity of the algorithms. The naive algorithm grows parabolic because it has complexity $O(m * n^2)$ where n is the number of peers in the system and m is the number of error function calls. After 30 peers entered the run for the naive algorithm is stopped because the computation time is too high. The complexity of one step of the Inc algorithm is $O(m * |L|)$ where $|L|$ is the size of the number of latencies measured by one peer. Complete information is assumed and new peers know all latencies toward all other peers in the system. Therefore, $|L| = n$ and the complexity of the Inc algorithm becomes $O(m * n)$. The Repeat and TIVPrev algorithms have a higher growth rate compared to Inc due to R coordinate updates. The complexity is $O(m * R * n)$.

The computation times of the TIVPrev and Repeat algorithm eventually become larger than 0.5 seconds. This means the computation time becomes problematic in networks with large number of peers. For instance, in networks with millions of peers the Repeat and TIVPrev algorithms will use such a large computation time that the user experience is effected. Users will be limited in using other applications simultaneously. This seems not to be a problem with the Inc algorithm because the linear growth is smaller. After 1000 incremental steps the computation time of the Inc algorithm is still smaller than 0.1 seconds.

Repetitively updating past calculated coordinates is highly beneficial for higher accuracy and faster convergence. When looking at the ranking accuracy and relative error performance metrics the TIVPrev and Repeat algorithms have the best performance (figure 5.2 and 5.3). They converge in less than 30 steps and have after convergence a ranking accuracy between 50% and 60% and a relative error close to 0.30.

In incremental algorithms the information is added in phases and thus no perfect decisions can be made. With no compensation in the Inc run the performance is worst. The run has a ranking accuracy after convergence between 40% and 50% and a relative error around 0.35. With the Repeat and TIVPrev algorithms a compensation mechanism

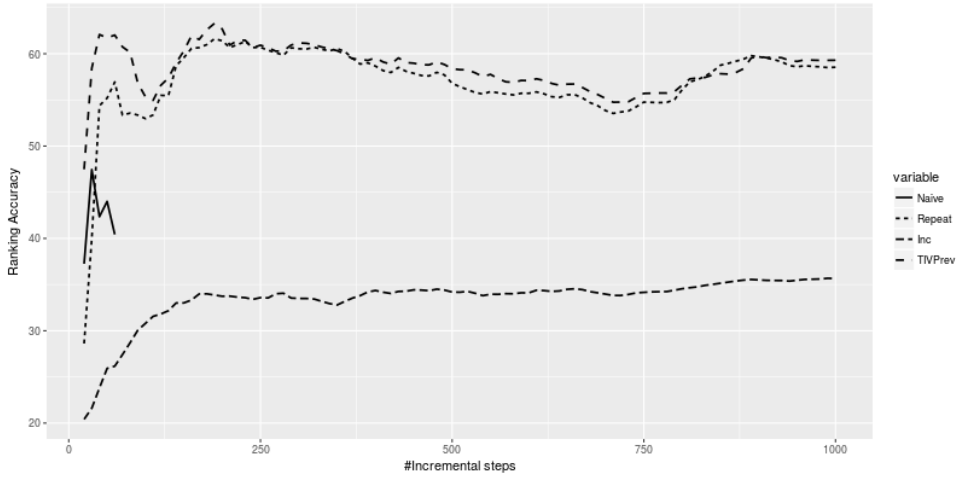

Figure 5.3: Graph of the ranking accuracy metric for different settings in the algorithm validation experiment. The x-axis are the number of peers that enter the system. The metric is the average of 10 x-axis values.

is implemented. Decisions from the past are reconsidered. With new latency information positions of past calculated coordinates are updated. This comes with the price of extra computation time when updating the coordinates.

Even though the naive algorithm uses a lot of computation time, it does not have the best accuracy. The accuracy is between the Inc- and the Repeat and TIVPrev algorithms. An explanation for the bad performance is that it could be hard to find a global minimum when minimizing the objective function. In the naive algorithm the coordinates of all corresponding hosts are in one objective function. When the coordinates of one host changes, the error of another host is influenced. Therefore, minimizing the function can be become rather complex. Therefore the minimization algorithm has higher probability to converge to a local minimum.

5.3. SYNTHETIC EXPERIMENT

The goal of the synthetic experiment is to validate the correct implementation of the low latency overlay in Tribler. It is executed with a synthetic dataset. In the first part of the experiment the accuracy is measured. In the second part is tested how well the overlay reacts to new entering peers to the network.

ENVIRONMENT PART 1

In the first part the low latency overlay is run with default settings on 500 Tribler instances, using 35 nodes. The nodes are managed by the ASCI Supercomputer 5 (DAS5) server cluster. Gumby is used as experiment framework to gather results [29]. Every 10 seconds the performance metrics "Relative Error" and "Ranking accuracy" are computed for every Tribler instance. The number of repetitions R done every second is set to 20 to enhance the speed of convergence. The *balance introductions* setting is set to 8. At the end of the experiment Gumby aggregates and averages the results at each time

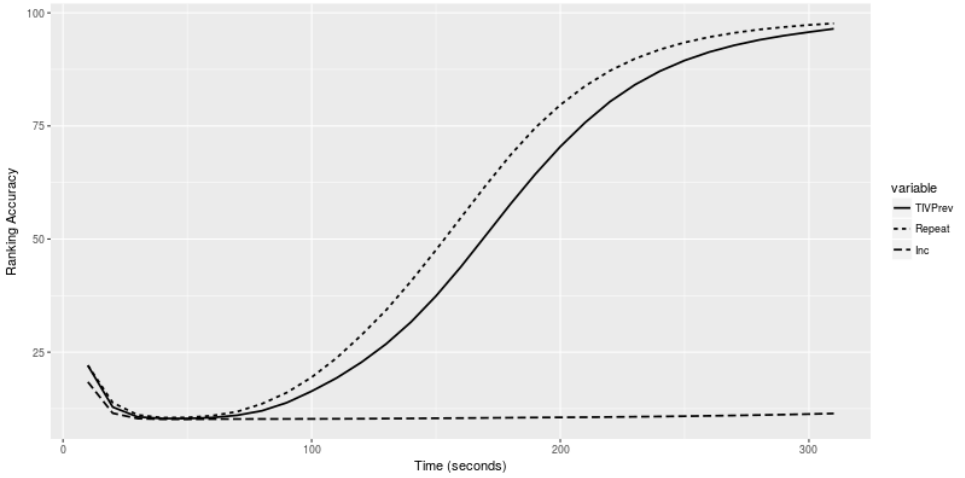

Figure 5.4: The figure shows the ranking accuracy metric calculated every 10 seconds in the synthetic experiment part 1. Each line represents the average ranking accuracy over every peer for a different algorithm at a certain point in time.

point. The experiment is run for 320 seconds to make the algorithms converge.

A synthetic dataset is constructed with latencies extracted from a two-dimensional map with 500 points. The coordinates of the points are randomly chosen between zero and 1. Latencies in the dataset are equal to the distances between the points in the map. Because the latencies are directly extracted from the map, it is expected that the latency estimation algorithms should give near-perfect predictions.

RESULTS PART 1

The experiment shows that the low latency overlay is correctly implemented. In the runs with the "TIVPrev" and "Repeat" latency estimation algorithms, all accuracy performance metrics converge to optimal values (figures 5.4, 5.5). With the "Ranking Accuracy" at almost 100% and the "Relative error" approximately zero the algorithms give near-perfect predictions after 300 seconds. The perfect predictions imply that all peers can be modeled as coordinates in a map where distances correctly estimate latencies. The TIVPrev and Repeat algorithms show that they can correctly reproduce the original peer locations of the data-set.

At the end of the experiment in the Inc run the predictions are not of high quality. The "Ranking Accuracy" is below 10% and the "Relative Error" is higher than 1 (figures 5.4 and 5.5). It appears to be highly beneficial to repetitively update past calculated coordinates to give a compensation to the inability of incremental algorithms to process all input information at once. When new latencies arrive past calculated coordinates can be updated that give better estimations. In the Inc run this does not happen.

ENVIRONMENT PART 2

In the second part the reaction of the low latency overlay to new entering peers is tested. The low latency overlay is run with the same settings and synthetic dataset as in the

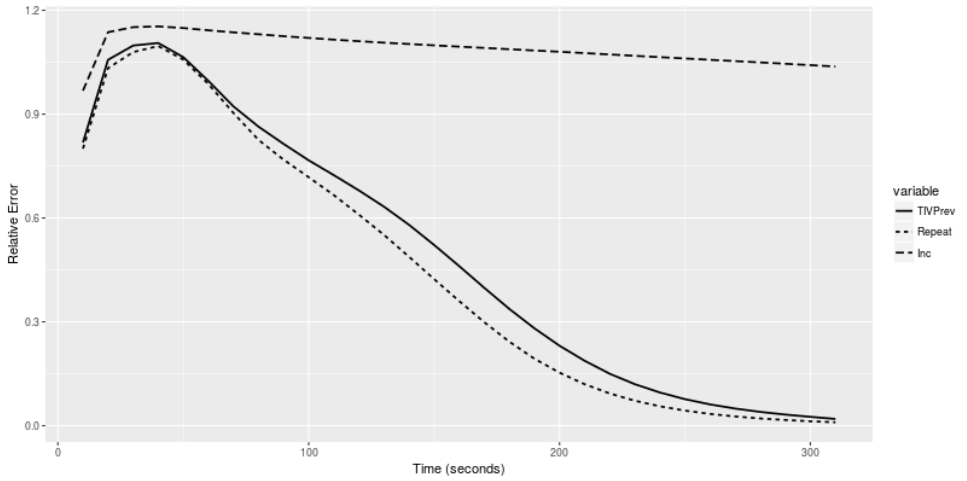

Figure 5.5: The figure shows the relative error metric calculated every 10 seconds in the synthetic experiment part 1. Each line represents the average relative error over every peer for a different algorithm at a certain point in time.

first. The experiment is run 600 seconds. Ten peers enter the network after 400 seconds. Before they enter, they do not participate. Every 10 seconds the performance metrics "Quality of introductions" and "Top 10 lowest latency neighbours" are computed for the ten entering peers.

RESULTS PART 2

With near 100% latency prediction quality in the "TIVPrev" and "Repeat" runs after 400 seconds, the neighbourhoods of the new entering peers are rapidly occupied with low latency peers. The "Quality of introductions" metric converges to a value around 5 after 50 seconds. With a value of 5 there is still some variation in the introductions. Multiple low latency peers are introduced to get a high quality neighbourhood. With a value of 1 the lowest latency peer would always be introduced and the new entering peers would not have a variety of low latency peers in their neighbourhood. After 80 seconds "Top 10 lowest latency neighbours" converges to almost perfect conditions (see figures 5.6 and 5.7). In the first 50 seconds the quality of introductions is higher because the latencies toward new entering peers are not yet well estimated. As more latencies are measured toward new entering peers the estimations become better.

As the new entering peers establish themselves in the neighbourhoods of nodes, the cluster effect adds to the quality of the overlay. The cluster effect is shown in figure 4.10. Clusters of nodes that have a low latency toward each other are more often introduced to other peers of the cluster. Therefore the number of low latency peers in the neighbourhood increases.

In the Inc run, the "Quality of introductions" converge below 50 and the "Top 10 lowest latency neighbours converges to a value of 9 (see figures 5.6 and 5.7). Despite the limited predictability of the Inc algorithm, the performance is high. This can only be

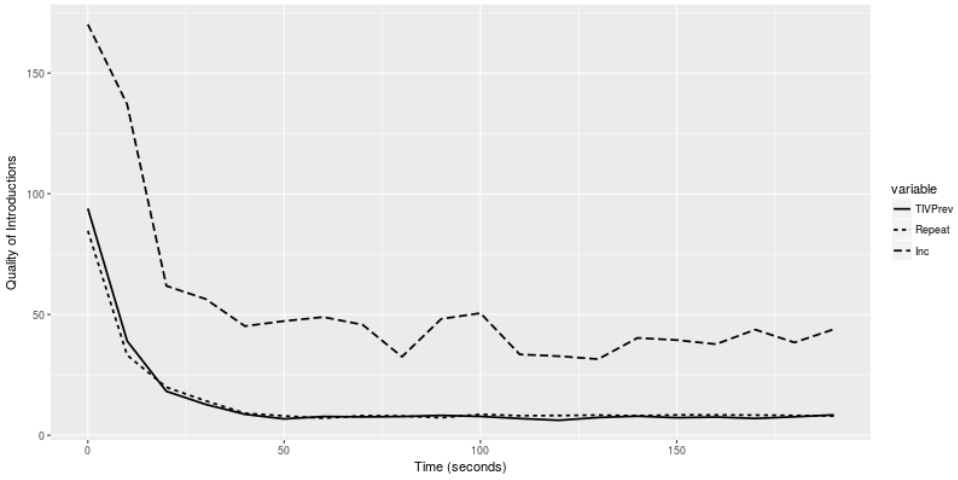

Figure 5.6: The figure shows the quality of introductions metric calculated every 10 seconds in the synthetic experiment part 2. Each line represents the average quality of introductions over every peer for each different algorithm at a certain point in time.

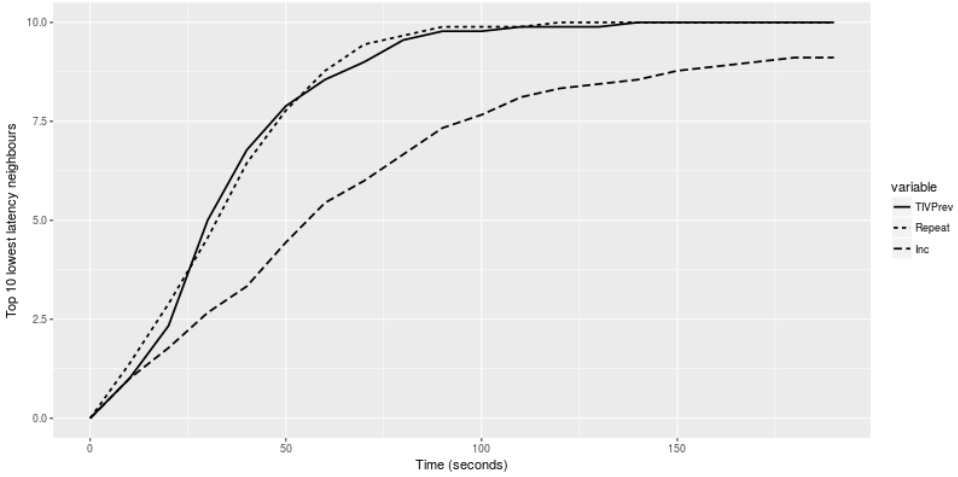

Figure 5.7: The figure shows the top 10 lowest latency neighbours metric calculated every 10 seconds in the synthetic experiment part 2. Each line represents the average top 10 lowest latency peers metric over every peer for each different algorithm at a certain point in time.

explained with the cluster effect. As more low latency peers gets occupied in the neighbourhood the quality of introductions increases.

5.4. ACCURACY EXPERIMENT

The goal of the experiment is to measure the performance of the low latency overlay. Multiple Tribler instances are run with different settings.

ENVIRONMENT

The experiment runs the low latency overlay with various settings on 500 Tribler instances, using 35 nodes. Gumby is used as experiment framework to gather results. The nodes are managed by the ASCI Supercomputer 5 (DAS5) server cluster. The experiment is run 2200 seconds. Every 10 seconds the performance metrics "Relative Error", "Ranking accuracy", "Quality of introductions" and "Top 10 lowest latency neighbours" are computed for every Tribler instance. At the end of the experiment Gumby aggregates and averages the results at each time point.

Latencies are extracted from the King Dataset in the same way as in the algorithm validation experiment. To integrate the King Dataset in Tribler each node in the P2P network is given a unique ID. The ID is an integer from 1 to 500. Whenever a node wants to lookup the latency measurement between two nodes with ID a and b it uses the King Dataset. The node retrieves the latency measurement between DNS Server a and b. This latency is in row a and column b of the King Dataset.

Different settings are tested in runs. The three different algorithms "TIVPrev", "Repeat" and "Inc" are run with default settings. Next to that is the "TIVPrev" algorithm run three additional times with other settings in the "TIV20Repeat", "TIVSlowWalk" and "TIVOldMethod" runs. In the "TIV20Repeat" run, previously calculated coordinates are updated 20 times a second. The default setting updates just once a second. The "TIVSlowWalk" the interval time between two walker steps is increased to 5 seconds. With the "TIVOldMethod" setting the overlay uses 50% of times the normal dispersy node selection method when taking a step.

RESULTS

Using the measured latencies of the king-dataset does not result in perfect predictions. The results of the accuracy of the latency estimation algorithms can be seen in figures 5.9 and 5.10. After convergence the relative error converges to a value around 0.35 and the ranking accuracy to 45% for the "TIVPrev", "TIVOldMethod" and "TIVSlowWalk" and "Repeat" runs. There are other factors that explain the latency differences between computers than their physical distance. Routing policies, Triangle inequality violations and the used transmission medium also effect the latency. Therefore, the locations of the DNS-servers of the king data-set cannot be perfectly modeled to corresponding coordinates in a two-dimensional map where the distances perfectly predict latencies. The location model gives limited predictions.

Compared to the algorithm validation experiment on a single machine the Tribler implementation has a similarly but slightly less accurate latency estimator. The best run in the Tribler implementation has a difference of about 5% to 10% in ranking accuracy and 0.05 in relative error compared to the experiment on a single machine (see figures

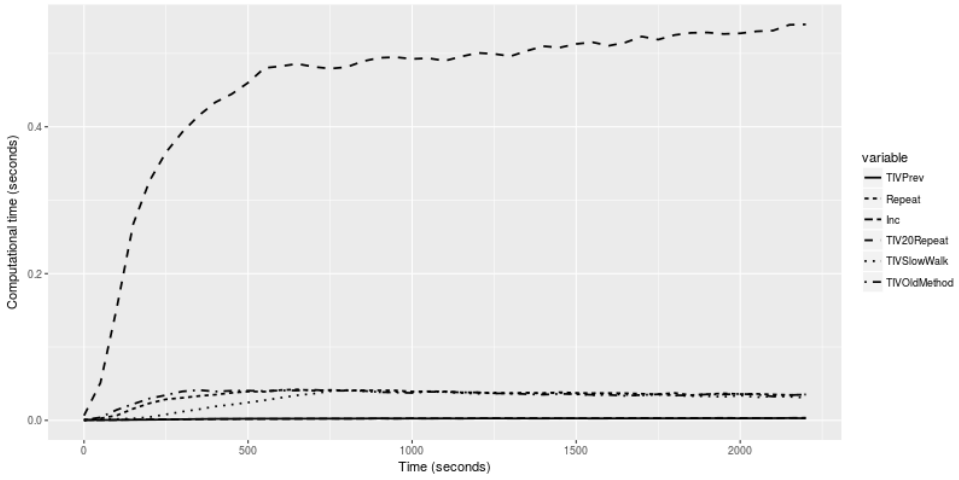

Figure 5.8: The figure shows the computation time metric calculated every 10 seconds in the accuracy experiment. The x-axis is the elapsed time of the experiment. Each line represent the average computation time over every peer for each different run at a certain point in time.

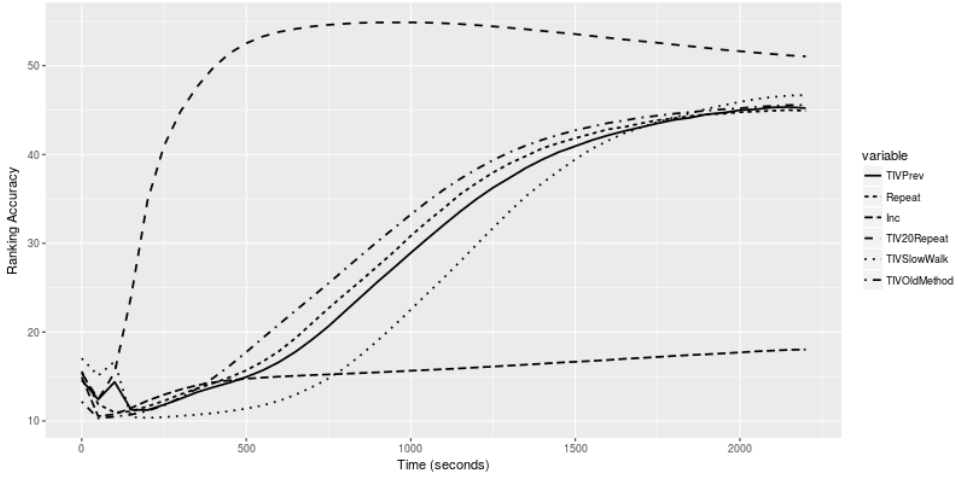

Figure 5.9: The figure shows the ranking accuracy metric calculated every 10 seconds in the accuracy experiment. Each line represents the average ranking accuracy over every peer for each different algorithm at a certain point in time.

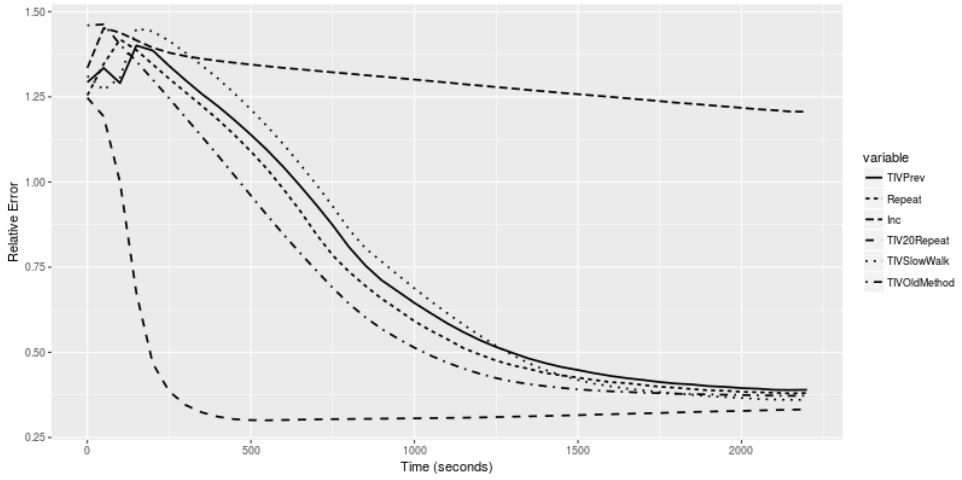

Figure 5.10: The figure shows the relative error metric calculated every 10 seconds in the accuracy experiment. Each line represents the average relative error over every peer for each different algorithm at a certain point in time.

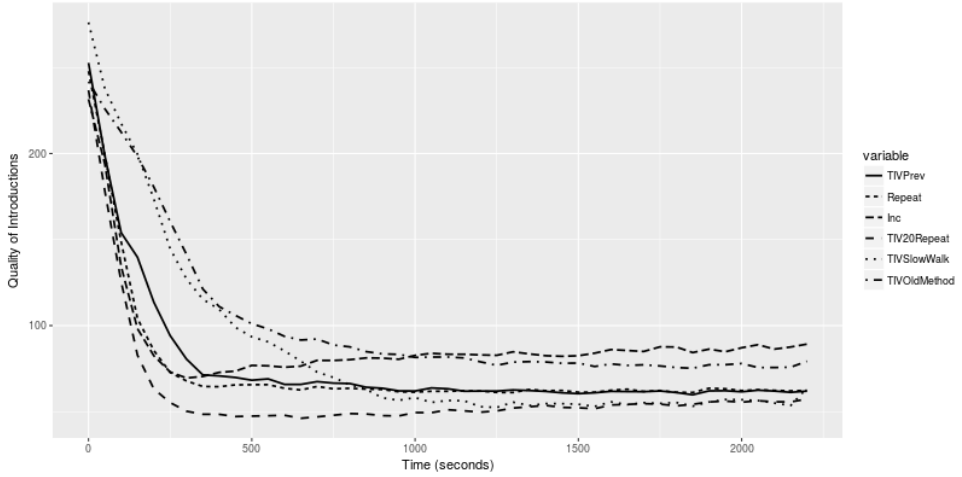

Figure 5.11: The figure shows the quality of introductions metric calculated every 10 seconds in the accuracy experiment. Each line represents the average relative error over every peer for each different algorithm at a certain point in time.

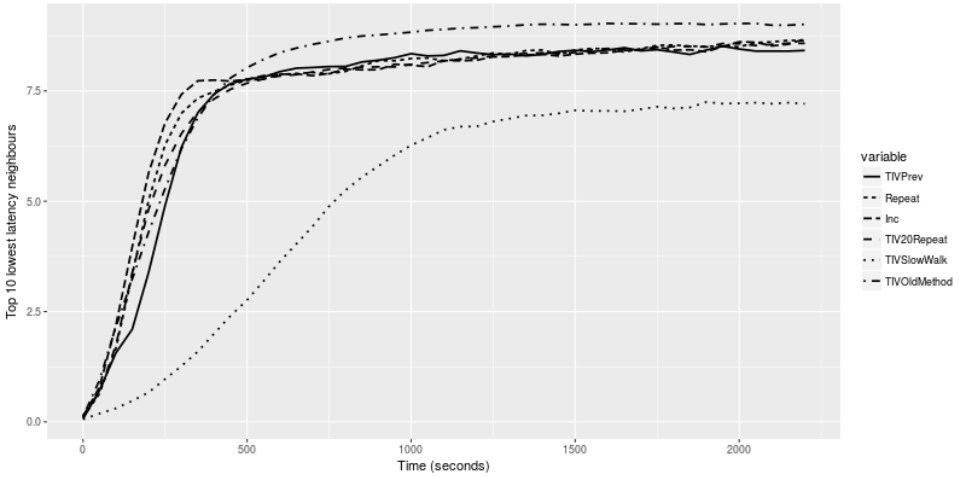

Figure 5.12: The figure shows the top 10 lowest latency peers in neighbourhood metric calculated every 10 seconds in the accuracy experiment. Each line represents the average relative error over every peer for each different algorithm at a certain point in time.

5

5.9, 5.10, 5.3 and 5.2). The only explanation are the differences in available latency information in both experiments. In the single machine experiment the knowledge of all latencies between peers is assumed while in the Tribler implementation they first have to be obtained. At the end of the accuracy experiment runs not all latencies between peers have been obtained.

The accuracy performance cannot be translated one-to-one toward real-world settings. The latencies from the King Dataset are between DNS servers instead of end-user devices in which Tribler operates. In the real world the latencies are probably higher and more volatile due to network conditions. This could have an implication on the accuracy of the estimations. It might be harder for the algorithms to find coordinates that provide the same results for the relative error and ranking accuracy.

The frequency of updating past calculated coordinates has a large effect on the speed of convergence. In the TIV20Repeat run, twenty coordinates are updated per second. Therefore the quality of predictions of the latency estimation algorithm converges earlier compared to other runs (see figures 5.9, 5.10). In each update new arrived latency information is taken into account. More frequent updating allow the latency estimator to captivate new information on peers and latencies with higher speed. It is beneficial to do this since with incremental algorithms temporarily solutions are created with incomplete input. When new information arrives the old solution should be reconsidered and updated. In the Inc run there are no updates and therefore the quality of predictions is much lower. There is a large prediction quality difference between the Inc run and runs that feature updating of past coordinates. After 1600 seconds of elapsed time in the experiment, the ranking accuracy difference is almost 30% and the relative error differs 0.5 (see figures 5.9, 5.10). The accuracy benefit of more frequent updating comes at the cost of computation time (see figure 5.8). With 20 updates per second computation time

rises to 0.5 seconds. With only one update per second the computation time is less than 0.05 seconds.

The performance of the latency estimation algorithms has an effect on the quality of introductions but less than expected (see figure 5.11). When the estimators quality is high with runs that repetitively update coordinates the quality of introductions converge to an average about 50. This value means that on average the fiftieth lowest latency is introduced. In the Inc run, it converges to an average around 75. The explanation for this is simple. When the overlay knows which peers have a low latency toward each other it is capable of introducing them. With no knowledge, the introduction quality decreases. However, despite bad predictability of the Inc algorithm the introduction quality is still quite high. This can only be explained with the cluster effect. When more of the lowest latency neighbours are discovered, the introduction quality automatically increases.

It is interesting to note that despite the high performance in latency prediction, the TIVSlowWalk run has the worst performance in neighbourhood quality (see figure 5.12). At the end of the experiment 70% of the 10 lowest latency peers are in the neighbourhood. In the TIVSlowWalk run the interval between steps in the peer discovery mechanism is set to 5 seconds instead of one. With slower walking, the overlay is not able to keep all lowest latency peers in the neighbourhood. The explanation for this is that peers leave the neighbourhood because the firewall closes the connection. This happens after a certain period, usually one minute, of inactivity. With a 5 seconds interval only $60/5 = 12$ steps are made per minute.

When walking less steps a minute in the TIVSlowWalk run, the quality of the latency estimation algorithms converges to the same point as with the TIV run, but slower (see figure 5.10 and 5.9). The reason is that it takes more time to obtain latency information. Latency information is important for letting the latency information algorithms perform well. With the TIVSlowWalk run, the quality of introductions is lower (see figure 5.11). This can be explained with the cluster effect. The cluster effect adds to the quality of the overlay if there are low latency peers in the neighbourhood. However, there are less of the lowest latency peers in the neighbourhood compared to other runs. Therefore the cluster effect is less effective.

When using the original Dispersy method for node selection 50% of times in the TIVOldMethod run, the quality of the neighbourhood converges to better results despite lower introduction and equal latency estimation quality (5.12, 5.11, 5.10 and 5.9). The original Dispersy method adds more randomness to the overlay to dampen the eclipse attack. The explanation for the better neighbourhood quality is that with randomization the overlay gets a higher variety of neighbours. In the new node selection mechanism the overlay walks toward the top 10 lowest latency peers in order to keep them in the neighbourhood. With more randomization the chances are higher that the introducee is one of the lowest latency peer. Because in 50% of times the new node selection method is used, the overlay is also able to keep the low latency peers in the neighbourhood.

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. When Tribler operates in the real world new peers enter the network over time. The nodes that were in the network already

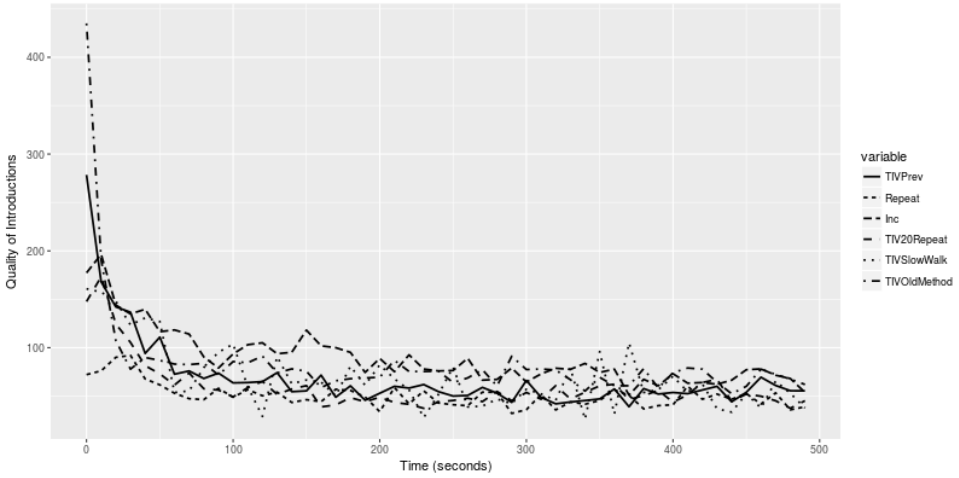

Figure 5.13: The figure shows the quality of introductions metric of the 10 new entering peers in the bootstrap experiment. Each line represent the average quality of introductions metric.

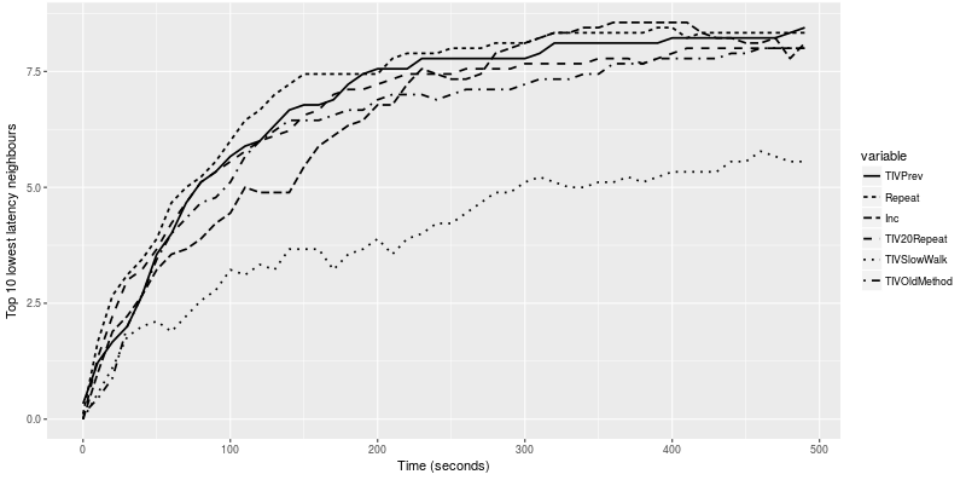

Figure 5.14: The figure shows the top 10 lowest latency peers metric of the 10 new entering peers in the bootstrap experiment. Each line represent the average of the top 10 lowest latency peers metric.

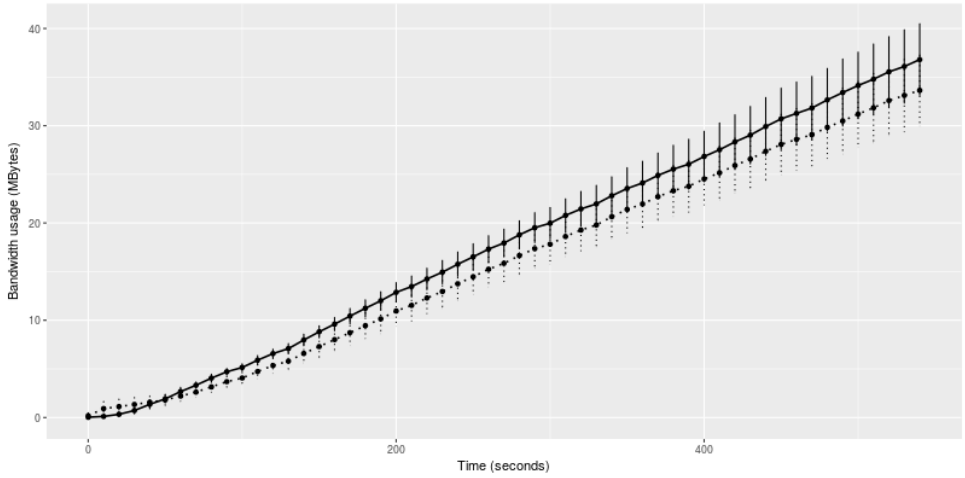

Figure 5.15: The figure shows the total upload and download for the 10 new entering peers in the bootstrap experiment. The solid line represent the upload and the dotted line the download speed. The bar represents the size of the variance of the total upload or download relative to the mean.

found their lowest latency peers. How fast the new entering peers find their lowest latency neighbours is tested in this experiment. Multiple Tribler instances are run in a test environment. At some point in the experiment new peers enter and their behaviour is measured.

ENVIRONMENT

The experiment runs the low latency overlay with various settings on 500 Tribler instances, using 35 nodes. Gumby is used as experiment framework to gather results. The nodes are managed by the ASCI Supercomputer 5 (DAS5) server cluster. The experiment is run 2200 seconds. Ten peers enter the P2P network after 1600 seconds. In the first 1600 seconds they do not participate. There performance is measured every 10 seconds with the metrics "Relative Error", "Ranking accuracy", "Quality of introductions" and "Top 10 lowest latency neighbours".

Latencies are extracted from the King Dataset and the same settings are used as in the accuracy experiment. The three different algorithms "TIV", "Repeat" and "Inc" are run with default settings. Next to that is the "TIV" algorithm run three additional times with other settings in the "TIV20Repeat", "TIVSlowWalk" and "TIVOldMethod" runs. In the "TIV20Repeat" run, previously calculated coordinates are updated 20 times a second. The default setting updates just once a second. The "TIVSlowWalk" the interval time between two walker steps is increased to 5 seconds. With the "TIVOldMethod" setting the overlay uses 50% of times the normal dispersy node selection method when taking a step.

RESULTS

The quality of introductions of new entering peers are about the same as with ordinary hosts after convergence. (see figure 5.13). In the first 50 seconds the new entering peers

have to be positioned by the algorithms. Therefore the quality of introductions is worse in the beginning of the experiment. The length of this period is about the same for all runs. The Inc algorithm performs only slightly worse than the other algorithms. Since the latency estimation is of low quality, the only explanation for such good performance is the clustering effect. In the Inc run the neighbourhood still gets occupied with low latency peers (see figure 5.14). This increases introduction quality.

Despite the good results for the introduction quality it takes about 250 seconds before the new entering peers are converged toward a neighbourhood with the lowest latency nodes (see figure 5.14). The time to convergence is still about two times faster than with the accuracy experiment but worse than expected. Latency estimation- and introduction quality are not sufficient for faster convergence. The algorithms are limited in predicting the latencies of the king-dataset since in the synthetic experiment predictions were near perfect.

The bandwidth usage of the new entering peers is as low as expected (see figure 5.15). Every run has the same byte cost. The upload and download speed are throughout the experiment about the same and show a linear growth. However, there are slightly more bytes download than uploaded. This is normal because no latencies are measured yet and thus less are send in the ping message when uploading. After 500 seconds the total byte usage is around 72Mbyte, 38Mbyte download and 34Mbyte upload. The average upload and download speed throughout the experiment is around 65 KByte per second. 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 and every second a new step is taken with the peer discovery mechanism. Therefore are in the 500 seconds of the experiment 250 ping and pong messages send to the entire neighbourhood of each peer. At the same time are also 500 steps taken by the peer discovery mechanism.

5.6. SYBIL EXPERIMENT

The goal of the experiment is to test the capability of the overlay to prevent the sybil attack. The success of a sybil attack is determined by the average amount of sybils in the neighbourhoods of peers. It makes it harder for an adversary to do a sybil attack when peers have low latency nodes in their neighbourhoods. An adversary has to control sybils with a low latency toward their target to successfully launch the attack. It is expected that with randomly chosen sybils in the network the attack will be less successful.

ENVIRONMENT

The experiment runs the low latency overlay on 500 Tribler instances, using 35 nodes. Fifty of the 500 Tribler instances are chosen randomly and marked as sybils. Every 10 seconds is determined how many sybils are on average in the neighbourhood of every peer. Gumby is used as experiment framework to gather results. The nodes are managed by the ASCI Supercomputer 5 (DAS5) server cluster. The experiment is run 800 seconds. Latencies are extracted from the King Dataset in the same way as in the accuracy experiment.

There are two runs in the experiment: "TIV20Repeat" and "Normal". In the "TIV20Repeat" run the low latency overlay is implemented. The TIV algorithm is used where previously

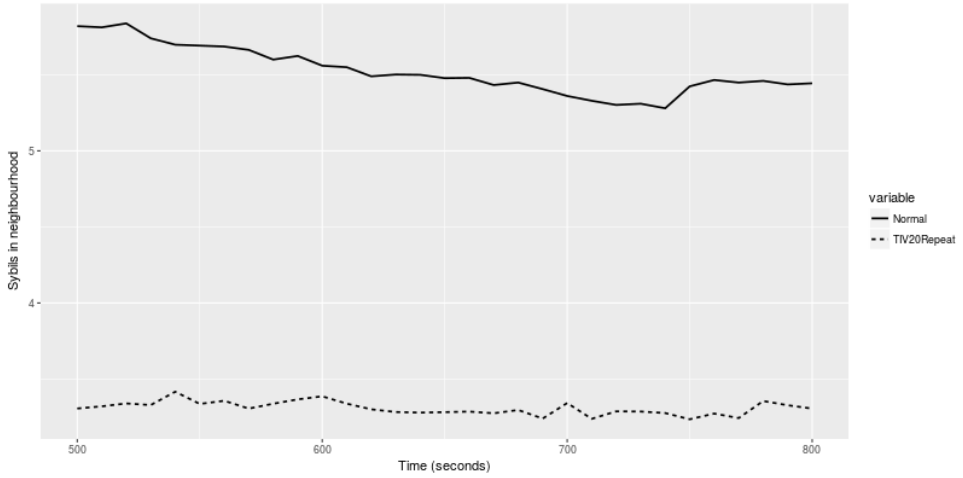

Figure 5.16: The figure shows the average number of sybils in the neighbourhood of peers in the sybil experiment. Each line represents one run from the experiment. On the x-axis is the number of seconds elapsed in the experiment.

calculated coordinates are updated 20 times a second. These settings had the best results in the bootstrap and accuracy experiments. In the "Normal" run the currently used peer discovery mechanism of dispersy is used without the low latency overlay. From the accuracy experiment can be concluded that the "TIV20Repeat" run is converged after 500 seconds. Therefore are the results collected after 500 seconds in both runs.

RESULTS

The results from figure 5.16 show that the low latency overlay makes the sybil attack less successful. With the TIV20Repeat run there are on average 3.2 sybils in the neighbourhood and with the normal run 5.5. The amount of sybils is reduced with the low latency overlay. The explanation is that the randomly chosen sybils are less likely to be the low latency peers in the neighbourhood. To successfully launch a sybil attack in the overlay an adversary needs to control the low latency peers of a node.

5.7. DISCUSSION

Nodes in the low latency overlay eventually find their lowest latency peers. However, due to inadequate latency predictions this can take a large amount of time and the lowest latency peers are not always found. The latency estimation algorithms are not able to do good predictions with real world measured latencies resulting in a low quality of introduction. It appears that the geolocation model for latency prediction between peers is not good enough to provide a high quality low latency overlay.

The performance of the runs that include the TIV algorithms are the best. To gain such performance it is vital to repetitively update past calculated coordinates. With default settings the updating is still done in a reasonable amount of computation time. When more coordinates are updated per second in the TIV20Repeat run, the accuracy

converges faster. However, it costs extra computation time to have more updates per second.

It is important to have small time intervals between steps in the peer discovery mechanism. When doing that the lowest latency peers are kept in the neighbourhood. They are then reconnected to the neighbourhood after NAT-timeouts. In the TIVSlowWalk the time interval is high and low latency peers leave the neighbourhood. Adding randomness with the normal dispersy walking method in the TIVOldMethod not only dampens the eclipse attack but also leads to faster convergence.

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 100 000 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 are limited because of the peer introduction mechanism. 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] Ciamac C Moallemi and Mehmet Sağlam. Or forum—the cost of latency in high-frequency trading. *Operations Research*, 61(5):1070–1086, 2013.

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

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

[4] 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.

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

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

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] About tribler. https://tribler.org/about.html, 2018. Accessed: 2018-04-25.

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

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

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

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

[17] 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.

[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] 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.

[20] 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.

[21] 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.

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

[23] 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.

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

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

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

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

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

[29] Tribler. Gumby test framework downloaded from https://github.com/tribler/gumby, 2018.