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

Albert Einstein

CONTENTS

1

INTRODUCTION

A decentralized market has been implemented in Tribler by Olsthoorn (2016) that does not guarantee the privacy of traders. Traders can exchange BitCoin against Multichain coin in a decentralized system. Ensuring the privacy of traders in an exchange is important because otherwise traders can play games and abuse the trade information of other parties for their own benefit. Sensitive trading information becomes public to other users and the trading position of a trader can potentially be derived at two points. At first, there is a decentralized matching engine where bid and ask offers are broadcasted to all other traders to make a match. [**?**] Secondly, the trading position of a trader might be exposed because the BitCoin wallet does not ensure privacy. The payment transactions are recorded in a decentralized public ledger from which much information can be deduced. An alternative to the BitCoin wallet is the Zerocash wallet which uses a changed version of the blockchain that ensures the privacy of transactions with zero knoweldge proofs and onion routing. [**?**] However, this is not an option because users should be allowed to pay with the BitCoin wallet and with other wallets from for instance traditional banks like ABN AMRO or ING.

2

Problem Description

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

2.1. Latency in trading

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

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

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

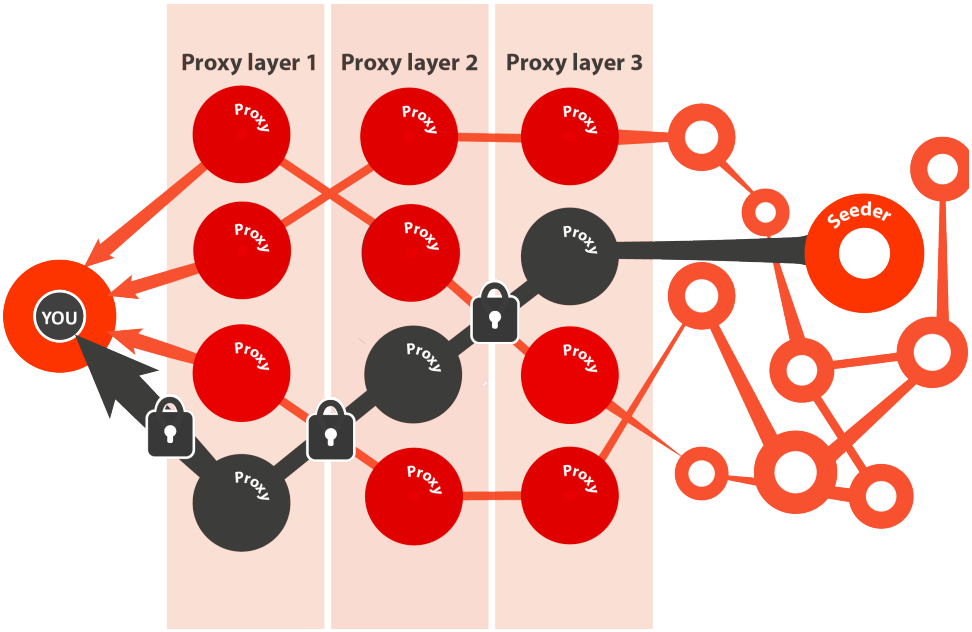

Figure 2.1: Overview of anonymization techniques in Tribler

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

Moallemi and Saglam (2013) estimate the latency cost based on cross-sectional data on volatilities and bid-offer spreads in the U.S. between 1995-2005 from the dataset of Ait Sahalia and Yu (2009). The results can be seen in 2.2. The median latency cost approximately increased threefold in the 1995-2005 time period. To obtain the latency cost estimation the data set is used in a model that under simplifications calculates the latency cost. The model assumes an individual trader with a fixed latency of 500ms. As time increases, the cost for this latency also increases. As can be seen later on, the Tribler market has latencies around 150 ms. The assumption of a trader with 500ms is realistic in the Tribler context. For details of the model we refer to the paper of Moallemi and Saglam (2013). [**?**]

2.2. LATENCY IN ANONYMIZATION TECHNIQUES

Anonymization techniques require data to go through different nodes to make it hard to link the sender and receiver of a message. In one of the early anonymization tech-

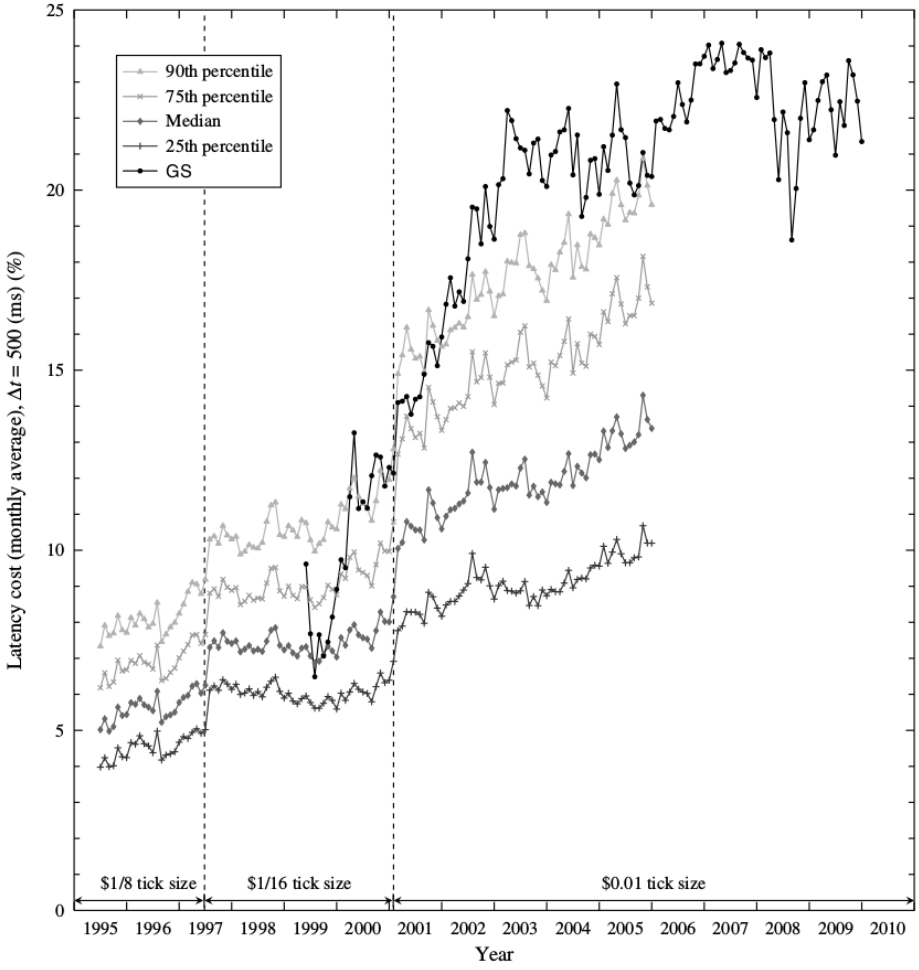

Figure 2.2: A hypothetical investor with a fixed latency of 500 ms is assumed. The latency costs are computed from the data set of Ait Sahalia and Yu (2009) The latency cost for GS is also reported, beginning from its IPO. The dashed lines correspond to dates where the NYSE tick size was reduced. The latency cost had a consistent increasing trend over the 1995-2005 period. The median latency cost approximately increased threefold by reaching roughly 14% from 5%.

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

2.3. LATENCY IN PARALLEL ALGORITHMS

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

2.4. THE CURRENT STATUS OF LATENCY IN TRIBLER

The latency of Tribler applications appears to be around 150ms normally. There are however outliers of latencies of 10 seconds. The normal latency response of 150ms is high for a exchange market but explained by the distributed nature of the Tribler market. Other exchange markets that are considered low latency have latencies around 10 ms. The outlier latencies of 10 seconds are unacceptable in the market application. These super high latencies result almost directly in the problems described by Cespa and Foucault, 2009. 1) Competitive advantage for other traders 2) Bad decision making from traders due to incomplete information and 3) Low priority serving because another trader gets served earlier due to the first come first served principle. [**?**]

3

LATENCY ESTIMATION SYSTEMS

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

3.1. GNP ALGORITHM

One of the first algorithms proposed is the GNP system by Zhang et al. It assumes that hosts H are coordinates in a D dimensional geometric space S. Because S is geometric the distance function between two host coordinates $C^S_{H_1}$ and $C^S_{H_2}$ is easily calculated by taking the euclidean distance. The distance function $f(C^S_{H_1}, C^S_{H_2})$ denotes the computed distance between two hosts $C^S_{H_1}$ and $C^S_{H_2}$. The distance function $f(C_{H_1}, C_{H_2})$ states the measured latency between the two hosts H_1 and H_2. In the first step a subset of landmarks L from all the hosts H are chosen as landmarks for points of reference. Landmarks enable fast host position calculation in step 2 of the algorithm. Suppose there are N landmarks chosen and each of the landmarks measure the latencies between hosts resulting in an NxN distance matrix. In order to uniquely compute host coordinates at least $D+1$ landmarks are chosen and thus $N > D+1$. The goal is to find a set of coordinates $C^S_{L_1}, ..., C^S_{L_N}$ for the N landmarks such that the overall error between the measured latencies and the computed distances in S is minimized. Thus, in the first stage the following objective function is minimized:

$$f_{obj}(C^S_{L_1}, ..., C^S_{L_N}) = \sum_{L_i, L_j \in \{L_1, ..., L_N\} | i > j} = \epsilon(f(C^S_{L_1}, C^S_{L_2}), f(C_{L_1}, C_{L_2}))$$

where $\epsilon(.)$ is the error measurement function: $\epsilon(f(C^S_{L_1}, C^S_{L_2}), f(C_{L_1}, C_{L_2})) = f(C^S_{L_1}, C^S_{L_2}) - f(C_{L_1}, C_{L_2})$

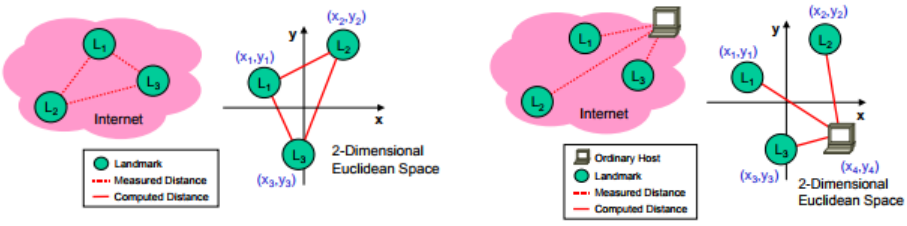

Figure 3.1: Part 1 and 2 of GNP algorithm: Landmark estimation and Host estimation

After the landmark coordinates $C^S_{L_1}, C^S_{L_N}$ are computed the second stage of the algorithm can start where other hosts place themselves relative to the landmarks.

In the second step a host first determines all the latencies to the landmarks. The landmarks have in this phase static positions that are already determined. A host H can compute its own coordinates C^S_H by minimizing the overall error between the measured and the computed host to Landmark distances. Thus for each host C^S_H we seek to minimize the following objective function:

$$f_{obj}(C^S_H) = \sum_{L_i \in L_1,\ldots,L_N} \epsilon(f(C^S_{L_i}, C^S_H)$$

The computation cost for the first step of the algorithm is $O(N^2 D)$ for each call of the objective function. There is a squared relationship with the number of landmarks in the first step. This makes the algorithm expensive for large number of landmarks N. The coordinate computation of hosts H is $O(N * D)$ per objective function call for H hosts. With host coordination there is only a linear relationship between the number of landmarks and computation time. It is likely that with more landmarks the algorithm becomes more accurate but takes more time to compute. The trade-off between number of landmarks and accuracy has to be made. In the example used by Zhang et al (2002) a normal number of landmarks chosen would be around 20 or 30. This makes the first step of the algorithm very fast. The second step has a linear relationship with the number of hosts H. Because with small N the computation time of the first step can be marginalized, the computation time will be almost linearly dependent on the number of hosts H.

3.1.1. MINIMIZING THE OBJECTIVE FUNCTION

3.2. VIVALDI ALGORITHM

Vivaldi is a variant on the coordinate-based systems. There is an error function defined:

$$E = \sum_i \sum_j (L_{ij} - ||x_i - x_j||)^2$$

where $||x_i - x_j||$ is the distance between the coordinates of nodes i and j. L_{ij} is the actual measured latency between i and j.. Vivaldi is similar to GNP in that it also tries to minimize such an error function. Vivaldi does this by conceptually placing a spring between each pair of nodes (i, j) with a rest length equal to the measured latency

between these nodes L_{ij}.

Every pair of nodes i and j exert a force F_{ij} on both nodes. where $(L_{ij} - ||x_i - x_j||)$ is the displacement of the spring from rest and the unit vector $u(x_i - x_j)$ gives the direction of the force on i. The direction for j is the opposite. The net force on i, F_i is the sum of all forces from other nodes.

$F_{ij} = (L_{ij} - ||x_i - x_j||) X u(x_i - x_j)$

$F_i = \sum_{i \neq j} F_{ij}$

The springs now move each node to the direction of the force in each step of the algorithm. At each step the forces are recomputed based on the new position of the nodes. This goes with the following formula:

$x_i = x_i + F_1 x t$

t is the length of the time interval that is calculated in each epoch of the algorithm. The choice of t is an important design criteria and determines how far a node moves after each epoch of the algorithm.

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

Each time the node communicates with another node, it moves it self in the direction of that node's spring for a short amount of time δ, reducing only the error towards that particular node. Nodes continually communicate with other nodes such that the positions eventually converge to a low error.

When a node i receives the coordinates of another node j and measures the latency l_{ij} between i and j the coordinates of i are updated in the following way:

$x_i = x_i + \delta X (l_{ij} - ||x_i - x_j||) X u(x_i - x_j)$

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

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

In order to obtain fast convergence and avoidance of oscillation Vivaldi varies δ depending on how certain the node is about its coordinates. Large δ values will help the node quickly go to a position with low error, while small δ values allows it to refine itself. The change in δ setting in Vivaldi also takes into account the error of the other node j, the node i is contacting. When the error of node j is high, the node should not get a lot of weight and thus δ should be lower. Vivaldi uses the following formula for changing the δ:

$\delta = c_c X \frac{local_error}{local_error + remote_error}$

With this approach, there is quick convergence, low oscillation and nodes with high error have a lower weight.

3.3. NPS SYSTEM

The GNP algorithm calculates node positioning with a centralized component. If an ordinary host wants to calculate its position, it has to probe all landmarks. This makes the landmark nodes and their network access links a bottleneck to the system. If one landmark or the connection towards a landmark fails, the system can hardly recover. The NPS algorithm improves the GNP algorithm by decentralizing it. NPS is build around the concept of the GNP algorithm defined in the paper as the "Basic network positioning method".

An important requirement in the decentralized NPS system is that all nodes have the same bases for the coordinate system and have consistent positions. If this is not the case, the coordinate systems of two arbitrary chosen hosts cannot be compared.

The landmark solution of the GNP algorithm immediately solves the consistency problem because all landmarks together form a base coordinate system. This base coordinate system is calculated in the first step of the GNP algorithm. The decentralized solution has to maintain consistency.

By imposing a position dependency hierarchy it is ensured that the positions of all hosts have the same bases defined by the Landmarks. Landmarks still define the bases and can serve as reference points for hosts in the system. In the NPS system, hosts can also serve as reference points to other hosts to define its base. This makes landmarks much less critical and landmarks become less of a bottleneck to the system. A membership server is introduced in the system to assign hosts to other hosts or landmarks and thus to create the hierarchical dependencies. We say host A depends on host B if A uses B as one of its reference points. The layer number is the maximum number of dependency hops of a host separating it from a landmark.

The membership server adds a centralized component to the system making the system not fully decentralized. Such a centralized component adds availability issues to the system: When the membership server fails, an ordinary host is unable to know who to contact for a coordinate-base reference.

In the first step of the GNP algorithm the following objective function is minimized:
$f_{obj}(C^S_{L_1},...,C^S_{L_N}) = \sum_{L_i,L_j \in \{L_1,...,L_N\} | i>j} = \epsilon(f(C^S_{L_1},C^S_{L_2}),f(C_{L_1},C_{L_2}))$

where $\epsilon(.)$ is the error measurement function: $\epsilon(f(C^S_{L_1},C^S_{L_2}),f(C_{L_1},C_{L_2})) = f(C^S_{L_1},C^S_{L_2}) - f(C_{L_1},C_{L_2})$

NPS implements a decentralized calculation of landmark calculation. $2xf_{obj}(.)$ can be expanded as:

$$
\begin{aligned}
2xf_{obj}(C^S_{L_1},...,C^S_{L_N}) = & \sum_{i\neq 1} \epsilon(f(C^S_{L_1},C^S_{L_i}),f(C_{L_1},C_{L_i}))+ \\
& \sum_{i\neq 2} \epsilon(f(C^S_{L_2},C^S_{L_i}),f(C_{L_2},C_{L_i}))+ \\
& ... \\
& \sum_{i\neq 3} \epsilon(f(C^S_{L_3},C^S_{L_i}),f(C_{L_3},C_{L_i}))
\end{aligned}
$$

The decentralized NPS algorithm works in such a way that each landmark chooses one term from the above equation. It computes the position of itself that minimizes that

term. Similar to an ordinary host computation in GNP. The newly calculated position is shared with other nodes and after 1 second of waiting the term is minimized again. The steps repeat until convergence is met which is achieved if after 3 consecutive iterations a landmark position has not moved by more than one millisecond in the euclidean space. The approach can embed 20 landmarks starting from their origin positions in approximately one minute and the resulting positions are just as accurate as the centralized approach.

3.4. PIC: PRACTICAL INTERNET COORDINATES FOR DISTANCE ESTIMATION

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

PIC works similar to GNP: each node is mapped in a d-dimensional euclidean space. Each new entering node n to the system determines the latency to a set of landmarks L, where L must have at least $d+1$ members. The entering node also obtains the coordinates of each landmark. The new node then computes its coordinate by minimizing the error between the measured distances and computed distances between the new entering node and the landmarks. The authors of the paper experimented with several target error functions to minimize, the one that performed the best was the sum of the squares of the relative errors:

$$\sum_{i=1}^{|L|}\left(\frac{d_i^m - d_i^p}{d_i^m}\right)^2$$

where d_i^m is the distance measured between the new entering node and the ith node in L. d_i^p is the distance predicted between the new entering node and the ith node in L.

In the PIC algorithm three different strategies have been tested to choose a subset of landmarks L out of all nodes N whose coordinates have already been computed. The strategies are as follows:

1) random: pick the elements of L randomly with uniform probability from N.

2) closest: pick the elements of L to be the elements of N closest to the new entering node n in the network topology.

3) hybrid: pick some elements random an others with the closest strategy.

The intuition behind the strategies is the following. The closest strategy gives the algorithm more information to lower relative errors of nodes that are close to the new entering node, while the random strategy will lower relative errors of landmark nodes that are further away from the new entering node. The hybrid strategy should achieve something in the middle.

The PIC algorithm with different strategies were tested in different environments with a variable amount of routers. In the GATech topology 5050 routers were organized hierarchically, in Mercator 102639 routers were used and also organized hierarchically and in CorpNet 298 were used. In each system the PIC algorithm was run on 40000 end nodes.

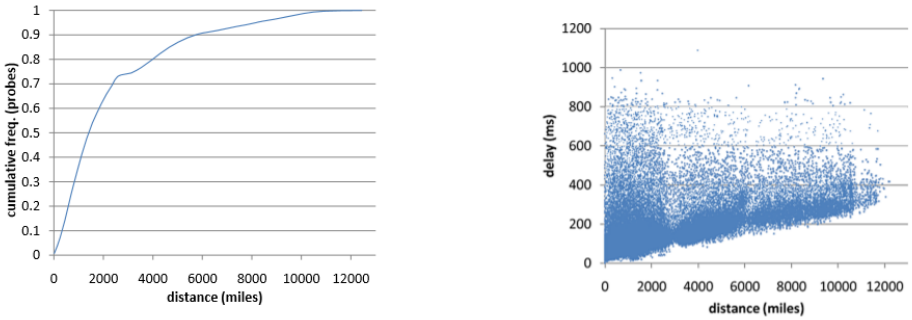

Figure 3.2: The figure on the left shows the cumulative distribution of the distance between consoles. The figure on the right shows the latencies measured for each distance between two nodes in miles.

The results of the experiment show that the hybrid strategy is nearly as efficient as the original GNP algorithm. The random and closest strategy always perform worse in all three experiments. The result tells us that choosing some peers close and some peers randomly gives the best performance of the PIC algorithm in a decentralized setting.

To meet the security requirements of PIC a triangle inequality test is introduced. Triangle inequality holds if for a triple of nodes a, b, c, $d_{a,b} + d_{b,c} \leq d_{a,c}$ where $d_{i,j}$ denotes either the measured network distance or the predicted distance between i and j.

If a node lies about its coordinates or its distance to the joining node, it is likely to violate triangle inequality. For most of the node triplets on the Internet, the triangle inequality holds. If an attacker lies about its coordinates or its distance to a joining node the attacker is likely to violate triangle inequality. The security test may also be useful when dealing with congested network links. When a link is temporarily congested, it will behave like an attacking node that makes the distance large. However, sometimes congestion of a joining node affects all measurements and the number of triangle inequality violations is not increased.

3.5. GEOLOCATION APPROACHES

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

It is hypothesized that the geographic distance between two consoles has a strong correlation with their measured latency. The great circle distance algorithm is used to calculate the distances between two consoles at a different geolocation. The distance between nodes can varies between 0 and 12000 miles. Figure x shows a cumulative distribution function for the distance between nodes. About 14% of the console pairs traversed over 5000 miles. We have enough samples to examine the correlation between distance and delay.

In the right graph of figure X we see a very strong correlation between the geographic

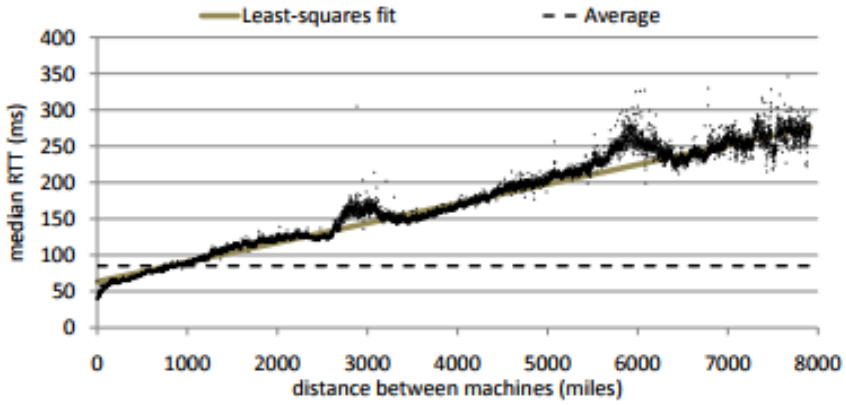

Figure 3.3: The correlation between the distance and latency. The latency data is the median of the data from the Halo 3 players database. The distance data is from MaxMind's IP-to-geo database. There is a clear linear relation between the distance and the median.

distance and the minimum latency measured between two consoles. Above this minimum there is a lot of noise. The geography of ip addresses is a useful predictor for filtering out pairs of IP addresses that are too far apart to have such a low latency.

3.6. HTRAE LATENCY PREDICTION SYSTEM

Htrae is a latency prediction method merging both network coordinate systems (NCS) and geolocation. The way this works is by geographic bootstrapping: initializing NCS coordinates to correspond to the locations of the nodes in actual space. With better initial positions, Internet latencies can be better predicted.

Figure x shows the correlation between the distance in miles and latencies. When the median is taken at each distance a linear relation can be seen from figure x. The least-squares fit line is also drawn in the figure. The slope of the line is 0.0269 ms/mile and the explained variance is 97,6% ($R^2 = 0.976$). The explained variance percentage is high, so there is a strong linear relation. rmediate route.

When a new machine enters the system the Htrae algorithm works as follows. At first, the IP-address is looked up in the commercial MaxMind's IP-to-geo database. This gives an initial geo-location for the NCS. A Vivaldi-like algorithm is then used where a node moves in the direction of the forces that pull on the new node by nearby coordinates. The Vivaldi algorithm is adapted to use spherical coordinates instead of a linear euclidean space to better model the spherical shape of the earth. An uncertainty model is also added that is used to calculate how strong a force to apply when updating coordinates: the greater a moving node's uncertainty, the stronger a force will be. Uncertainty is defined as the difference between the observed and calculated latencies.

The Htrae system implements additional things to improve the algorithm such as Triangle Inequality Violation (TIV) avoidance and autonomous systems correction. When

updating a nodes coordinate, Htrae will skip the coordinate update if the measured latency exceeds the predicted latency by some number δ to remove TIV's. A big difference in the estimated latency and predicted latency is usually caused by inefficient routing between two nodes. This causes a large delay between them compared to the sum of delays via some other intermediate route.

4

INCREMENTAL ALGORITHMS

In order to solve the complexity problems of the GNP algorithm in the decentralized Tribler setting we introduce an incremental algorithm approach to stretch the computation of the solution over time. With incremental algorithms the input changes over time. Given a sequence of input, the algorithm calculates an output sequence. At each new time point when a new input vector is given to the algorithm new solutions are calculated. According to Sharp, 2007 we can further specify the algorithm class to online incremental algorithms. Online algorithms differ from normal incremental algorithms in that there is no knowledge on future input while in normal incremental algorithms there is complete knowledge. [**?**] [**?**]

A good problem to to use as an example what online algorithms are is the k-server problem. Figure 4.1 illustrates the k-server problem. Suppose there are k reporters who have to travel to and investigate on news events in a country. Every time a new news event happens one of the reports is chosen by the algorithm to go toward that event and to investigate on it. The goal of the algorithm solution is to minimize the sum of the distances that all reporters travelled. When the algorithm decides on which reporter to send towards a new event it does not know about the locations of future events. This lack of knowledge results in sub-optimal solutions in the above example. [**?**]

4.0.1. INCREMENTAL ALGORITHMS AND THE PEER DISCOVERY MECHANISM

Peer discovery is constructed in such a way that it allows easy incorporation of an incremental algorithm. To show this we will first explain how peer discovery works in Tribler.

In the dispersy implementation of the peer discovery mechanism a request and response mechanism is build to test the communication between two peers. The result is a list of peers called the neighbouring list that contains peers to which the peer always can exchange data. The communication lines between two peers in the neighbouring list are symmetrical by nature. If peer A has peer B in its neighbouring list, peer B also has peer A in its neighbouring list. Both peers A and B assume the role of client and server in the P2P network. To let the peer discovery mechanism work on the large scale of the internet, random computers have to be able to communicate to each other on the

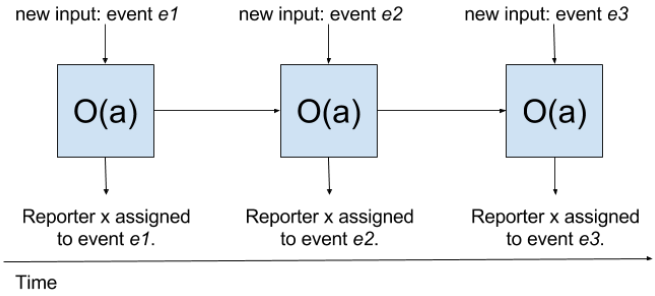

Figure 4.1: Illustration of K-server online incremental algorithm. At each new input event e a calculation is done in $O(a)$ time where a is a polynomial function to decide which reporter x to assign on event e. Past solutions can be used in future calculations.

internet.

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

Network Address Translation (NAT) is also designed for the client-server model and not suitable for a P2P setting. Figure 4.2 gives an overview of the NAT protocol. 64% of the computers connected to the internet do Network Address Translation (NAT) to hide the IP and port combination of computers from a local network to the internet. The ip addresses and ports of the local peers 1, 2 and 3 are hidden from the peer on the internet with the NAT box. The NAT box has two IP addresses. One is available for the local network and one for the internet. The peer on the internet only communicates with the NAT box and the NAT box translates the ip,port combination to a peer from the local network. The peer on the internet cannot distinguish between the three local peers if it wants to address one of the local peers and send messages to it. Therefore the local peers always have to act as clients and initiate the connection. The NAT box identifies and remembers the peer that initiated the connection and makes the translation for the peer on the internet that gives a response to the NAT box. The peer on the internet can never initiate a connection and is forced in the server-role. [**?**]

To directly message a peer of a local network the NAT box has to be punctured. The puncturing is integrated in dispersy in the peer discovery mechanism. There are four phases in the peer discovery mechanism of Tribler. These four phases are also illustrated in figure 4.4. These four phases represent one step and multiple steps are a walk. By walking each peer discovers a set of known peers that are that peers neighbourhood.

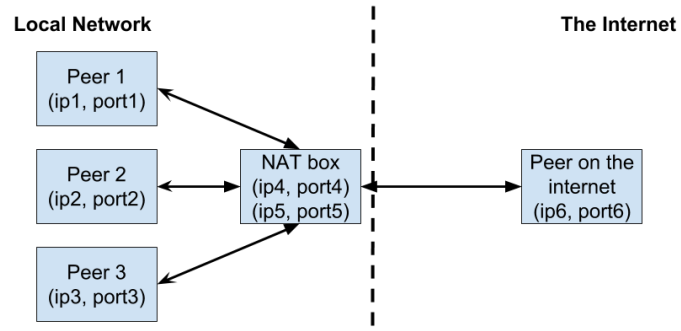

Figure 4.2: Network Address Translation (NAT). The NAT box has two ip, port combinations. ($ip4, port4$ is available on the local network and $ip5, port5$ is available on the internet.

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

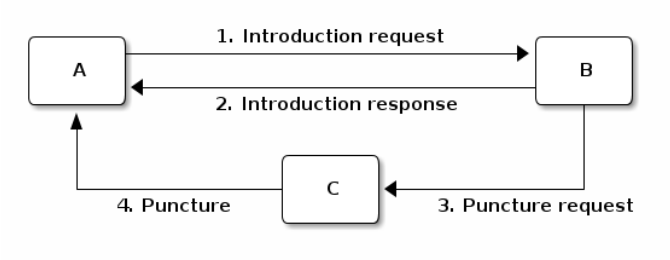

Figure 4.3: Overview of peer discovery in Tribler

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

ECLIPSE ATTACK

The current node selection process in Tribler is build to prevent against the eclipse attack or routing table poisoning. In the eclipse attack an attacker can gain partly or complete control over the data that is received by a victim node. This is achieved by manipulating the candidate lists of the victim and its neighbours. When selecting a node it is important to take into consideration that attacker nodes might become part of the candidate list. If the colluding attackers control a large part of the neighbourhood of a victim node they can "eclipse" victims by dropping or rerouting messages that attempt to reach them. In the case of complete control over the neighbours of a victim peer (all neighbours are colluding attackers) the attackers gain full control over all the traffic toward the victim. [**?**]

Candidate lists can be easily manipulated with the well known Sybil attack. The Sybil attack is not equal to the eclipse attack because an attacker is not necessarily bounded to use the Sybil attack and might use other attacks. By creating a large number of pseudonyms that are colluding, the attacker can force to populate the neighbouring lists of victims by only introducing other pseudonyms to the victim. If a victim accidentally selects an attacker node, the attacker node introduces other attacker nodes which then introduce again other attacker nodes until only attacker nodes are in the victim neighbouring list.

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

1) Engineering block races A block race occurs in a block-chain when two miners discover blocks at the same time. One of these miners receives mining rewards for that block and his block will become part of the block-chain while the other miner will be ignored and create an "orphan" block. Attackers can forge block races by holding back mined blocks that are mined by eclipsed miners. Once a non-eclipsed miner discovers a competing block the block mined by the eclipse miner is released later resulting in an orphan block for the eclipsed miner.

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

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

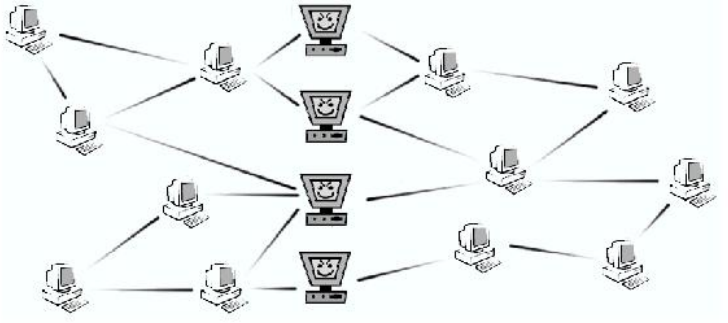

Figure 3.1: An Eclipse Attack: the malicious nodes have separated the network in 2 subnetworks.

Figure 4.4: Separating a network with the Eclipse attack

miners left.

4) 0-confirmation double spend In a 0-confirmation transaction the attacker exploit systems where a merchant gives a confirmation of the transaction to a customer before the transaction is verified by the block-chain. This happens sometimes in systems where it is inappropriate to wait 5-10 minutes before a transaction in a block gets confirmed. For instance in the retail service system BitPay or in gambling sites like Betcoin. The coins spend by the customer to the merchant is double spend by the attacker. The attacker first eclipses the merchant. When the merchant wants to confirm transaction T as payment for the goods of the customer, the attacker double spends the bit-coins in the network with transaction T' but sends an confirmation of T to the merchant. Because the merchant is eclipsed he can never tell the network about T. When the attacker is the customer he can rewire the money back to himself with T' and thus not pay for the goods. This attack has happened in a real world situation.

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

4.0.2. Robust Node Selection

To prevent eclipse attacks a dispersy node will divide his candidate list into three categories:

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

Nodes that have replied to an introduction-request message are put into Category II, while the node they introduce is put in Category IIIb. Nodes that have send us an introduction-request are placed in Category IIIa. A special list of predefined nodes, i.e. trackers is put in the trusted node category. A node which was introduced to us moved from Category IIIb to Category II after a successful connection attempt.

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

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

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

Halkes et al (Bron) measured NAT timeouts we remove nodes from the neighbouring list after a certain amount of time. Introduced nodes are removed after 25 seconds and nodes that are send to or received an introduction-request from after 55s are also removed. In combination with a step time of 5 seconds the average node degree becomes around 11 seconds. (Bron)

4.0.3. LOW LATENCY NODE SELECTION

In the low latency-overlay, neighbours should be selected and introduced to other peers that have a low latency toward that other peer. Various algorithms will be discussed later that that estimate what would be the latency between two peers in a P2P network. The low latency overlay does not always have to perfectly introduce the peer that has the lowest latency toward the peer that did the introduction-request. If the introduced peer is one of the lowest latency neighbours but not the lowest, but the accuracy of the overall algorithm is still good the algorithm can still be successful. EVEN HERSCHRIJVEN

To still maintain the protection against the eclipse attack the low latency overlay has to be incorporated in the current node selection process. The use of the groups have a dampening effect on the eclipse attack. The oldest node is currently selected from the groups to prevent NAT-timeouts. In the low latency overlay the nodes with the lowest latency from the particular group are selected. This does not happen for the trusted nodes, as these nodes are a fixed group. The NAT-timeouts do not become a problem because introduced nodes are removed after 25 seconds and nodes that are send an introduction-request or where introduction-requests were received from are removed after 55 seconds from the neighbouring list. Nodes stay for such a short time in the neighbouring list that NAT-timeouts do not become a problem. The selection process of low latency's in the latency overlay does not always guarantee the lowest latency, but the prevention against the eclipse attack is still maintained.

4.0.4. Performance of incremental algorithms

The performance of an online algorithm can be analyzed by comparing the solution to the optimal solution which can be calculated offline. The optimal solution of an online algorithm can be computed offline with complete knowledge.

4.0.5. Block-chain in problem description

4.0.6. Low Latency overlay

P2P overlay networks are distributed systems without any hierarchical organization. There is no centralized component in a P2P overlay. An overlay network is an overlay over the Internet Protocol (IP) offering a various features such as trust and authentication, attack resilience or anonymity. The dispersy overlay features node selection, eclipse attack resilience, message authentication verification and cryptography and NAT puncturing. In this work we want to add latency preference in peer selection. When a peer $p1$ introduces a peer $p2$ to another peer $p3$, it is preferred that the latency between $p2$ and $p3$ is low. In order to achieve this $p1$ needs to make an estimation of the latencies other peer have with each other. $p1$ can than choose $p3$ in such a way that the latency between $p2$ and $p3$ is low. In this work we will compare various algorithms on how to calculate these latencies and how to choose peers to introduce in such a way that the latencies between a peer and its neighbouring peers are low. See Figure X.

4.0.7. Incremental algorithm and peer discovery

Whenever a new peer occupies the neighbouring list after a step of the peer discovery mechanism, a new input event $e1$ happens for the incremental algorithm. The new peer adds new latency data to the algorithm such that a better latency estimation can be made for the introduction of peers to other neighbours.

5

System Design

5.1. Low Latency Overlay

Each node can measure the latency to every other node in the swarm. This results in an NxN matrix of latencies measured in the swarm with the total number of nodes in the swarm is N. Every node measures the latency from itself to its neighboring nodes and a node can also remember the latencies measured from past neighbors. A node maintains a list l of latencies measured from itself toward other nodes in the system. Because a node constantly discovers new peers a node also discovers new latencies from himself to other peers in the system. $|l| < N$ because the number of latencies measured in l is always smaller than N as there are no duplicate latencies measured. As nodes leave the neighborhood of a peer, their measured latency becomes outdated. Old latencies have to be removed from the list l of known latencies. In the beginning the size of l will become larger because more neighbors will be introduced. At some point l will not grow larger as at that moment in time the rate of number of latencies that are outdated and removed from l is the same rate as the number of latencies that add to l because of new discovered neighbors.

5.2. New Algorithms

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

5.2.1. GNP with N landmarks

A variant of the GNP algorithm is implemented with $N = L$. This means all nodes in the system are landmarks. Only the first step of the algorithm is applied. A node will collect latencies from neighbours and neighbours of neighbours to make the NxN matrix of latencies measured more dense. It is likely that the more dense the matrix is, the better the algorithm will perform. After latency collection the following minimization is calculated to compute the coordinates of nodes.

$$f_{obj}(C^S_{L_1},...,C^S_{L_N}) = \sum_{L_i,L_j \in \{L_1,...,L_N\} | i>j} = \epsilon(f(C^S_{L_1},C^S_{L_2}),f(C_{L_1},C_{L_2}))$$

where $\epsilon(.)$ is the error measurement function: $\epsilon(f(C^S_{L_1},C^S_{L_2}),f(C_{L_1},C_{L_2})) = f(C^S_{L_1},C^S_{L_2}) - f(C_{L_1},C_{L_2})$ and $C_{L_1},C_{L_2},...,C_{L_N}$ are all the nodes that are known to the node.

Tribler is not able to distuinguish between landmark and non-landmark nodes. Therefore, only the first step of the GNP algorithm is applied and any node in the system can be a landmark. In a normal implementation of GNP the number of landmarks would be around 20 or 30. Because the sum function is two dimensional and the number of landmarks can grow large, the calculation of the minimization can becoming very expensive.

5.2.2. SIMPLE INCREMENTAL ALGORITHM

To lower the computative intensity, incremental algorithms are used that are fed latency data in steps. In every step a new row of the NxN latency matrix will be known. In the simple incremental algorithm only the position of this new node will be calculated every step. The way a new node's position is calculated works is similar to the PIC and NPS algorithm described in the literature. Let's say the position of the node that represents the second row in the NxN matrix needs to be calculated. The algorithm will then minimize the following function.

$$\sum_{i \neq 2} \epsilon(f(C^S_{L_2},C^S_{L_i}),f(C_{L_2},C_{L_i}))$$

Note that only the columns of the NxN matrix are added to the minimization function of which latencies are actually measured. This prevents that nodes who are not probed for latencies by the new node do not have an influence on the position of the new node. The incomplete information as a result of unkown latencies by nodes not being probed can have a negative effect on the accuracy of the algorithm. In the experimental section is the algorithm tested in both a local environment with complete information and in the decentralized tribler environment with incomplete information.

5.2.3. INCREMENTAL ALGORITHM WITH R RANDOM REPEAT

The Incremental algorithm with N random repeat adds another feature to the simple incremental algorithm to increase its accuracy. After each new step where the new node's position is calculated R random nodes that were already calculated in past steps are recalculated based on the new information that has been added since then to the incremental algorithm. Since the last calculation the positions of new nodes have been updated and new latencies have been probed by the nodes that have been calculated in the past.

The update and recomputation of the coordinates of nodes calculated in the past should not weigh to much on the computation time of the algorithm. In the experimental section we will test with various numbers of R to see its impact on the computation time and accuracy. It will be most likely that a larger R will increase the accuracy but lower the computation time. A good design choice for R will depend on the results of these experiments.

5.2.4. INCREMENTAL ALGORITHM WITH R FIXED REPEAT

With a random repeat of node updates some nodes are updated more frequently than others. A structured repeat of coordinate updates of past calculated nodes is implemented to further improve the accuracy of the N random repeat algorithm. The structured repeat ensures that all past calculated nodes are updated once before a past calculated node is updated another time. In this way no nodes are left behind in updating and no nodes are updated more frequently than other nodes. The computational cost of this algorithm will be the same as the R random repeat version because again the coordinates of R nodes are updated with the same minimization function.

5.2.5. INCREMENTAL ALGORITHM WITH R FIXED REPEAT AND TRIANGLE INEQUALITY VIOLATION METHOD

In this algorithm the Incremental Algorithm with R fixed repeat is used with uncertainty corrections to solve the Trianble Inequality Violations. The problem of Triangle Inequality Violations is solved in a similar way as in the Htrae system. Nodes that have a large uncertainty, e.a. the computed distance differs from the measured latency distance by a constance δ, are simply removed from the minimization function. Other solutions are versions where high uncertainty nodes have less effect on the node coordinate.

The calculation of the uncertainty and removel of uncertain nodes takes some additional computation. The effect on the accuracy and computation is compared to the other algorithms is further evaluated in the experimental section.

6

PRIVACY SYSTEMS

6.1. CHAUM MIXES

Chaum , D.L. first published about anonymization techniques in 1981 now known as *mix networks*. [**?**] The purpose of mix networks is to unlink the sender and receiver of messages. A mix is a node in the network with its own public/private key pair. Messages are send towards mixes encrypted with the public key of the mix. The mix hides the correspondence between incoming and outgoing message. To achieve this the mix does three things:

1. Incoming messages are batched together and send in one batch.
2. The mix strips of the encryption layer of incoming messages with its private key and forwards messages to another mix or to the final destination node of the messages.
3. The order of the messages is permuted.

A mix network is a series of mixes connected together. More mixes in the network make the unlikability property stronger but result in a higher latency.

The identity of the next recipient in the network is encrypted together with the message to let the mix know to which node it has to send the next batch.

$E_{MIX}(message, A) \xrightarrow{MIX} message, A$

Thus the encryption for a mix network of three layers looks the following.

$E_{MIX_1}(E_{MIX_2}(E_{MIX_3}(message, A), MIX_3), MIX_2), MIX_1) \xrightarrow{MIX_1} E_{MIX_2}(E_{MIX_3}(message, A), MIX_3), M$

$E_{MIX_2}(E_{MIX_3}(message, A), MIX_3), MIX_2) \xrightarrow{MIX_2} E_{MIX_3}(message, A), MIX_3), MIX_2$

$E_{MIX_3}(message, A) \xrightarrow{MIX_3} message, A$

Because messages are batched together mix networks require that a (threshold) mix has to wait until N messages are arrived to forward a new batch of messages. This gives a

high latency to the system. In a timed mix the mix forwards every t seconds. If a limited number of messages arrive in the time interval the mix loses its unlinkability property. For instance, if one message arrives in the time interval it can be easily linked to the only outgoing interval. To solve this problem dummy messages with no meaning can be send into the network. Dummy messages also lower the latency and make the unlinkability property in a threshold stronger.

In a *Trickle attack* the adversary can slow down messages that are send into the mix to ensure only one message is send into a timed mix every t seconds. The *Flooding attack* injects $N-1$ messages in a threshold mix and then distinguishes its own injected message from other messages.[**?**] [**?**]

6.2. TOR ONION ROUTING

TOR onion routing is a method developed by Dingledine, R. *et al* that like mix networks also aims to provide anonymity for users but operates at a lower latency compared to mix networks. The onion routers are real time mix networks. Messages are not batched together but passed on nearly in real-time. This makes TOR onion routing vulnerable to the global passive attack where peers sniff all the network traffic and can then link sender and receiver to each other. When only parts of the network can be sniffed, TOR onion routing still provides anonymity.

Clients create a path through the network where each node only knows its predecessor and successor node in the path. The end node connects with the recipient of the messages. Session keys are negotiated between each pair of successive nodes in the path to ensure "Perfect forward secrecy" With "Perfect forward secrecy" a hostile node cannot record traffic and decrypt it later at another compromised node in the network.

6.2.1. PRIVACY OF TRADERS IN MATCHING ENGINE

A matching has to be found by broadcasting the price and quantity details towards other peers. Peers gossip the information towards each other. In this broadcasting process a path between two traders is made via other peers in the peer to peer network. The path creates a tunnel like in the design of the TOR protocol and chaum mixes. The path is used in all future communication between the two traders to ensure privacy. A session key is shared using Diffie Hellman key exchange between the two peers in the tunnel to ensure privacy against the 3 peers that facilitate the tunnel. The session key is shared using Diffie Hellman key exchange. [**?**] [**?**]

The first step in the matching process is the broadcast of a bid or ask towards other peers in the network as shown in Figure 6.1. The price and quantity (qtt) details of the bid or ask are first encrypted with the private key of the sending peer to let the receiving peer make sure the match is coming from the sending peer. A second layer of encryption is added with the public key of the receiving peer to ensure that only the receiving peer can read the information of the match. The match is three times forwarded towards other peers to make the tunnel with three peers into it. The time to live (ttl) field maintains how many times the match is forwarded. Also the first part of the Diffie Hellman key exchange A and a unique random number n_i to distinguish between peers to which

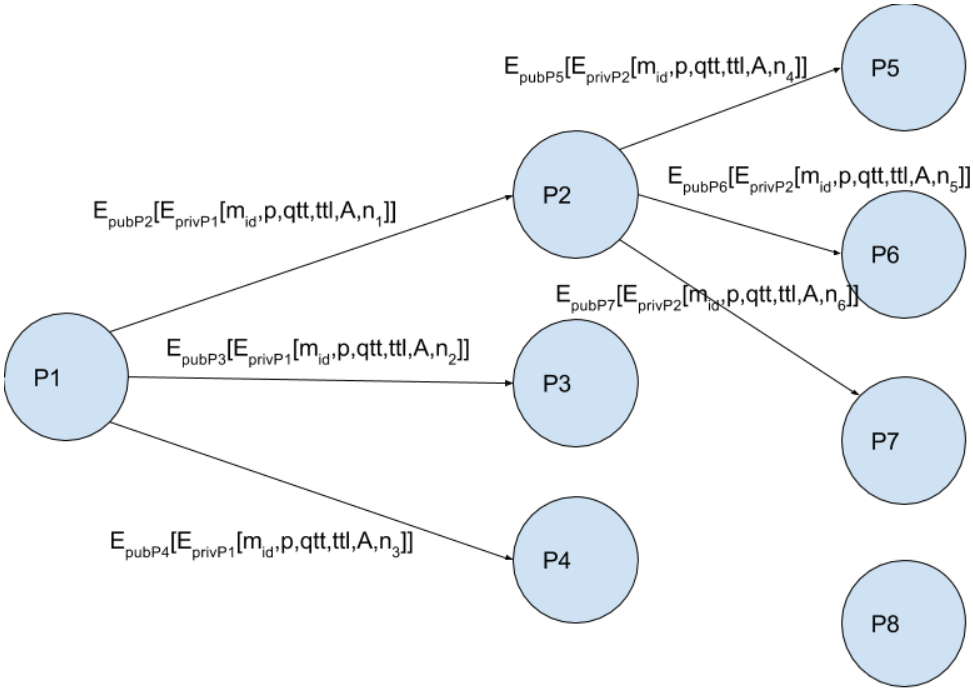

Figure 6.1: Broadcast of bid or ask match request towards other peers.

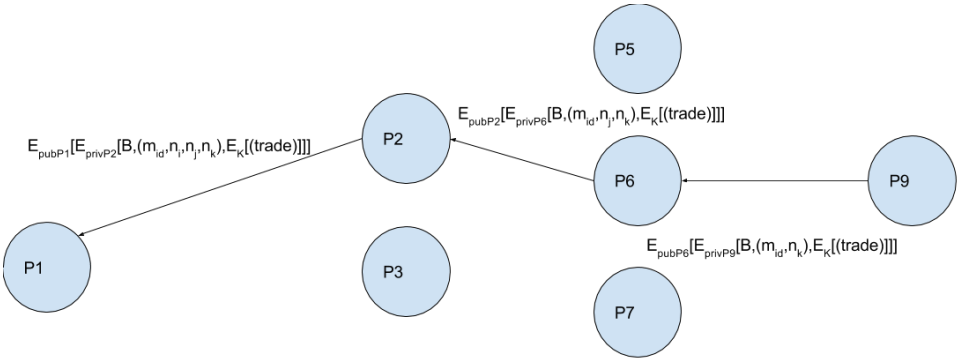

Figure 6.2: Match send back towards broadcaster. The path identifier is created upon hopping back

the match is forwarded is calculated and send with the broadcast. The peer saves the peer to which the match is forwarded in the tuple (m_{id}, n_i) where m_{id} is the match id. For example in Figure 6.1 P1 would save P2 in the tuple (m_{id}, n_i). This information is later used to distinguish between multiple matches made with one broadcast. Also the m_{id} is saved tell from which peer a broadcast was coming. For example $P2$ would let m_{id} correspond to $P1$ because the match with m_{id} was coming from $P1$.

When after three hops a match is found the second step starts and the matching peer sends a proposed trade back towards the broadcasting peer via the tunnel. The second part and the session key of the Diffie Hellman key exchange is calculated. Because multiple matches can be made there will be multiple unique session keys. The proposed trade is encrypted with the session key K and is send back into the tunnel together with the second part B of the Diffie Hellman key exchange. The broadcasting peer receives the second part B of the Diffie Hellman key and calculates the session key K to decrypt the proposed trade. The communication to accept a trade, decline a trade or propose a counter-trade between the two trading peers at the end of the tunnel is from this point in time done with the session key that both ends know.

To distinguish between multiple matches in the same broadcast the tuple (m_{id}, n_i) was saved that tells to which peer the broadcast was send. A path identifier (m_{id}, n_i, n_j, n_k) is created on the way back from matched peer to the broadcast peer and can be used to distinguish between paths on the way forward from the broadcast peer. Thus (m_{id}, n_i) tells the first peer who is the next peer in the path. (m_{id}, n_j) tells the second peer the next peer in the path and (m_{id}, n_k) tells the third peer the last peer in the path. The m_{id} is used by a peer to go back toward the broadcasting peer. An overview of the second step is given in figure 6.2

7

EXPERIMENTS

7.1. INCREMENTAL ALGORITHM

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

7.1.1. PERFORMANCE METRICS

To fully evaluate the performance of the incremental algorithm the trade-off between the computational time and the accuracy of the algorithm needs to be explored. Because of the incremental nature of the algorithm the computation is separated over time. Every time a peer explores a new neighbouring peer a new data vector containing the latency's measured by the newly explored peer is added to the latency data-set of the exploring peer. The computational time it takes to process this new data vector can easily be measured by taking the time difference of the time before and after the computation. The accuracy change after each incremental step of the algorithm is harder to measure and requires specifically designed metrics.

We use two metrics to measure the accuracy performance of the algorithm: ranking accuracy and relative error. We will first discuss ranking accuracy. Because we are building a low-latency overlay to select new peers for introduction we are only interested in the closest neighbours of a peer. How good the algorithm selects new peers is measured in rank accuracy. A close related metric is used in the literature to measure the performance of the GNP algorithm [**?**]. Let's say we are interested only in the top 20 of closest peers to each peer. The idea is that after each incremental step we can calculate the predicted distances between peers and know the real distances based on the measured latency's. We then sort the predicted distances and measured distances to calculate a top 20 closest peers list to each known peer for both the predicted distances and measured distances. The ranking accuracy is defined as the percentage of peers that is both in the top 20 list of predicted closest peers and in the top 20 list of the measured closest peers. If the ranking accuracy is 100% accurate then the 20 predicted closest peers are also the top 20 measured closest peers. If the accuracy is only 50% accurate then 50% of the peers

of the 20 predicted closest peers list are also in the top 20 measured closest peers list.

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

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

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

7.1.2. RESULTS OF LOCAL EXPLORATION OF ALGORITHMS

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

In the experiment the location based latency estimation algorithms are tested on an increasing swarm size. Every time the swarm size increases a new iteration starts and a new latency vector is added to the incremental algorithm. The locations of the peers in the 2D graph is updated at each iteration. The amount of time this computation takes is shown in Figure 6.1. After each iteration two metrics for the accuracy of the algorithm are also calculated: "Ranking accuracy" and "Relative Error", the details of the exact calculation of these metrics were described in a previous section. The time needed to calculate these metrics is not included in the computational time measured in Figure 6.1. Comparing the different algorithms on these performance metrics gives a good indication of the performance of the algorithms.

The computational time of the naive implementation grows exponentially, while the computational time of the incremental algorithms grow linear. If the computation time becomes larger than 0.5 seconds, the computation becomes impractical and will block the application. The application will react later or not react at all to new incoming events reducing user experience and increasing the latency between peers. BEWIJS HIERVOOR laten zien. The incremental algorithms also become impractical with increasing swarm size, in particular the Repeat20 and RepeatStructured algorithm. The RepeatTIV and Inc algorithms have relatively low computation time with also large swarm size. This makes them practical to use from computational time perspective.

The RepeatTIV algorithm has the best performance while the naive algorithm has the worst performance. The naive algorithm shows a higher score on the "Relative Error" performance metric and a lower score on "Ranking accuracy" compared to the incremental algorithms. This is surprising as it was expected that the naive implementation gives a more accurate performance as more calculative effort is done to get a good

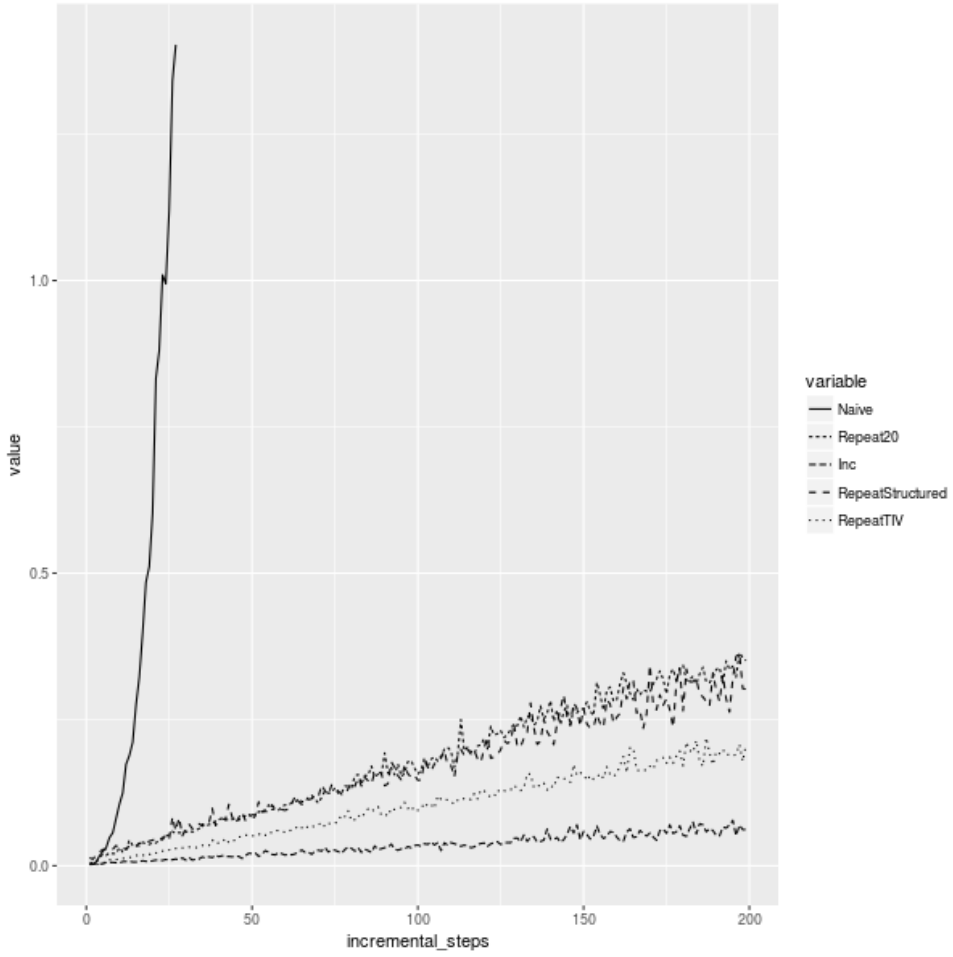

Figure 7.1: Computational time for different algorithms.

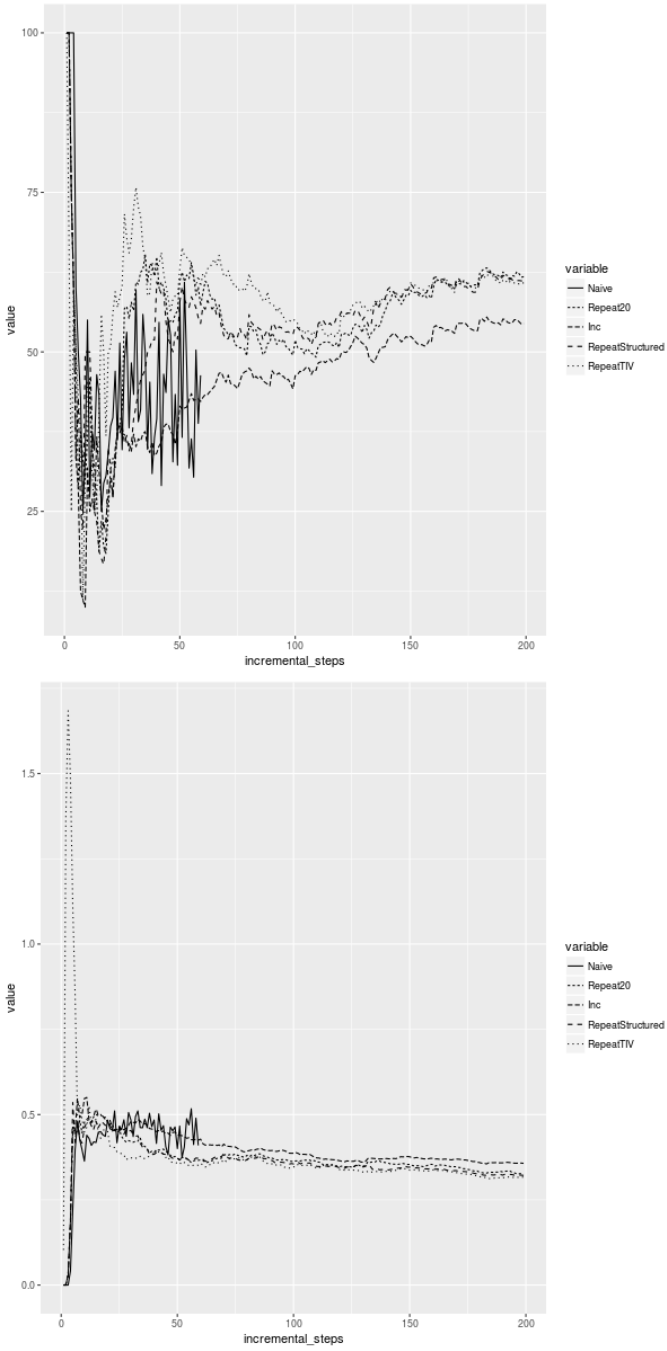

Figure 7.2: Ranking Accuracy and relative error in different algorithms.

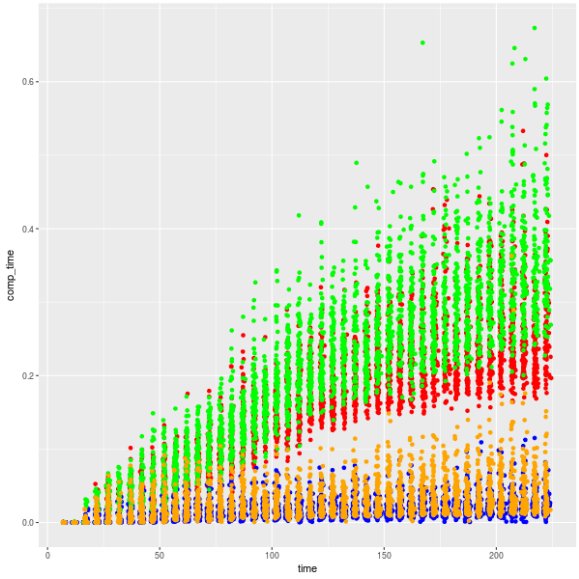

Figure 7.3: All Computational times in Tribler setting. The colors red and blue represent experiment 3 with the naive implementation of the algorithm. Green and orange represent experiment 4 with random choice.

performance. The performance of the incremental algorithms are close to each other on both performance metrics. The larger the swarm size the closer the performance of the incremental algorithms are to each other. The RepeatTIV algorithm has a higher "Ranking accuracy" and lower "Relative Error" with a swarm size below 150 peers. In particular the "Ranking Accuracy" differs and is relatively higher for RepeatTIV. Also RepeatStructured has a slightly better performance compared to Repeat20 and Naive for both performance metrics.

7.1.3. EXPLORATION OF DIFFERENT ALGORITHMS IN DECENTRALIZED TRIBLER SETTING

The computational time of the incremental algorithm increases slightly as the problem size increases but stays short with most computation times below 0,1 seconds. The variation in computation time becomes larger as the problem size increases. Both the ranking accuracy and error seem to converge as the problem size increases. The relative error is larger compared to the naive algorithm. The accuracy metric have a startup period at the beginning of the algorithm when both metrics show large variations across peers.

The difference between the latency overlay and a normal implementation can clearly be seen. The overlay implementation has a lower "Relative Error".

7.1.4. COST OF JOINING A CONVERGED TRIBLER INSTANCE

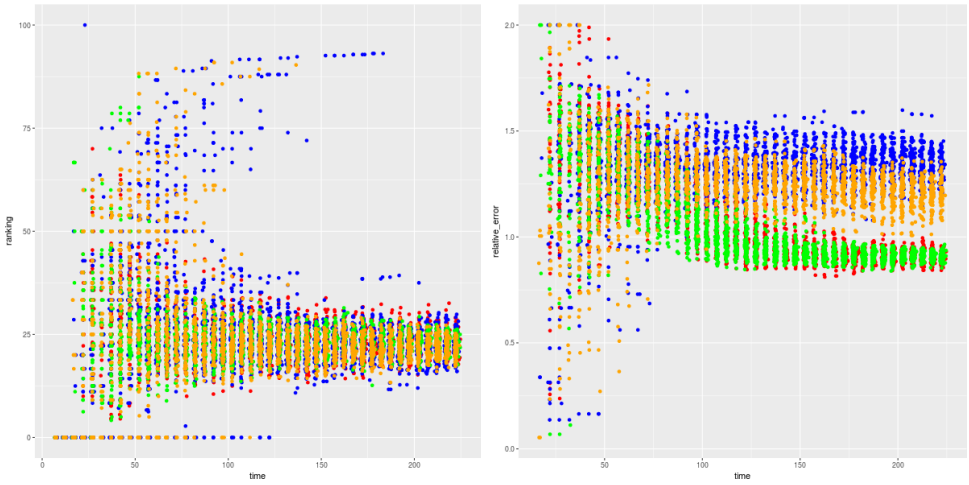

Figure 7.4: Ranking Accuracy and relative error in Tribler setting

8

CONCLUSION

This is a concluding chapter explaining the scientific and technical implications for society of the research findings in considerable detail.