Title

Optional Subtitle

B. van IJzendoorn

Title

Optional Subtitle

by

B. van IJzendoorn

in partial fulfillment of the requirements for the degree of

Master of Science
in Applied Physics

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

Supervisor:	Prof. dr. ir. A. Einstein	
Thesis committee:	Prof. dr. C. F. Xavier,	TU Delft
	Dr. E. L. Brown,	TU Delft
	Ir. M. Scott,	Acme Corporation

TUDelft Delft University of Technology

Preface

Preface…

B. van IJzendoorn
Delft, January 2013

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. [1] 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. [2] 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. [3]

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. [4] An individual trader has numerous advantages when using trading in a system with low latency: [5]

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

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

3. Lower latency traders are served with a higher priority. Offering a lower price gives a trader always a higher priority as other traders would buy a product with a lower price faster. However, when the price is the same. The offer that arrives first is served. A trader with a high latency needs to lower its price in order to get a higher priority. If the high latency trader does not lower its price it is simply not served. Also, offers at the same price level with a higher priority have less adverse selection. [6] [7]

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 2.1. 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). [4]

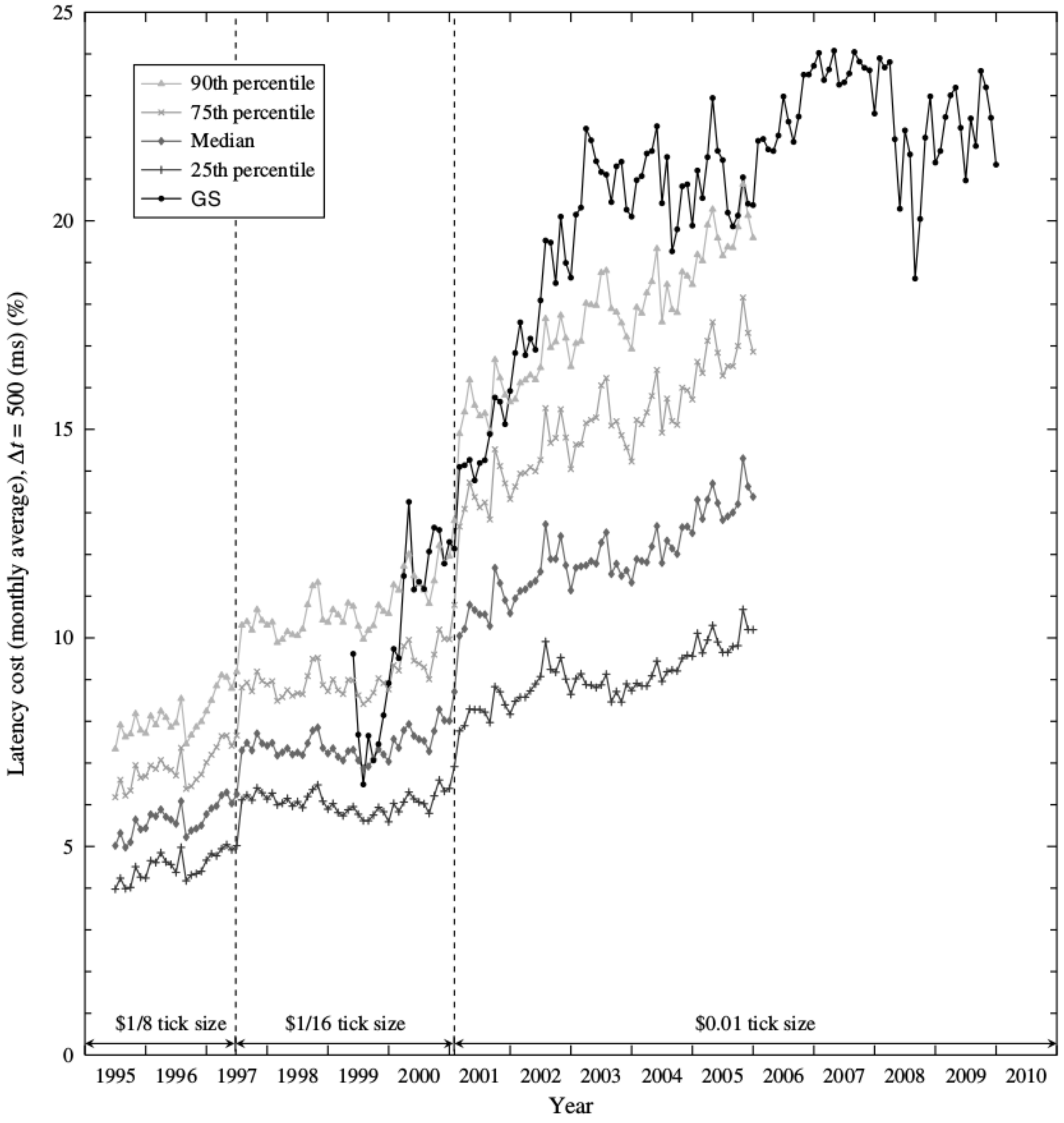


Figure 2.1: A hypothetical investor with a fixed latency of 500 ms is assumed. The latency costs are computed from the data set of Ait Sahalia and Yu (2009) The latency cost for GS is also reported, beginning from its IPO. The dashed lines correspond to dates where the NYSE tick size was reduced. The 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%.

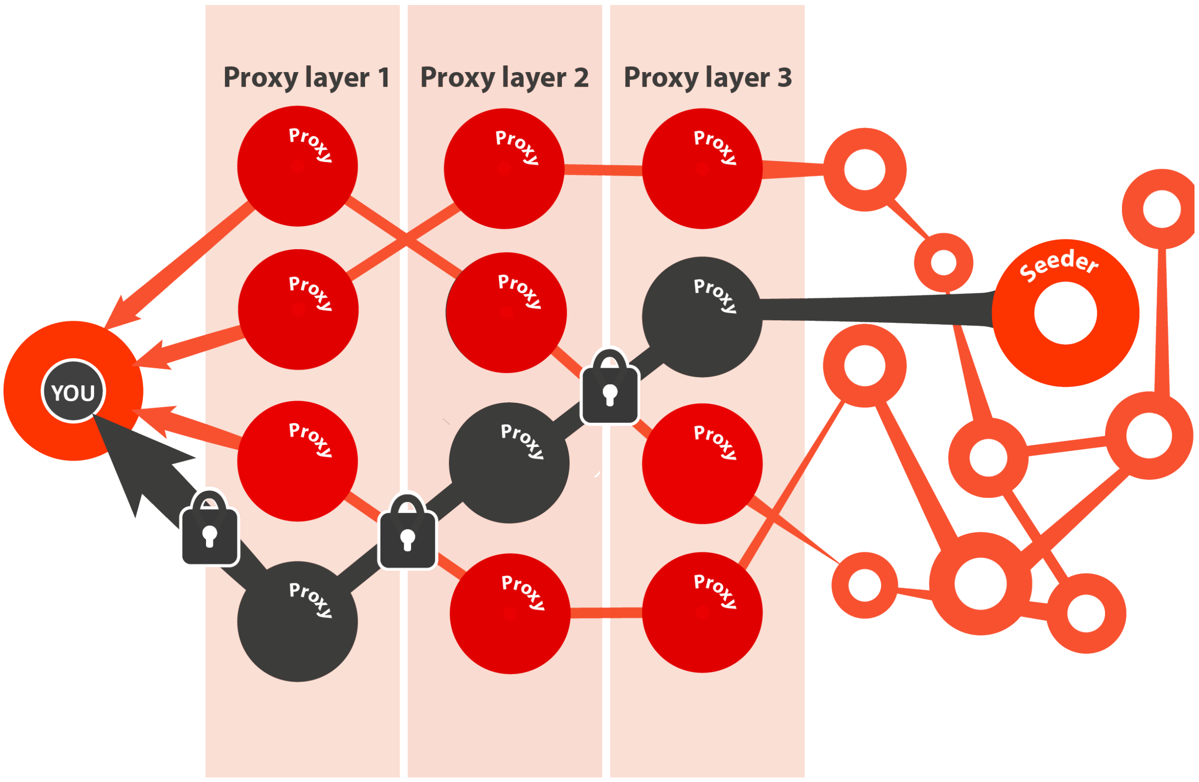


Figure 2.2: Anonymization in Tribler

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 techniques called mixes by Chaum developed in 1981 latency was a big problem. Messages are batched at nodes and a new batch is send forward at a node when n message are received giving a large delay between sending and receiving a single message. [8] 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. [9] Because anonymization requires multiple nodes to which data travels a high latency between these nodes is unacceptable for a good working protocol. Figure 2.2 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. [10] [11]

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

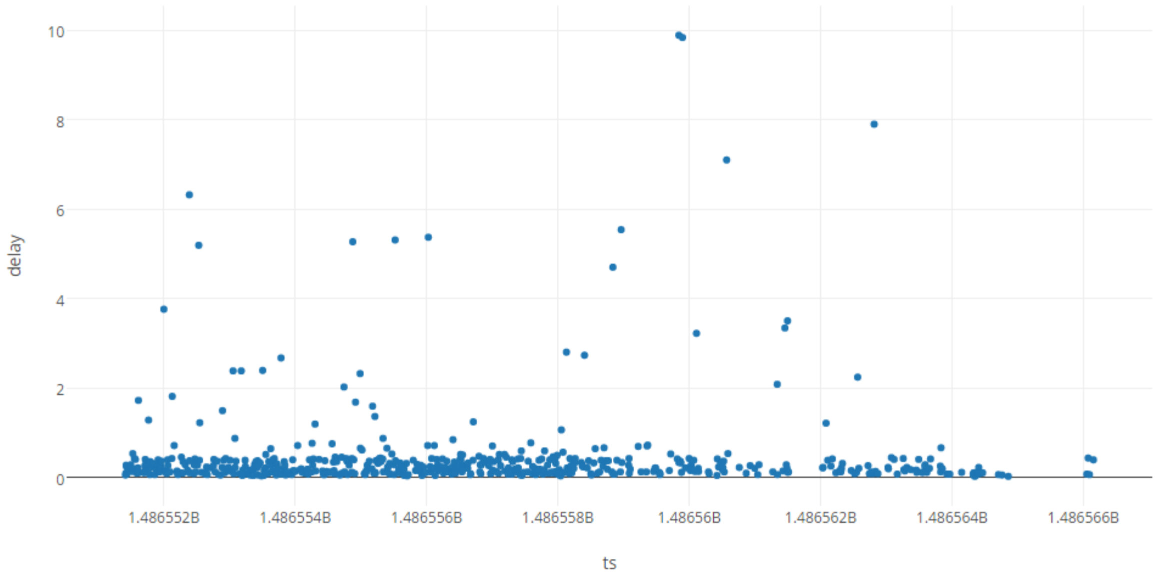


Figure 2.3: Delays while waiting for similarity responses for 4 hours in a real world Tribler application. [12]

3) Low priority serving because another trader gets served earlier due to the first come first served principle. [5]

3
SYSTEM DESIGN

A number of systems have been proposed for estimating latencies by computing the synthetic coordinates of servers. One of the first systems is the GNP system by Zhang et al. It assumes that hosts H are coordindates in a D dimensional geometric space S. Because S is geometric the distance function $f(C^S_{H_1}, C^S_{H_2})$ between two host coordinates $C^S_{H_1}$ and $C^S_{H_2}$ is easily calculated by taking the euclidean distance between these two host coordinates. The GNP algorithm consists of two stages. In the first stage a subset of landmarks L from all the hosts H are chosen as points of reference for fast host position calculation in stage 2. 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 distances and computed distances in S is minimized. Thus, in the first stage the following objective function is minimized:

$$f_{obj}(C^S_{L_1}, ..., C^S_{L_N}) = \sum_{L_i, L_j \in \{L_1, ..., L_N\} | i>j} = \epsilon(f(C^S_{L_1}, C^S_{L_2}), f(C_{H_1}, C_{H_2}))$$

where $\epsilon(.)$ is the error measurement function: $\epsilon(f(C^S_{L_1}, C^S_{L_2}), f(C_{H_1}, C_{H_2})) = f(C^S_{L_1}, C^S_{L_2}) - f(C_{H_1}, C_{H_2})$

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

4
Related Work

4.1. Chaum Mixes

Chaum , D.L. first published about anonymization techniques in 1981 now known as *mix networks*. [8] 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.

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

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

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

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.[13] [14]

4.2. 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.

4.3. TRIANGLE VIOLATIONS

5 System Design

5.1. Latency Community

The latency community has 4 distinct packets. Ping Pong Request Latencies Response Latencies

GNP is often more accurate on a small number of hosts. Calculating GNP with a large number of nodes takes a lot of computation time. Therefore a small number of nodes is required. The only requirement for the nodes is to remove outliers. This should be feasible with a small number of nodes.

5.2. Enhancing privacy

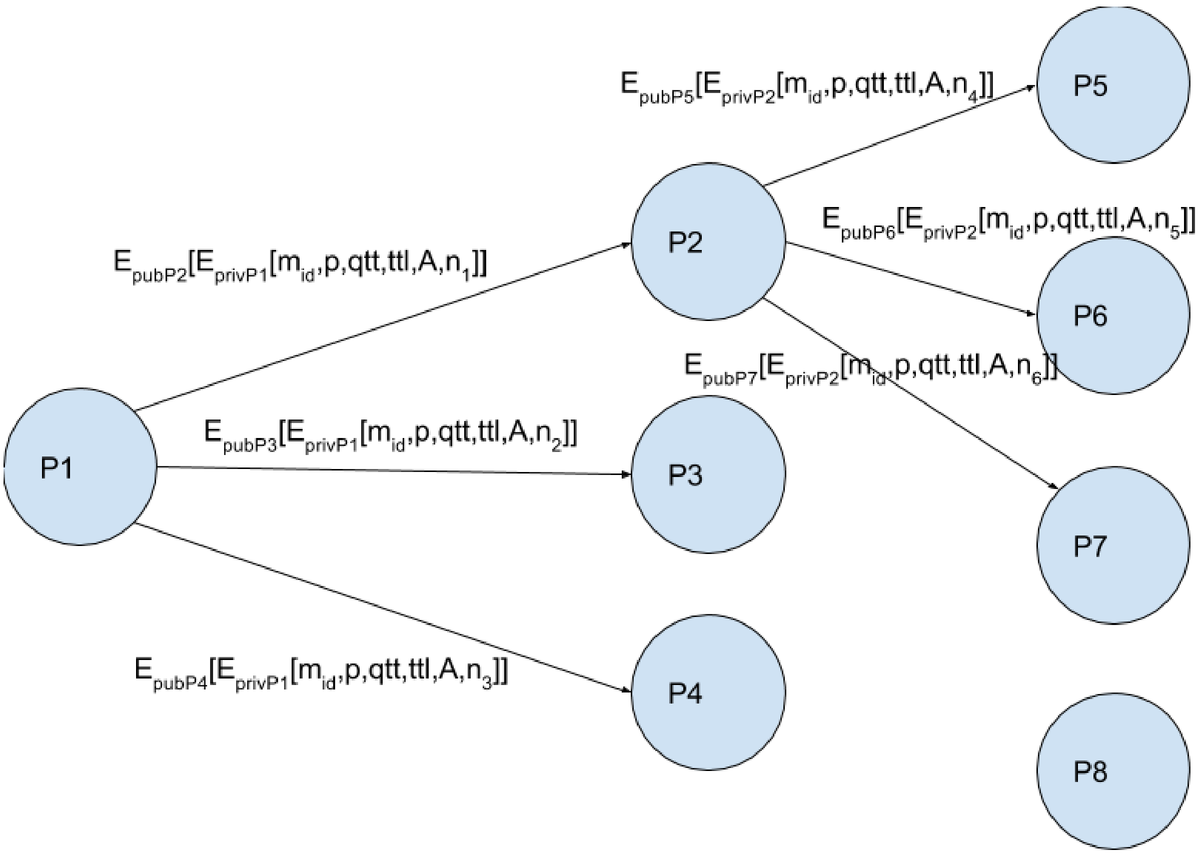
5.2.1. Privacy of traders in matching engine

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. [8] [9]

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 5.1. 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 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 5.1 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 $P1$.

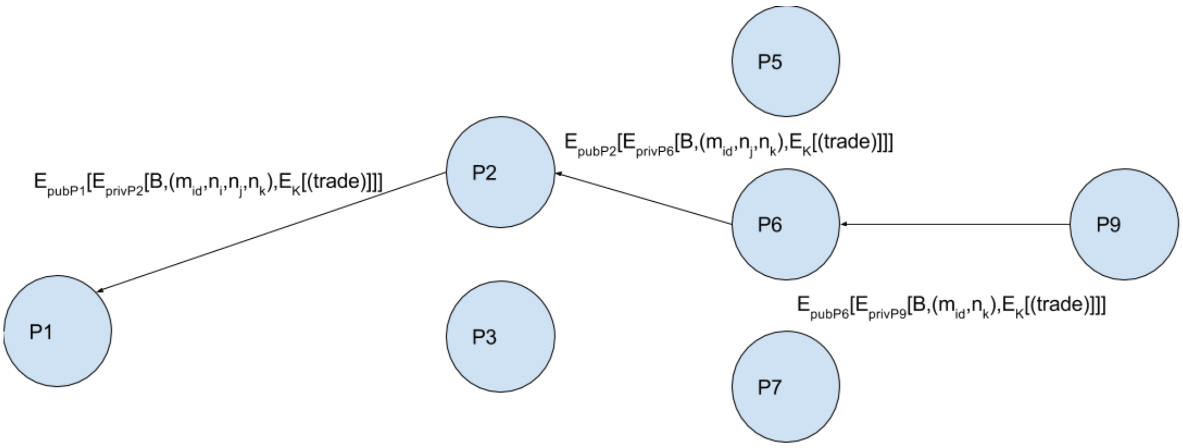
When after three hops a match is found the second step starts and the matching peer sends a proposed trade back towards the broadcasting peer via the tunnel. The second part and the session key of the Diffie Hellman key exchange is calculated. Because multiple matches can be made there will be multiple 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



$E_{pubP5}[E_{privP2}[m_{id},p,qtt,ttl,A,n_4]]$
$E_{pubP6}[E_{privP2}[m_{id},p,qtt,ttl,A,n_5]]$
$E_{pubP2}[E_{privP1}[m_{id},p,qtt,ttl,A,n_1]]$
$E_{pubP7}[E_{privP2}[m_{id},p,qtt,ttl,A,n_6]]$
$E_{pubP3}[E_{privP1}[m_{id},p,qtt,ttl,A,n_2]]$
$E_{pubP4}[E_{privP1}[m_{id},p,qtt,ttl,A,n_3]]$
P1
P2
P3
P4
P5
P6
P7
P8

Figure 5.1: Broadcast of bid or ask match request towards other peers.



P5
$E_{pubP2}[E_{privP6}[B,(m_{id},n_j,n_k),E_K[(trade)]]]$
$E_{pubP1}[E_{privP2}[B,(m_{id},n_i,n_j,n_k),E_K[(trade)]]]$
P2
P6
P9
P1
P3
$E_{pubP6}[E_{privP9}[B,(m_{id},n_k),E_K[(trade)]]]$
P7

Figure 5.2: Match send back towards broadcaster. The path identifier is created upon hopping back

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 5.2

5.2.2. REQUIREMENTS

- Proxies in trading to ensure privacy.
- No Trader id in first offer.
- Encrypted trader id in proposed trade.
- Reputation system to prevent DDoS attacks.
- Reputation reward upon contributing in a proxy trade.
- Only trade with low latency peers to prevent DDoS attacks.

5.2.3. OPTIONAL REQUIREMENTS

- Multiple relays in proxies
- Use proxies in negotiating proposed trade.

5.2.4. TESTS

- DDoS test with a large number of peers doing trade offers to the system.

5.3. 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. [15] [16]

A good problem to to use as an example what online algorithms are is the k-server problem. Figure 5.3 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. [15]

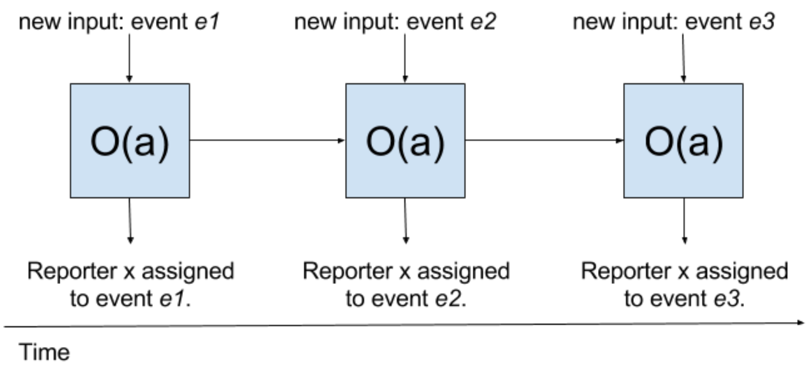

new input: event e1
new input: event e2
new input: event e3
O(a)
O(a)
O(a)
Reporter x assigned to event e1.
Reporter x assigned to event e2.
Reporter x assigned to event e3.
Time

Figure 5.3: 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.

5.3.1. INCREMENTAL ALGORITHMS AND THE PEER DISCOVERY MECHANISM

Peer discovery is constructed in such a way that it allows easy incorporation of an incremental algorithm. There are four faces in the peer discovery mechanism of Tribler. These four fases are also illustrated in figure 5.4.

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;
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.

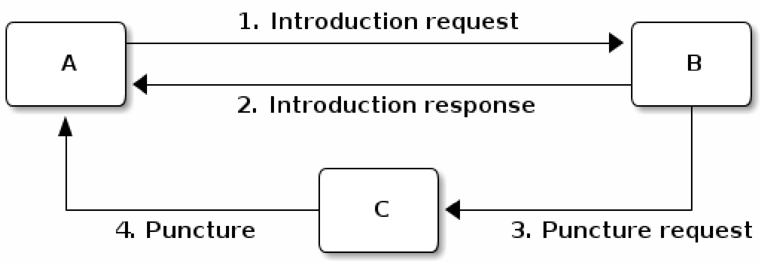


Figure 5.4: Overview of peer discovery in Tribler

5.3.2. 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.

6
EXPERIMENTS

6.1. INCREMENTAL ALGORITHM

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

6.1.1. 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 [17]. 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 [17]. 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.

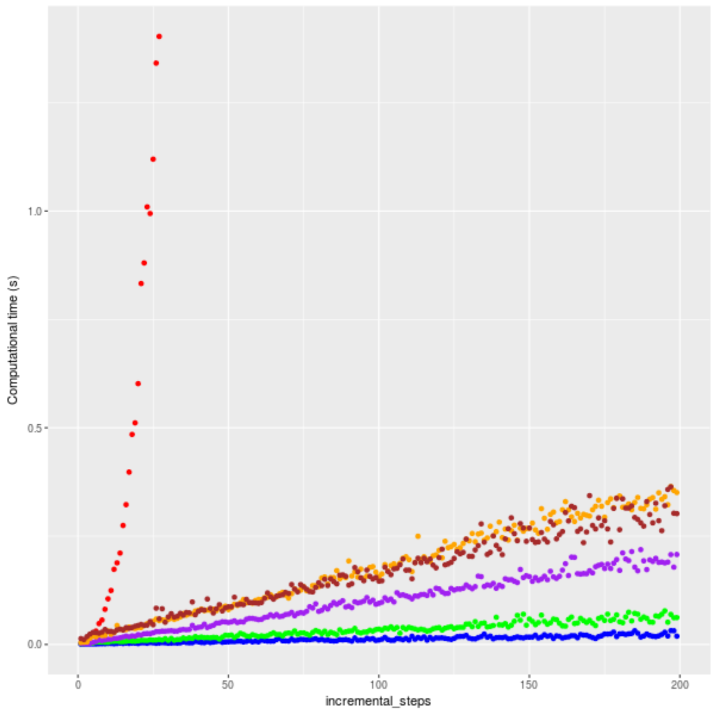


Figure 6.1: Computational time for different algorithms. The Red dots are the naive implementation, blue dots represent the simple incremental algorithm. Green, orange dots are improved incremental algorithms with respectively 10 and 20 random repeats of coordinate calculation at every incremental step. The brown line is an algorithm with systematic 20 repeats every incremental step. The purple line has systematic 20 repeats plus triangle inequality violation correction.

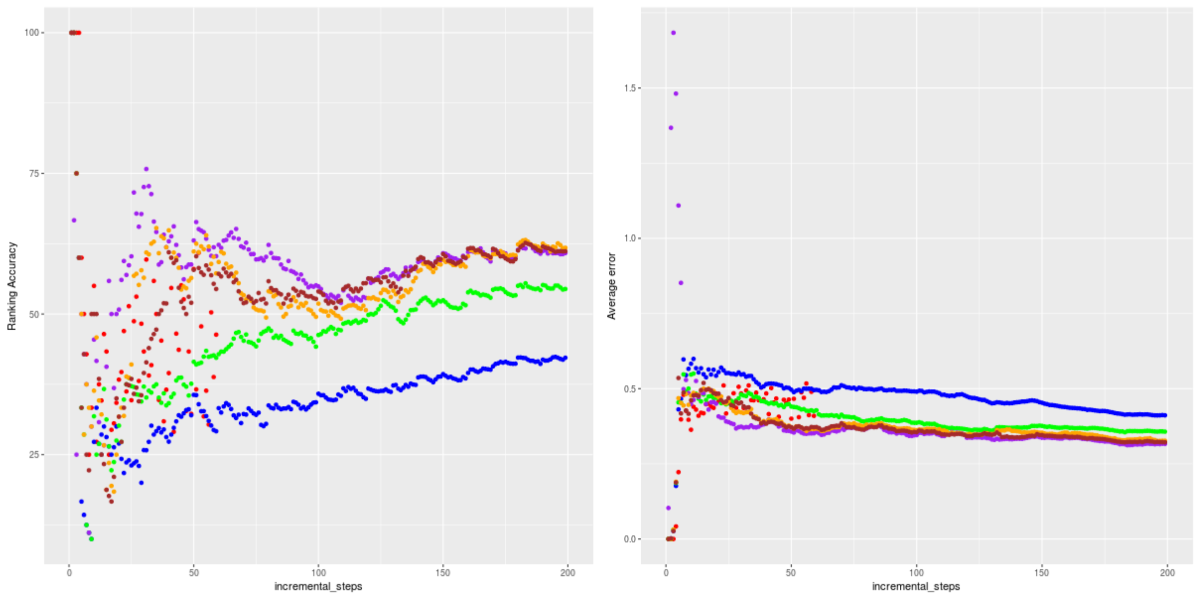


Figure 6.2: Ranking Accuracy and relative error in different algorithms. The colors represent the same algorithms as in figure 6.1

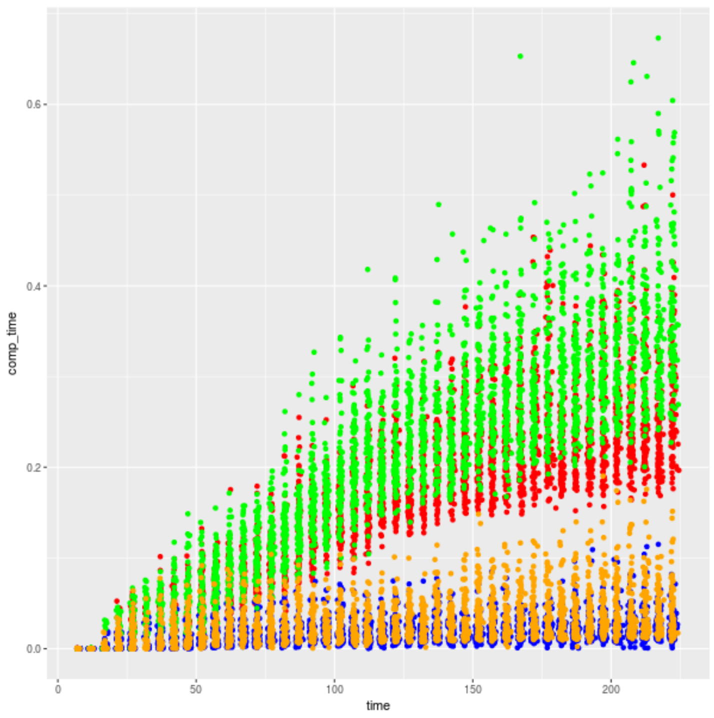


Figure 6.3: All Computational times in Tribler setting. The colors red and blue represent experiment 3 with the naive implementation of the algorithm. Green and orange represent experiment 4 with random choice.

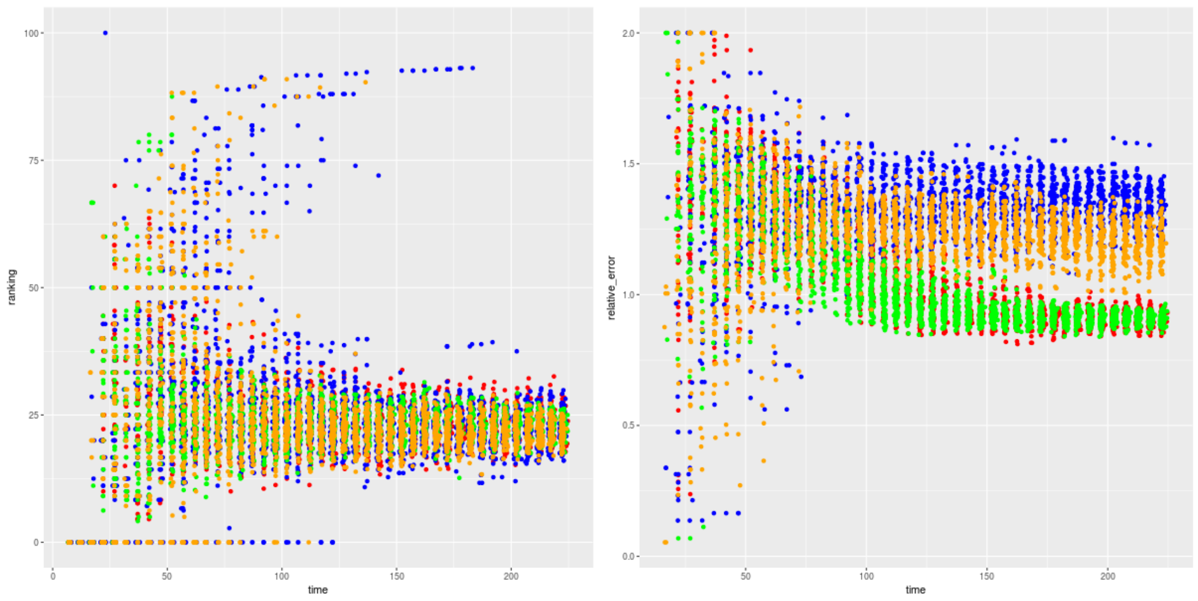


Figure 6.4: Ranking Accuracy and relative error in Tribler setting

6.1.2. Results

6.1.3. Naive implementation experiment

The figures show the computational time and accuracy metrics as the size of the problem increases. In the naive algorithm the computational time grows $O(n^2)$ and becomes larger than 1 second if the number of neighbouring peers and thus the matrix size is larger than 40. The ranking accuracy seems to decrease as the matrix size becomes larger. In the naive algorithm the relative error metric converges to a certain level and does not decrease. It is interesting to note that the relative error seems to stay the same as the problem size increases but the ranking accuracy measuring the selective performance decreases. The ranking accuracy metric has a startup time at the beginning of the algorithm when the matrix size is below 20. This is normal as the ranking accuracy takes the top 20 closest peers list into consideration. When the number of peers known in the system is small, the performance metric becomes insufficient.

6.1.4. Incremental algorithm in Tribler

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.

6.2. LATENCY MEASUREMENTS

In the first experiment the GNP algorithm is implemented and the optimalization is calculated on every node. This gives scalability issues with swarms of more than 50 nodes as can be seen in Figure 5.2 to 5.5. In the second experiment a decentralized version of the GNP algorithm is implemented to solve the scalability problem. The latencies are returning to normal with an experiment of 500 nodes. This can be seen in Figure 5.6. The error function at 50 nodes converges. King Dataset.

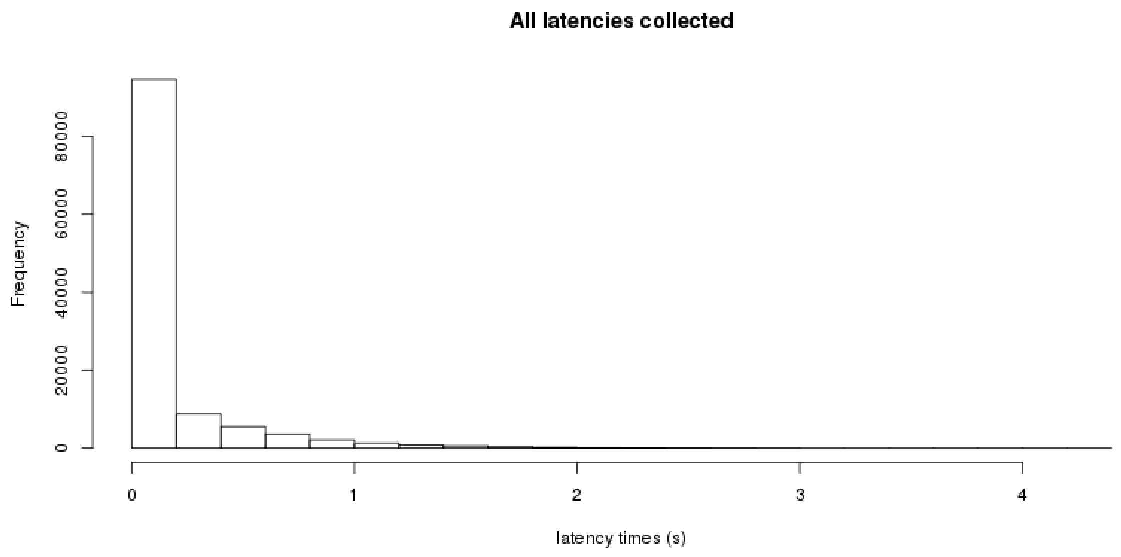


Figure 6.5: Histogram of collected latencies with NPS algorithm and 500 nodes.

The following figures show that applying the GNP algorithm has scalability problems.

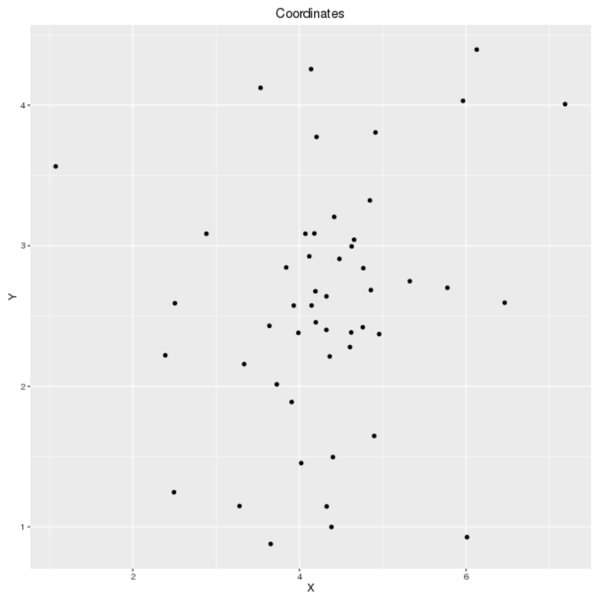


Figure 6.6: Plot of 2D landmark coordinates after applying the NPS algorithm to the king dataset with 50 nodes.

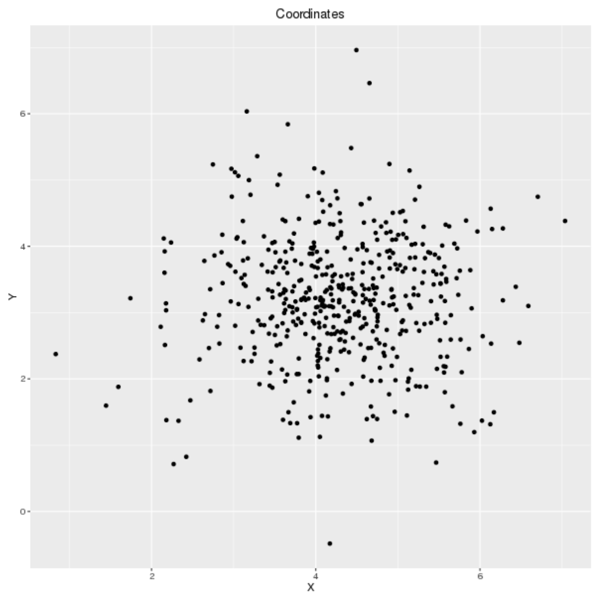


Figure 6.7: Plot of 2D landmark coordinates after applying the NPS algorithm to the king dataset with 500 nodes.

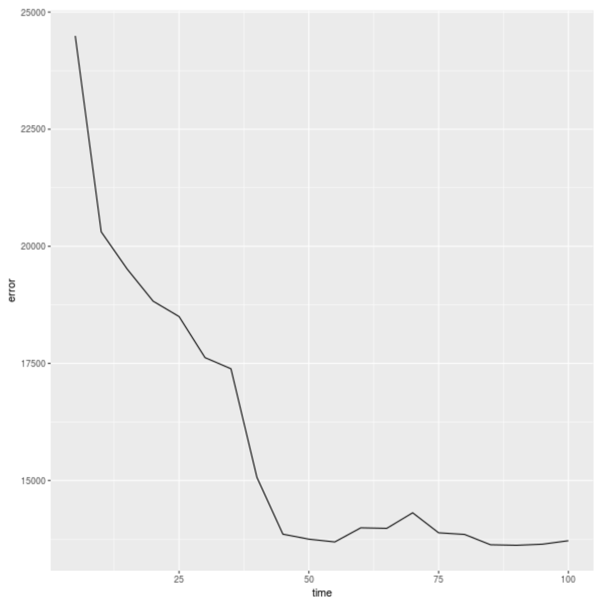


Figure 6.8: Plot of error function over time after applying the NPS algorithm to the king dataset with 50 nodes.

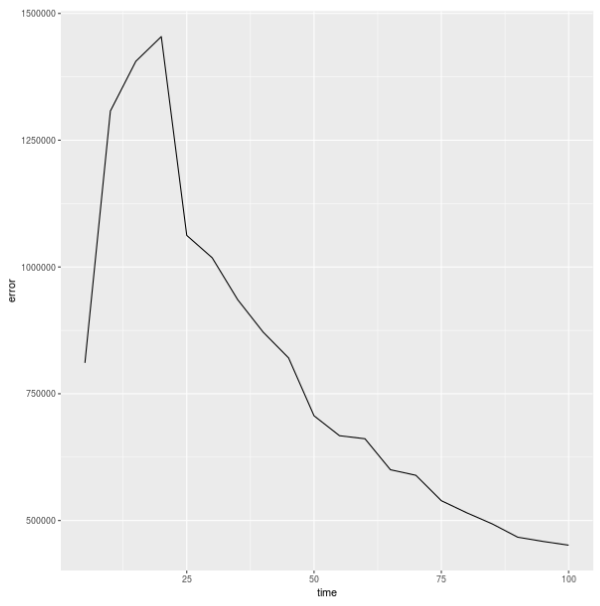


Figure 6.9: Plot of error function over time after applying the NPS algorithm to the king dataset with 500 nodes.

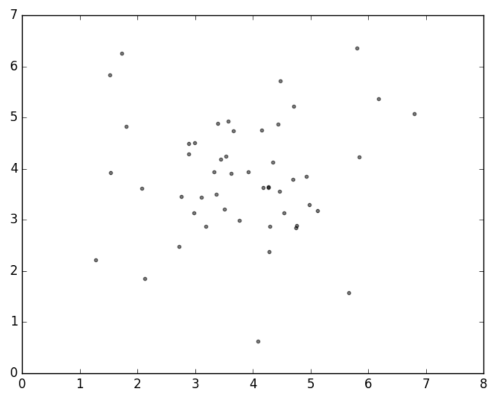


Figure 6.10: Plot of 2D landmark coordinates after applying the GNP algorithm to the king dataset with 50 nodes.

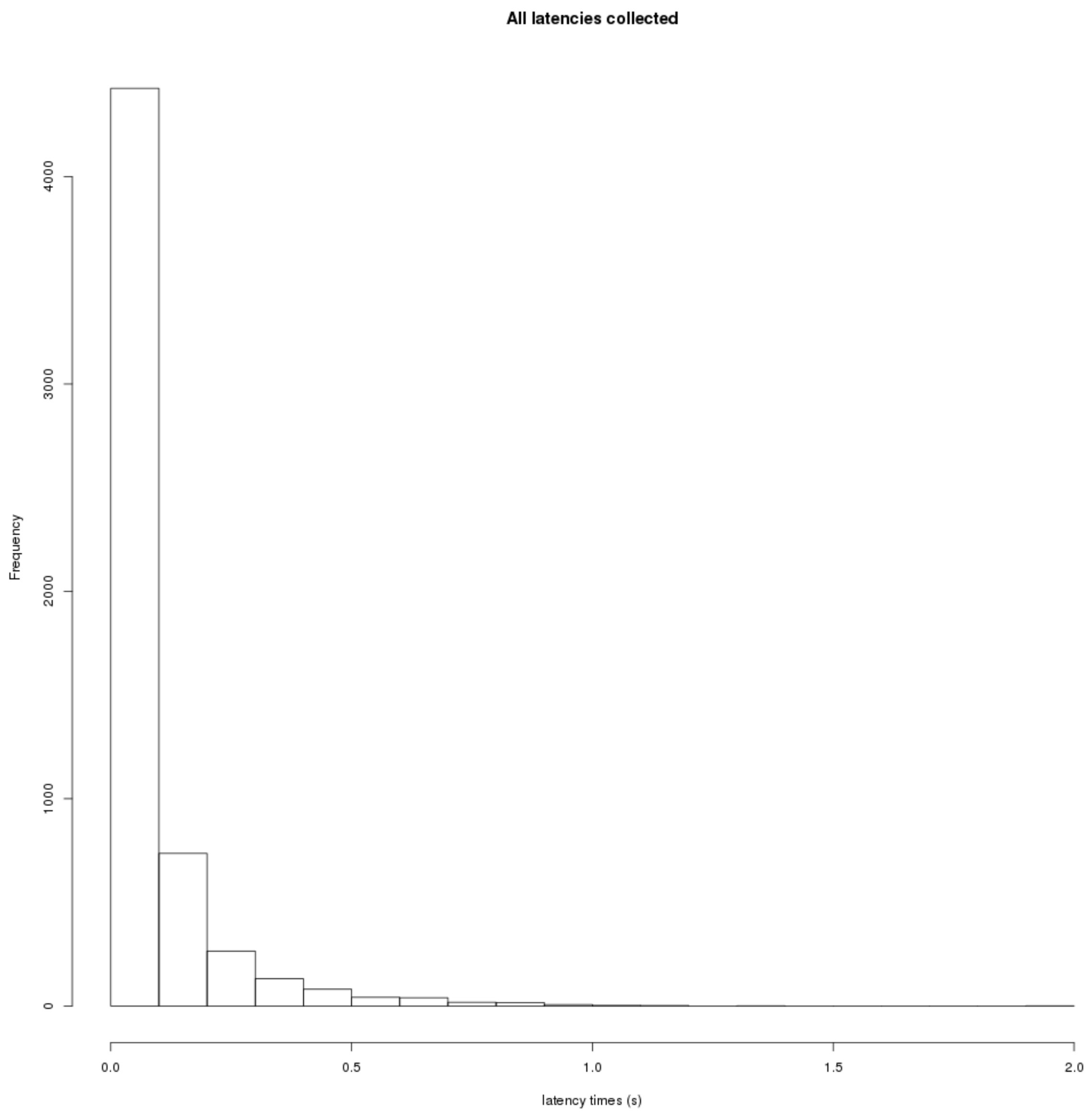


Figure 6.11: Histogram of all collected latencies after application of the decentralized GNP algorithm described by Szymaniak et al, 2004 [18] in a Tribler environment with 75 nodes. The system appears to function normal. The algorithm does not add extra latency to the system.

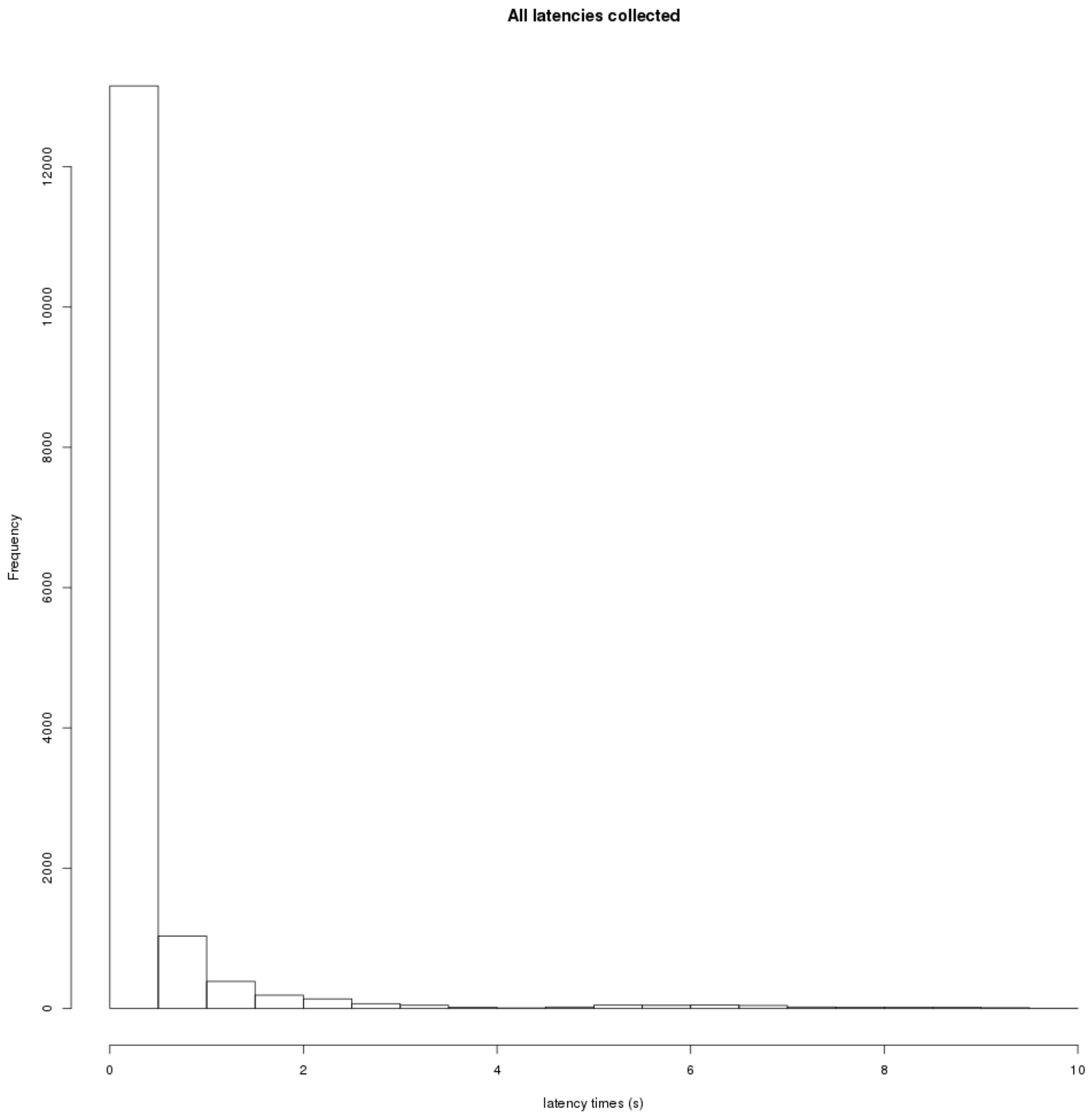


Figure 6.12: Histogram of all collected latencies after application of the decentralized GNP algorithm described by Szymaniak et al, 2004 [18] in a Tribler environment with 100 nodes. The system has some latencies higher than 1 second. The algorithm appears make the system slower.

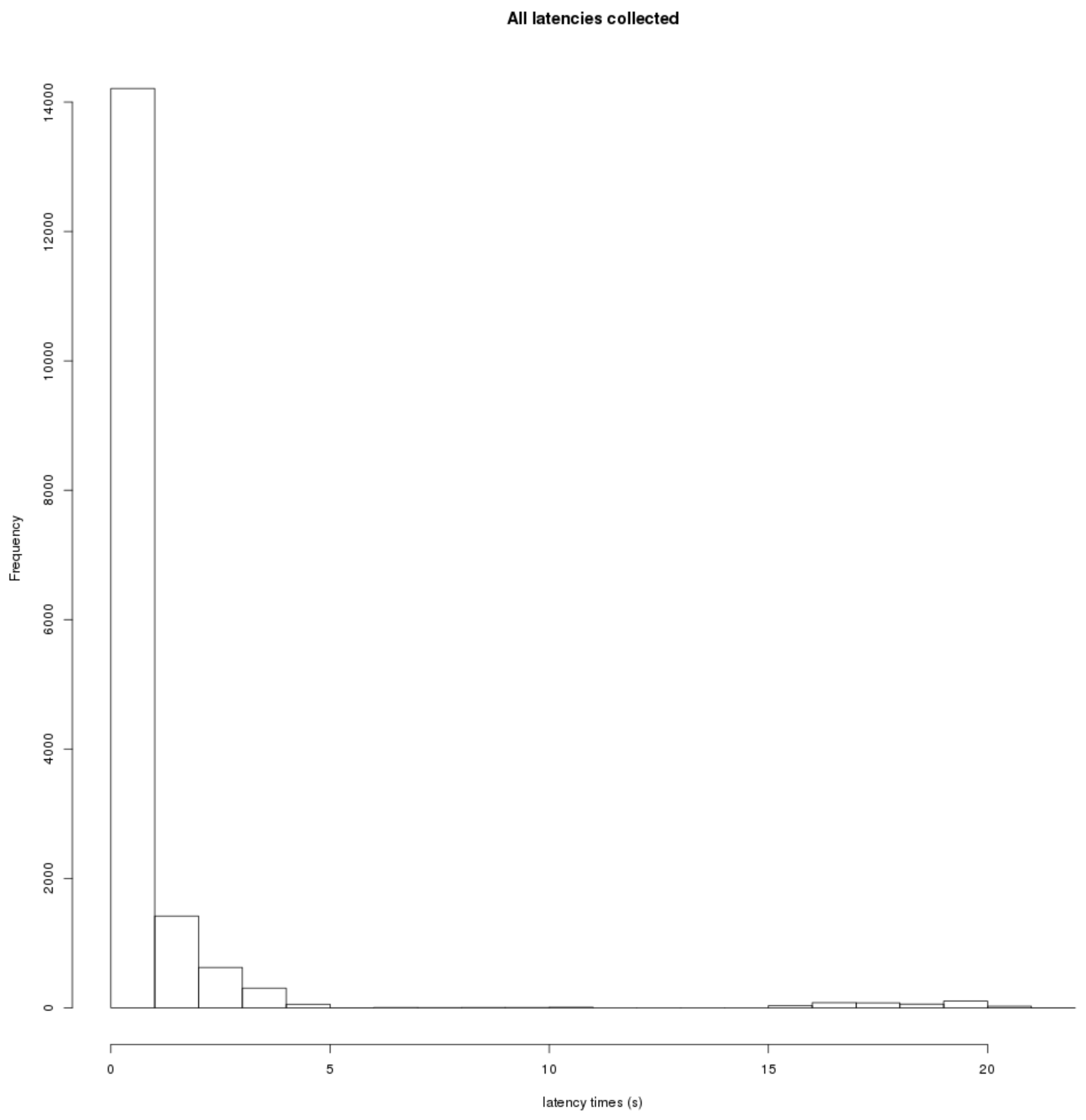


Figure 6.13: Histogram of all collected latencies after application of the decentralized GNP algorithm described by Szymaniak et al, 2004 [18] in a Tribler environment with 75 nodes. The system has a lot of latencies higher than 1 second. The algorithm appears to disrupt the system because of the long calculations.

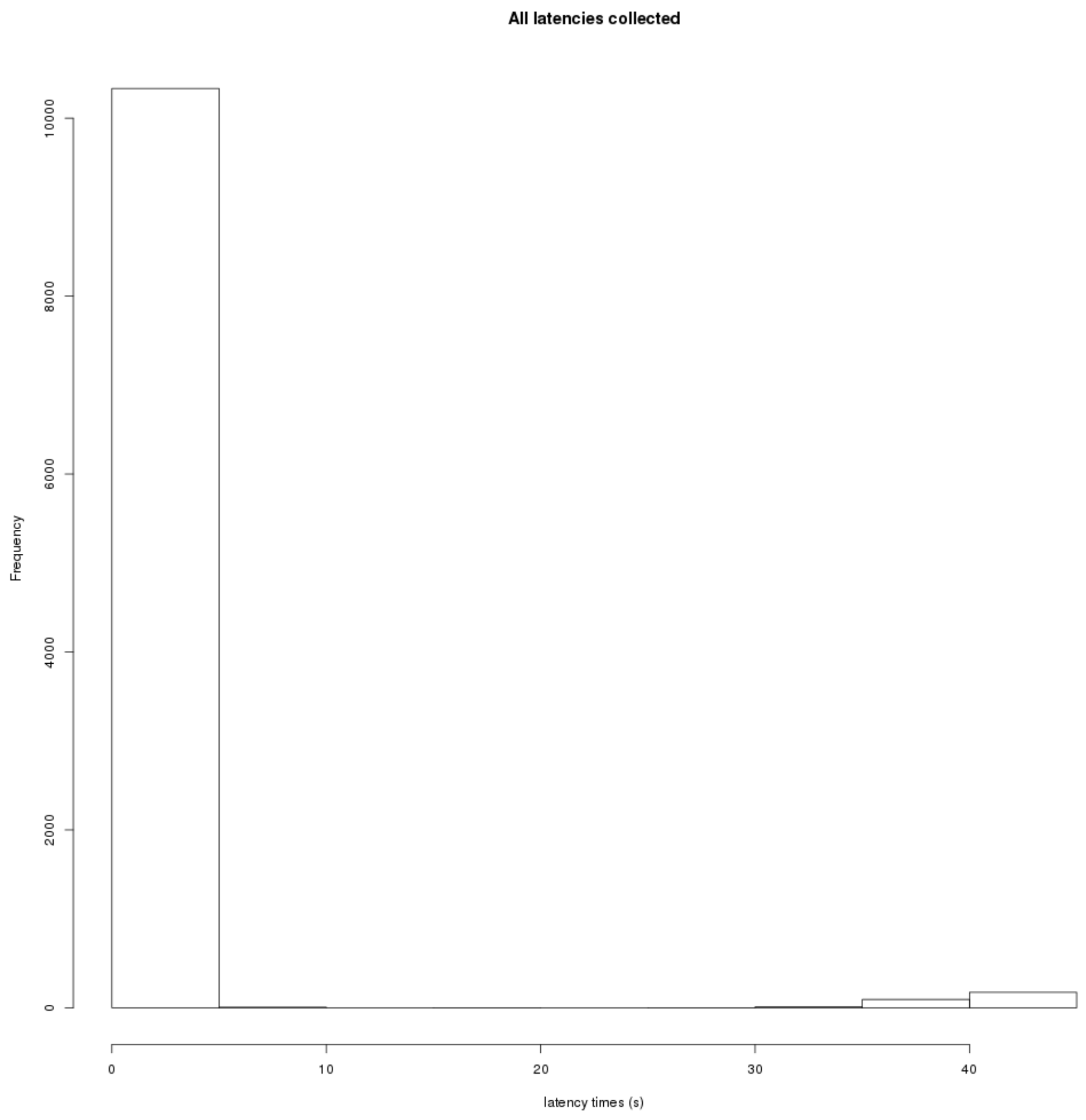


Figure 6.14: Histogram of all collected latencies after application of the decentralized GNP algorithm described by Szymaniak et al, 2004 [18] in a Tribler environment with 100 nodes. The system has a lot of latencies higher than 1 second. The algorithm appears to disrupt the system because of the long calculations.

Bibliography

[1] Olsthoorn, M.J.G.,Winter, J., *Decentral market,* (2016).

[2] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, *Zerocash: Decentralized anonymous payments from bitcoin,* in *Security and Privacy (SP), 2014 IEEE Symposium on* (IEEE, 2014) pp. 459–474.

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

[4] C. C. Moallemi and M. Sağlam, *Or forum—the cost of latency in high-frequency trading,* Operations Research **61**, 1070 (2013).

[5] G. Cespa and T. Foucault, *Insiders-outsiders, transparency and the value of the ticker,* (2009).

[6] L. R. Glosten, *Is the electronic open limit order book inevitable?* The Journal of Finance **49**, 1127 (1994).

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

[8] D. L. Chaum, *Untraceable electronic mail, return addresses, and digital pseudonyms,* Communications of the ACM **24**, 84 (1981).

[9] R. Dingledine, N. Mathewson, and P. Syverson, *Tor: The second-generation onion router*, Tech. Rep. (DTIC Document, 2004).

[10] Y. Yao and N. Zhong, *Potential applications of granular computing in knowledge discovery and data mining,* in *Proceedings of World Multiconference on Systemics, Cybernetics and Informatics*, Vol. 5 (1999) pp. 573–580.

[11] U. K. V. Rajasekaran, M. Chetlur, G. D. Sharma, R. Radhakrishnan, and P. A. Wilsey, *Addressing communication latency issues on clusters for fine grained asynchronous applications—a case study,* in *International Parallel Processing Symposium* (Springer, 1999) pp. 1145–1162.

[12] .

[13] A. Peter, *Anonymous communication,* (2016).

[14] A. Serjantov, R. Dingledine, and P. Syverson, *From a trickle to a flood: Active attacks on several mix types,* in *International Workshop on Information Hiding* (Springer, 2002) pp. 36–52.

[15] A. M. Sharp, *Incremental algorithms: solving problems in a changing world* (Cornell University, 2007).

[16] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules,* Communications of the ACM **28**, 202 (1985).

[17] T. E. Ng and H. 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*, Vol. 1 (IEEE, 2002) pp. 170–179.

[18] M. Szymaniak, G. Pierre, and M. van Steen, *Scalable cooperative latency estimation,* in *Parallel and Distributed Systems, 2004. ICPADS 2004. Proceedings. Tenth International Conference on* (IEEE, 2004) pp. 367–376.