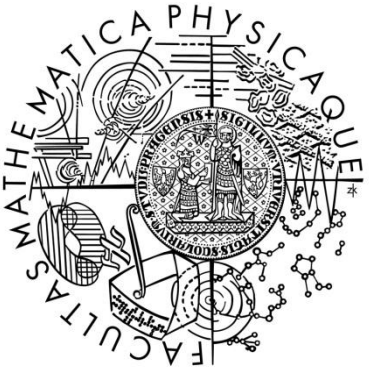
Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS



Bc. Martin Ladecký

Reputation-based Adaptation for Structured P2P Networks

Department of Software Engineering

Supervisor of the master thesis: Mgr. Miroslav Novotný

Study programme: Computer Science

Specialization: Software Systems

Prague 2011

I would like to thank my girlfriend Lucka for her never ending trust and patience. This thesis took me long time which could be spent with her. I will compensate it for you!

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague on Martin Ladecký

Název práce: Adaptace reputation-based managementu pro strukturované P2P sítě

Autor: Bc. Martin Ladecký

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: Mgr. Miroslav Novotný, Katedra softwarového inženýrství

Abstrakt: Cieľom tejto diplomovej práce je implementovať reputačný manažment do smerovacieho protokolu peer-to-peer siete. Táto implementácia je porovnaná so súčasnými riešeniami bezpečnosti v peer-to-peer sieťach – buď žiadna bezpečnosť alebo reputačný manažment, ktorý nie je implementovaný v smerovacom protokole. Zámerom je zvýšiť výkon smerovania a efektivitu bezpečnostných algoritmov.

Klíčová slova: P2P, reputace, síť, bezpečnost

Title: Reputation-based Adaptation for Structured P2P Networks

Author: Bc. Martin Ladecký

Department: Department of Software Engineering

Supervisor of the master thesis: Mgr. Miroslav Novotný, Department of Software Engineering

Abstract: The goal of this thesis is to implement reputation based trust management into the routing protocol of peer-to-peer network. This implementation is compared with current standard peer-to-peer security implementations – mostly no security at all and reputation based management without implementation in the routing mechanism. The objective is to improve routing performance and efficiency of secure algorithms.

Keywords: P2P, reputation, network, security

Contents

1 Introduction

Peer-to-peer (P2P) networks are nowadays playing important role not only in the file sharing but also in the modern way of communication and entertainment. They are very well prepared for a high number of users and their interest in new data. The most used implementation is distributed hash tables.

These networks are not entirely new but they are still in the phase of development – therefore they suffer from different child diseases. Most of them are not secured enough because they have been developed very rapidly. Security is therefore solved as the next layer above routing and sharing layers. This brings problems with efficiency because the routing has to be done before we can say whether it is safe or not.

The goal of this thesis is to adopt reputation based management to the routing mechanism in the P2P network. We should study a current state, problems and then implement reputation based management into the routing protocol. Then we should compare it with the state without any security and with the state where security is implemented as a separate layer.

We introduce concept of P2P in the second chapter. There is also a brief description of mostly used P2P networks. The third chapter is about one particular network – Kademlia. The fourth chapter describes current security approaches.

The fifth chapter is devoted to the simulation environments which are used to simulate large P2P networks. Our main choices and decisions are described in the sixth chapter what makes it very important. We chose security approach described in the next chapter. Its deep description and possible modifications are described in this chapter.

The eighth chapter is about our experiment settings and the results from these experiments are evaluated in the ninth chapter.

2 Peer-to-Peer Network

2.1 Definition

Peer-to-Peer (P2P) network is a distributed application architecture that partitions tasks or workloads between peers. (1) As the name says, the communication is driven directly between end nodes which are called peers. Peer-to-Peer architecture is opposite to the traditional client-server one.

All peers are equal in any sense. They can have different roles such as server or client and they can change the roles as needed. This is a very important fact which influences peers behaviour.

2.2 Motivation

Client-server architecture has several problems which are nicely solved by P2P networks. We will show them in this chapter and discuss pros and cons of each solution.

2.2.1 Single Point of Failure

This term is used to mark a possible weakness in a system. It describes a part which if fails, will stop entire system from working. (2) These points are undesirable because they can be very costly. Therefore redundancy and other mechanisms are in use.

Client-server architecture has one single point of failure (SPOF) by default – it is the server. If it fails, all data, traffic and computations are not available. Take gmail as an example. Gmail is an email service for people and companies. It is run by Google so its reliability should be high. According to (3) it is up most of the time but not always. Microsoft Exchange is even worse. If your company Microsoft Exchange server breaks for any reason then nobody will have access to his emails.

P2P networks have no central point therefore no single point of failure. Any number of peers can be down and network still works. (4)

2.2.2 Scalability

Scalability is a real problem in the case of client-server architecture. All resources in the network are related to the server. These resources are shared by all clients. That

means that more clients the network has, the less resources per client. The resource can be bandwidth, CPU time and other.

Server for video on demand has its fixed network bandwidth and therefore its maximum capacity. If there are more clients then the stream size and video quality are lesser. In the worst case server is unavailable for all clients.

In P2P network, resources are supplied by all peers. The bigger the network, the more resources.

2.2.3 Costs

As always, money is in the first place. Costs for building an entire infrastructure in the case of client-server architecture can be huge. Nice example is YouTube. According to (5) 13 million hours of video has been uploaded only during the year 2010. It is equivalent to 13 million of gigabytes of data storage. That is very costly. Still, it is just storage, not bandwidth, computers and other things you need.

P2P networks seem to have almost no costs – at least not by the producer of data. There are huge data sharing networks where you can upload almost anything and others will share it for free.

2.3 History

The first peer-to-peer file sharing internet service was Napster. Most of the shared files were mp3 audio files. (6) (7) Napster was not pure P2P network – it was based on central catalogue server.

Catalogue server contained list of all files and their location in the network. When user wanted to download a file, he found it in the catalogue and received list of peers which had this file. The data exchange went directly between peers.

The presence of the central server was the weakness of Napster. Not even in the technical view but also in the reliability and law. Not having all necessary copyrights cost Napster its existence. The service was found guilty by the court and shut down. Because the single point of failure was present, whole network collapsed.

This act had one positive effect – P2P networks without central server appeared.

2.4 Architecture

Peer-to-peer systems often implement an abstract overlay network, built at Application Layer, on top of the native or physical network topology. Such overlays are used for indexing and peer discovery and make the P2P system independent from the physical network topology. (1) *Nodes in the overlay can be thought of as being connected by virtual or logical links, each of which corresponds to a path, perhaps through many physical links, in the underlying network.* (8)

All systems we will write about are based on overlay networks. If it is needed we will use whole term, otherwise we will use just term "network". Using overlay networks influences different statistics such as number of hops. One hop in overlay network can mean many hops in the underlying network.

2.5 P2P Network Types

There are two main types of P2P networks – structured and unstructured. Unstructured peer-to-peer networks do not have any rules for links in the overlay network. On the other hand, structured peer-to-peer networks have some rules for making this links.

2.5.1 Unstructured Peer-to-Peer Network Types

Unstructured P2P networks create links ad hoc therefore connection of a new node is fairly simple. Searching in such a network is very slow and unreliable. Because of lack of any structure we can use only basic search algorithms from the graph theory. Very popular is flooding what is sending the message to all nodes we know. They reply or send it to their known nodes (neighbours). As you can see there is a problem with too many messages. It is called flooding because it reminds of wave that is spreading and flooding everything. To stop the flood we need to have some kind of limit of how far the wave can go. Such a limit is usually called time to live (TTL). Because of this mechanism you do not have guaranties your search will be successful. It can't be guaranteed in any unstructured P2P network.

There are three models of unstructured peer-to-peer networks:

Pure P2P Networks

These networks fulfil definition 2.1. Example is Gnutella network till version 0.4.

Centralized P2P Networks

Central servers are in use. These servers are used for bootstrapping, indexing, searching and other things. (7) (9)

Napster is an example of this kind of network. As history showed, this is not a very reliable kind of peer-to-peer network.

Hybrid P2P Networks

There are two types of nodes in hybrid P2P networks – nodes and supernodes. Supernodes have all properties of normal nodes with some special functions. They provide special services such as traffic monitoring and shaping, decision making and other services. Well know example is Skype. (10) (11) Another example is Gnutella after version 0.4 or Kazaa.

2.5.2 Structured P2P Network Types

There are rules for making overlay topology in this kind of networks. Rules can differ a lot but the main purpose is to be able to find a way to every key in the network. The most common type of structured P2P network is distributed hash table (DHT) what is distributed type of consistent hashing. (1) We will discuss DHT later in the thesis.

2.6 Unstructured P2P Networks

2.6.1 Gnutella

Gnutella was the first decentralized peer-to-peer network for file sharing. (12) There was no structure until version 0.4. Searching was possible only with flooding. Peer *A* sent query to given number of neighbours (typically five) and they checked whether they contained given file. If so, file was offered to *A*, else the query was resent to the neighbours of current peer. To prevent network congestion, maximum number of possible resents was set. This number is called number of hops and for Gnutella it was typically set to seven. As mentioned above, this limitations had unwanted effect – even if your file existed, you had no guaranty that it would be found. Popular files were found easily but rare files could not almost be found.

Gnutella with its design was limited to the certain number of peers because of traffic congestion during flooding. One search generated almost two hundred megabytes of

routing data. It scaled exponentially with parameters number of hops and number of neighbours. (13)

Therefore ultrapeers were implemented in the version 0.6. Ultrapeers are kind of local servers or supernodes in our notation. “Normal” peers are connected to small number of ultrapeers (typically three); each ultrapeer is connected to high number of other ultrapeers (typically thirty two). Maximum number of hops is four. Ultrapeers use Query Routing Table of hashed keywords. These changes made Gnutella to be able to expand more up to maximum of three millions of users in year 2006.

2.6.2 Gnutella 2

This network is an example of hybrid peer-to-peer network. It is based on Gnutella after version 0.6. Gnutella 2 is better adapted for big networks. (1) It has system of hubs and leaves. Leaves are connected to two hubs and hubs can accept hundreds of leaf connections. It works very similar to Gnutella therefore we will not write all the details.

Gnutella 2 allows using of old Gnutella protocol but reverse direction is not possible. (14)

2.7 Distributed Hash Table

2.7.1 Hash Table

Hash table is a data structure which uses hash function to map identifying values, known as keys (e.g. a person’s name), to their associated values (e.g., their telephone number). (15) Hash table is often implemented as an associative array. Hash function is used to get index into this array. You can see the principle in the Figure 1.

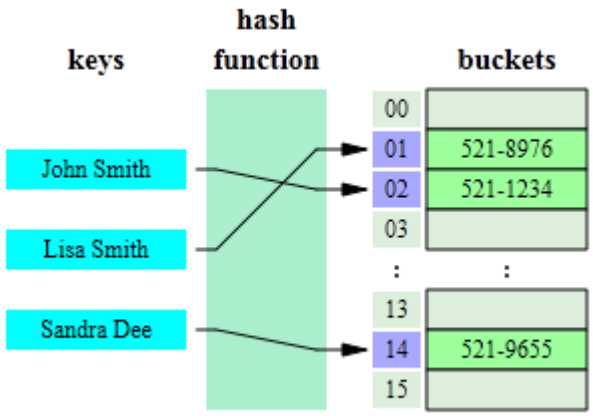


Figure 1: Mapping of name to its telephone number

http://en.wikipedia.org/wiki/File:Hash_table_3_1_1_0_1_0_0_SP.svg

2.7.1.1 Notation

S – Represented set ($S \subseteq U$)

f – Hash function ($f: U \rightarrow \{1, .., n\}$)

A – Array with the size of n

i – Item to be hashed and stored, also called key

In our example above is S the list of all names of our friends. A is the memory with size n. U is any possible combination of letters.

2.7.1.2 Facts

Item i will be stored in the array A at the position $f(i)$. If $f(i) = f(j)$ and $i \neq j$ then a collision occurs. Collision is that more than one item is hashed into the one array bucket.

If there are no collisions at all, we speak about perfect hashing. Achieving perfect hashing is not easy. Even if we have million buckets and only 2500 keys to hash, there is 95% chance of collision. (16) To find an item i means accessing to a bucket $A(f(i))$.

2.7.1.3 Collisions – chaining

Collisions has to be solved otherwise we would not know which value corresponds to which key. Very often used method is called chaining. Each bucket of the array A is a linked list that contains all pairs (key; value) which are hashed to the same location. Main idea is showed in the Figure 2.

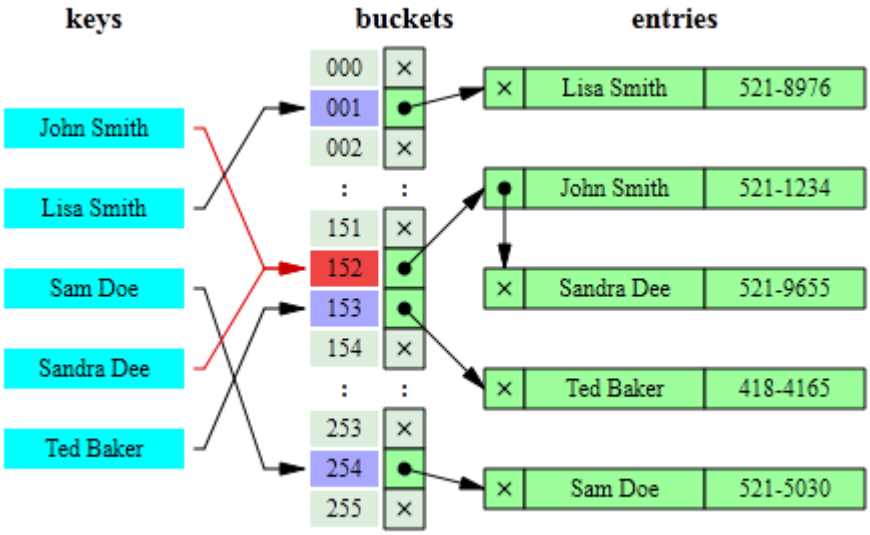


Figure 2: Chaining example

http://en.wikipedia.org/wiki/File:Hash_table_5_0_1_1_1_1_1_LL.svg

2.7.1.4 Consistent hashing

Consistent hashing is a special type of hashing which allow us to add or remove buckets from the mapping array without need to remap all keys. (17) In traditional hashing schemes we have to remap almost all keys with change of number of bucket. Even one less bucket initiates cascade effect and all keys have to be remapped.

Consistent hashing allows us to remap only $|S|/n$ keys. In peer-to-peer network number of all keys can be in millions therefore it is very important.

2.7.2 Definition

Distributed hash table (DHT) is a type of decentralized consistent hashing. Each node in the network has its part of the table. Responsibility for maintaining the table is up to all nodes in the network. This allows DHT to scale to extremely large numbers. (18)

Storing and searching of the item i requires sending message to the node which has control of given part of DHT. Figure 3 shows the principle of DHT.

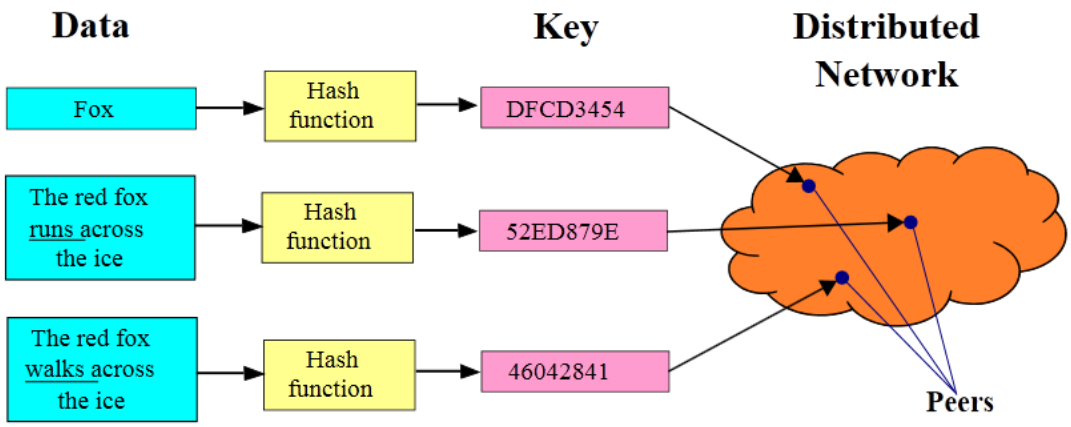


Figure 3: DHT network

http://en.wikipedia.org/wiki/File:DHT_en.svg

2.7.3 Properties

DHT are intended for large peer-to-peer networks therefore some of its properties are obvious.

Decentralization – the nodes form the system without any central coordination. (18)

Fault tolerance – the system should be reliable even with nodes continuously joining, leaving, and failing. (18)

Scalability – the system should function efficiently even with thousands or millions of nodes. (18)

The set of keys is often marked as K. Key $k \in K <=> \exists s: h(s) = k$. This key is then marked as k_s.

Function $\delta: K^2 \rightarrow N$ is distance between two keys. This function is unrelated to geographical distance or latency. Each item and peer in DHT network has its unique $id \in K$. Item s is assigned to the peer its id is closest to k_s according to the function δ.

Peer reconnecting or disconnecting influences just its small neighbourhood.

2.7.4 Routing Tables

To be able to route requests and to find all items, each peer is either holder of the given key or has record in the routing table which points out to the peer which is closer to the key.

Unlike unstructured peer-to-peer networks, item in the DHT networks is always found. Searching stops after finite number of steps.

Efficient searching requires minimization of the number of steps needed to find an item or key. Not every node needs information about entire network. The goal is to minimalize the size of routing tables and number of hops.

2.7.5 Naïve Implementation

2.7.5.1 One-Hop Lookup

Each routing table contains all peers. Therefore it is also called full information lookup. Number of hops is straight zero. Size of the routing tables is n. (9) (19) An example of such network imagination can be seen in the Figure 4. New peers have complications of joining the network because they have to get information about the entire network.

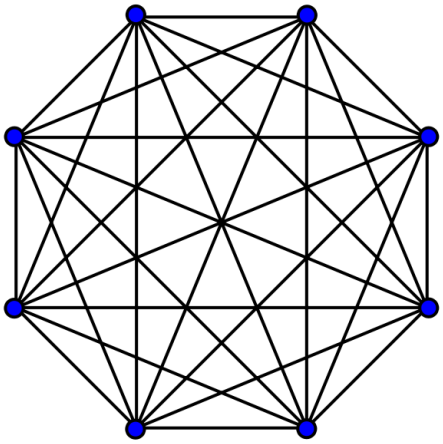


Figure 4: One hop lookup DHT example

http://nl.wikipedia.org/wiki/Bestand:Complete_graph_K8.svg

2.7.5.2 N-Hop Lookup

Each routing table contains just one record – to next peer in the row. It is also called minimal information lookup. Number of hops can be n, what is number of all peers

in the network. It is very easy for new peers to join the network – they need to contact only two peers. Example is an oriented cycle as shown in the Figure 5.

Figure 5: N-hop lookup example

http://en.wikipedia.org/wiki/File:DC8.png

2.7.6 Common Implementations

Average number of hops for network of size n is $\log n$. Common size of routing tables in these networks is also $\log n$.

Differences between these implementations are mainly in a realization of the function δ. Managing of the routing tables is important factor either. These two things influence also the routing algorithm. We will introduce some common used implementations of DHT networks in the next chapter. We will discuss differences among them. Main focus will be delivered to the Kademlia which is the implementation we chose to work with.

2.7.7 Content Addressable Network

Space of keys is a virtual multidimensional Cartesian coordinate space. Coordinates in the d-dimensional space represent keys. One can easily see that this virtual space is independent on the location and network connection. Entire system is made as d-torus so it is fully closed.

Function δ is defined as a distance in the space. Entire space is divided into zones – each zone is owned by one peer. Each peer has pointers to its neighbours' zones.

2.7.7.1 Joining

To join the network, peer p randomly generates its coordinates. It sends join message to the bootstrap peer (any known peer by the incomer). It finds the peer r which zone

contains initialize coordinates. Zone is split between peers p and r. Neighbour peers are informed about this change. The process is showed in the Figure 6.

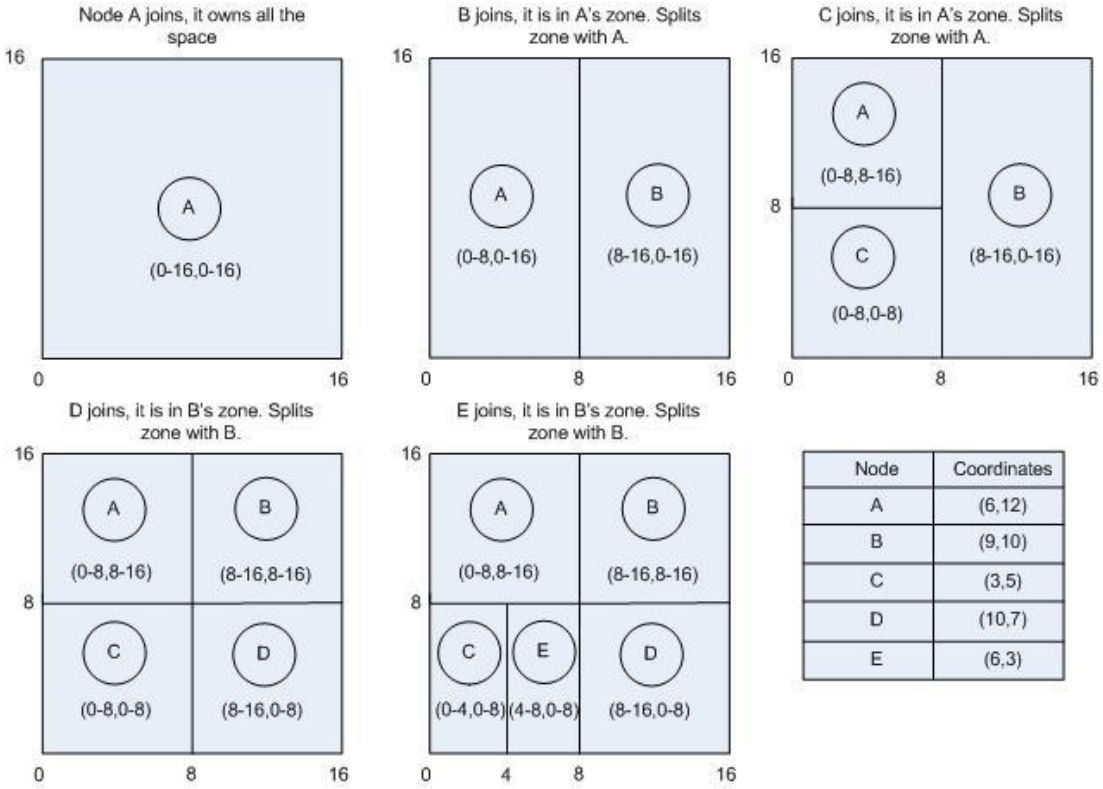


Figure 6: Five nodes joining a CAN network

2.7.7.2 Leaving

Leaving is a reverse process to joining the network. The peer r which can take over the zone is found among neighbours. Zones can be merged or peer r temporarily owns two zones.

2.7.7.3 Lookup

Searching is realized by using a greedy algorithm. Distance between searched key and peer has to decrease. Example is showed in the Figure 7. Searching begins at peer A.

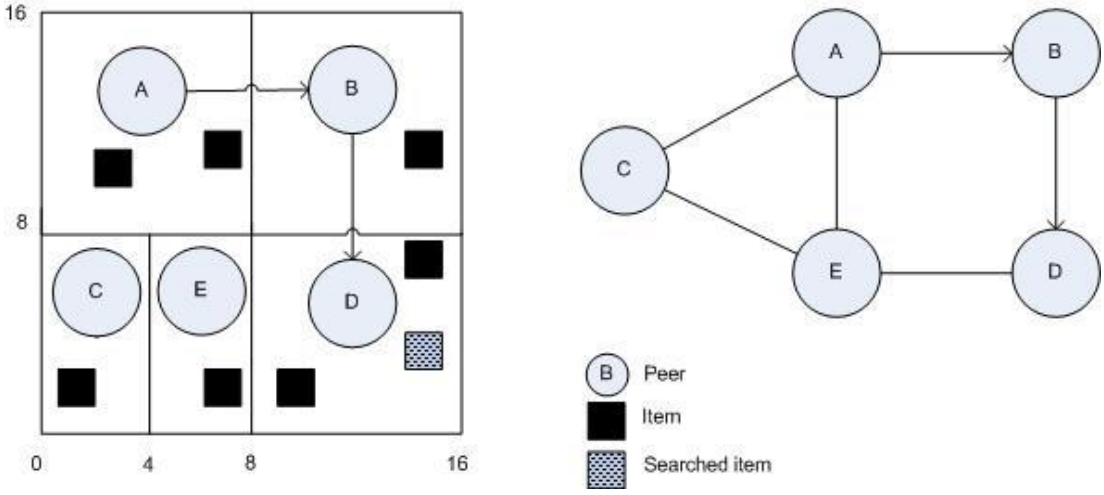


Figure 7: Searching in CAN. Peer *A* is searching for marked item.

2.7.7.4 Replication and Fault Tolerance

Failure detection has two main ways in CAN. When peer tries to communicate with neighbour and gets no answer, it takes over neighbour's zone. Second way is sending regular Hello message among neighbours. If this message does not come, every neighbour knows it. The neighbours will negotiate who will take over the zone – mostly a peer with the smallest zone. (9) (20) (21)

Replication is also achieved in two ways. The first one is to use n hash functions so data are sent to n nodes. Lookup is also made by all n hash functions. The peer received n answers and can choose one of them.

Second way is to create multiple instances of the coordinate space. Each instance is called "reality". If a peer is disconnected in one reality, data could be found in another reality.

2.7.7.5 Properties

Number of all information peer has to hold is $O(d)$. Average number of hops is $O(d * n^{\frac{1}{d}})$. This is not typical value for DHT networks but for $d = (\log_2 n)/2$ we get typical value.

2.7.8 Chord

Chord assumes circular space of size N. Keys are m-bit numbers on this circumference. (21) Function δ is a distance between two keys on circumference clockwise.

Chord defines function $Succ(u)$ as the first peer which is clockwise on the circumference space. There is a function $Pred(u)$ which is defined as the first peer which is counter-clockwise. Peers therefore form a double linked list.

Item with key k is stored on $Succ(k)$. In other words, key k is stored on the first peer that follows clockwise on circumference. Situation is showed in the Figure 8.

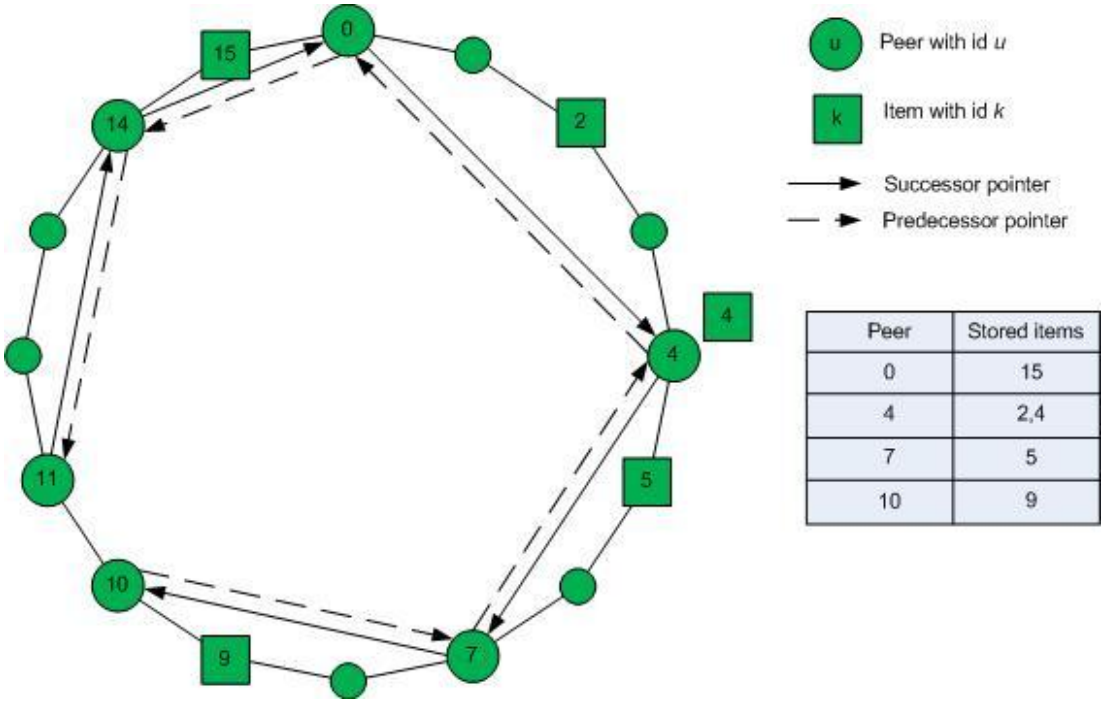


Figure 8: Chord network with 5 stored items

Each peer has a list of pointers called fingers. Peer holds $M = \log_2 N$ fingers. Set of fingers of peer u is marked F_u, and is defined as follows: $F_u = \left\{\left(u, Succ\left(u + 2^{i-1}\right)\right)\right\}, 1 \leq i \leq M$, calculations are modulo N. Example of such fingers is in the Figure 9.

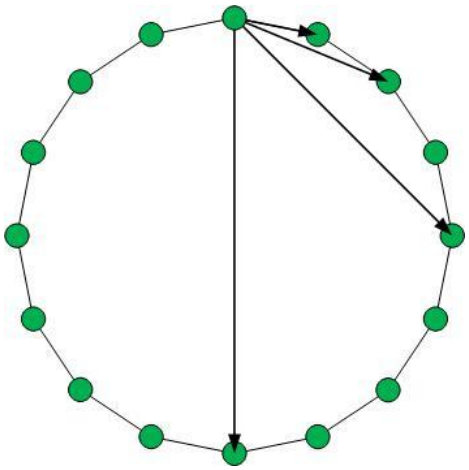


Figure 9: Fingers of one peer in Chord

2.7.8.1 Joining

As for most of DHT systems, first contact is made by bootstrap. The incoming peer initializes search query for itself. This allows it to find out successor and predecessor. Peer plugs itself among its successor and predecessor. Everything else is done by the stabilizing algorithm.

2.7.8.2 Leaving

Leaving is similar to joining. Peer just informs successor and predecessor. All its items are moved to successor and the rest is done by the stabilizing algorithm.

2.7.8.3 Lookup

Routing table of each peer has size M. Record i covers the area of size 2^{i-1}. The example of fingers and size of areas are in the Figure 10.

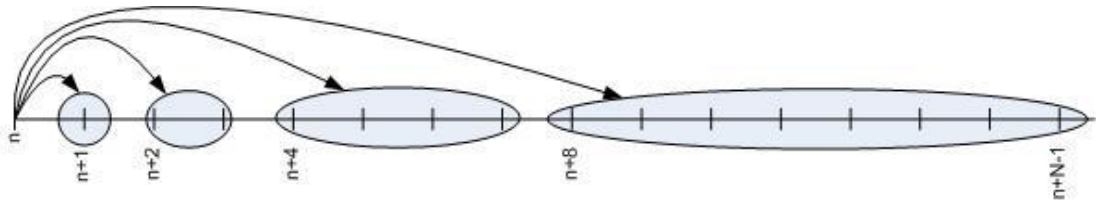


Figure 10: Routing table and fingers in peer *n*

When searching for item with key k, we know it is on the peer $Succ(k)$, therefore we are searching for this peer. Peer u asks for advice a peer which is the closest preceding finger of k.

Because each step shrinks the area to search at least by half, the number of hops for key lookup is $O(\log n)$. To number of all routing information which is needed is $O(\log n)$. (22) (21) (9)

2.7.8.4 Replication and Fault Tolerance

To avoid loss of information caused by peer failure, each peer maintains the list of size $M = \log_2 N$ of following peers on the circumference. Each item on peer u is replicated to this list. Failure of $\log_2(N) + 1$ consequent peers is needed to lose an item.

This list has also another usage – if peer u finds out that its successor is down, it can easily replace it.

2.7.9 Pastry

Keys are m-bit numbers of base 2^b. For $b = 4$ it can be seen as sequence of hexadecimal numbers. Function $share(k_1, k_2)$ gives us a length of common prefix of keys k_1 and k_2. For example:

$$share(0x123ABC, 0x123111) = 3$$

$$share(0x123ABC, 0x121111) = 2$$

$$share(0x123ABC, 0x111111) = 1$$

Each peer has a list of $\frac{L}{2}$ successors and $\frac{L}{2}$ predecessors where L is network parameter. It is known as leaf set. Peer has also a list of M nodes which are near to it by another metrics such as latency or network hops. This list is not used for routing, it is used for maintenance. The last kind of data is routing table which contains $\lceil \log_{2^b}(N) \rceil$ rows by $(2^b - 1)$ records. Row number is the length of common prefix given by function $share$. Numbers b, M and L are input parameters of the network. Values of $b = 4$, $M = 2^b$ and $L = 2^b$ represent typical values. (23)

Typical routing table can be seen in the Figure 11. Character x in each cell represents routing information to peer with given prefix. White cells represent prefixes which are identical with actual peer.

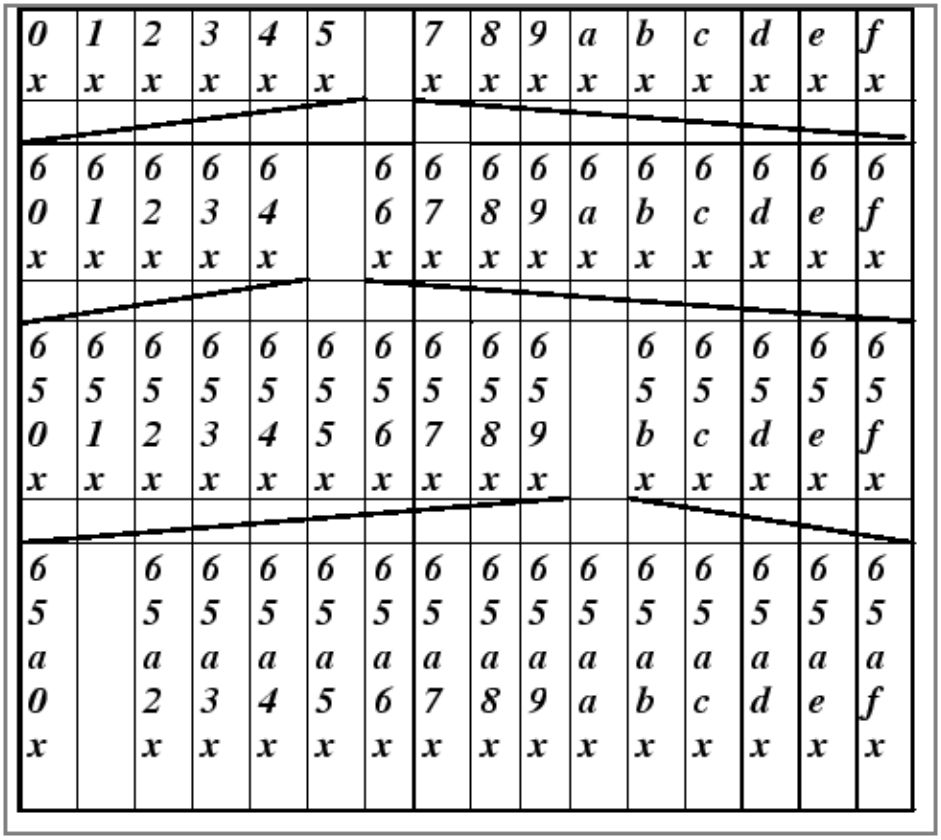


Figure 11: Pastry routing table of peer with id=*0x65a1*

http://www.usenix.org/event/osdi02/tech/full_papers/castro/castro_html/

2.7.9.1 Joining

Let peer X want to join the network. At first, it contacts any peer A in the network. Peer A routes a message $join(X)$ to the peer P which is the closest peer to the peer X. Peer X gets leaf set from the peer P. Peer X gets i^{th} row of its routing table from i^{th} peer on route from A to P. Finally, X informs other peers about its existence in the network. (21) (9)

2.7.9.2 Leaving

If a peer is leaving, it puts over its data to another peer. Routing tables are repaired automatically.

2.7.9.3 Lookup

Let peer X search item with key k. If k is in the X's leaf set, routing is directly according record in the leaf set.

If k is not in the leaf set, the routing table is searched for a peer which common prefix with k is bigger than between k and current peer. If no such peer is found, routing is forward to the peer from the leaf set, which numerical value is closest to the k. (21) (23)

2.7.9.4 Replication and Fault Tolerance

Items are replicated to the k closest peers from the leaf set. This allows not only fault tolerance but caching either. Caching are replicas of data on another peers than owner. These replicas allow searching to be faster and avoid traffic congestions on the peer with popular data.

2.7.10 Kademlia

We will discuss Kademlia network in the next chapter because it is the network we chose for our experiments. Using of separate chapter allows us to be more precise and describing.

We will compare basic properties of all mentioned DHT peer-to-peer networks in the end of the Kademlia chapter.

3 Kademlia

Kademlia is very similar to Pastry. Keys are also m-bit numbers. The main difference is in the function δ, which is defined as follows: $\delta(x, y) = x \oplus y$ (XOR).

XOR (Exclusive Or) was chosen because it shares some properties with the geometric distance formula. (24) Specifically:

- The distance between a peer and itself is zero.
- It is symmetric: $x \oplus y = y \oplus x$.
- The triangle inequality holds.

These properties provide most of the important behaviours of measuring a real distance with a significantly lower amount of computation to calculate it. (24)

Kademlia divides the space of identifiers exactly like Pastry. The difference is in the point of view. Peers' keys are represented as leafs in a binary tree where peer's position is determined by the shortest prefix of its id. It is an analogy to the $share$ function. In Kademlia, the $share(k_1, k_2)$ function is defined as the longest common binary prefix of keys k_1 and k_2. For example:

$$share(0000, 0001) = 3$$

$$share(0000, 0011) = 2$$

$$share(0000, 0111) = 1$$

Imagine it as a binary tree. In the Figure 12 we assume that keys are 4-bit numbers. Therefore entire network can have $2^4 = 16$ peers at max. You can easily see the length of common prefix.

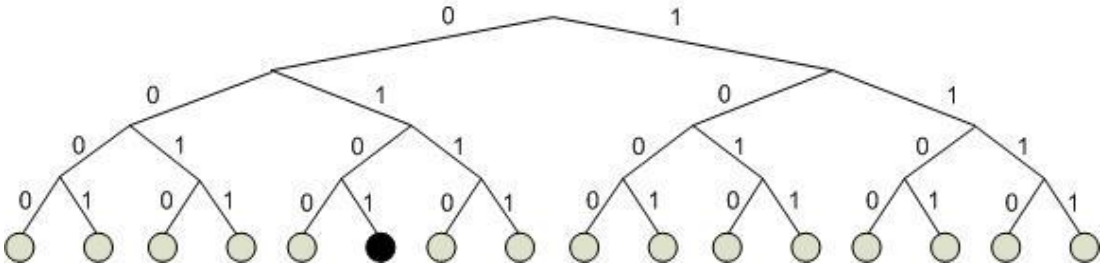


Figure 12: Kademlia network as a binary tree

3.1 Routing table

To be able to route messages quickly, Kademlia uses a system of k-buckets. It holds a list for each bit of the peer key – together there are m lists. Elements in the i^{th} list of a peer k have $share(i, element) = i - 1$. In other words, they differ in the i^{th} bit. Such example is in the Figure 13.

Black peer is our current peer. The lists' members are from dashed areas. Each list contains not only one pointer to the given area but k pointers if possible. Bigger size of lists allows us faster routing and better fault tolerance.

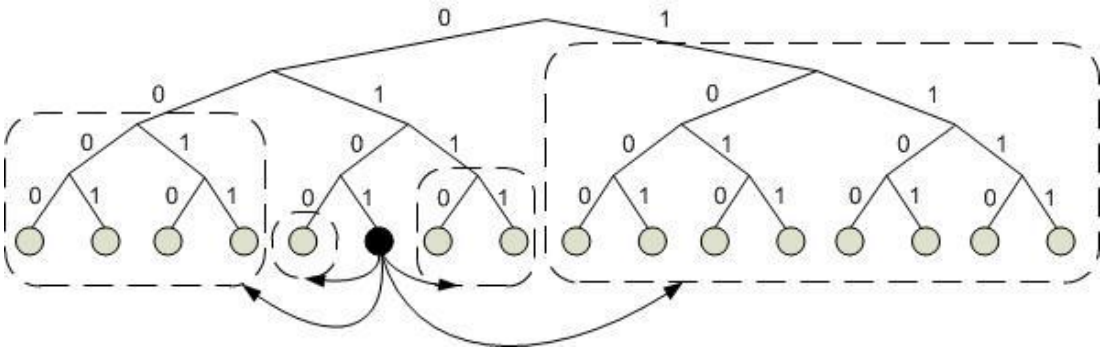


Figure 13: Kademlia k-buckets principle

It is easy to fill the first list because area contains 1/2 of all peers in the network. The next list can use only 1/4 of all peers. You can see that areas are bigger if the common prefix is lower. That means areas are smaller when prefix is longer. Size of area is decreasing exponentially. Therefore for higher i there will be just a few peers in the list. Take the peer with $id = 4$ for example. It is the only peer in the area. Therefore the list will not be full.

Peers in each k-bucket are ordered ascending by the time of adding. It means older nodes are first in the k-bucket. The reason for this is that peers which are online longer have higher chance to keep online for the next hour. (25) This dependency is showed in the Figure 14 which is also from (25).

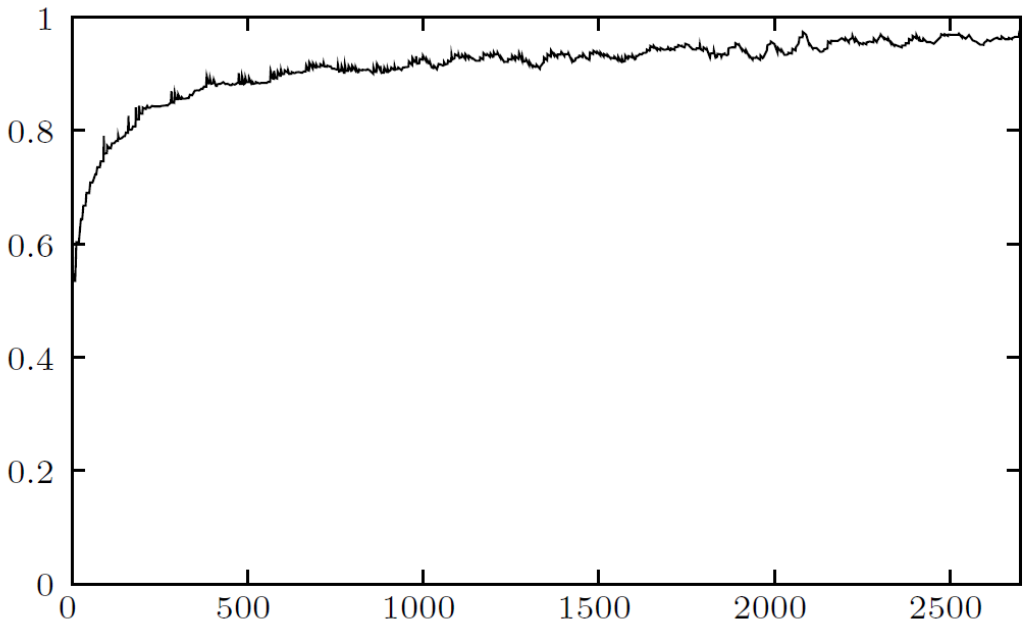


Figure 14: Probability of remaining online another hour as a function of uptime. The x axis represents minutes. The y axis shows the fraction of nodes that stayed online at least x minutes that also stayed online at least x + 60 minutes.

***Figure and its title are from* (25).**

When a peer $\boldsymbol{u}$ is searching and peer $\boldsymbol{v}$ is processing this request, than $\boldsymbol{v}$ checks if a $\boldsymbol{k}$-bucket is full. If it is not full, $\boldsymbol{v}$ adds $\boldsymbol{u}$ to the end of given $\boldsymbol{k}$-bucket. If it is full, the least recently seen peer is pinged. If pinged peer will not answer, $\boldsymbol{u}$ is put to the end of the list. Otherwise $\boldsymbol{u}$ is not used at all.

3.2 Joining

Joining is very easy. The new peer needs to know only one peer from the network. After that it searches for random peers and that will fill its $\boldsymbol{k}$-buckets.

3.3 Leaving

Peer just disconnects and the network will handle it.

3.4 Lookup

Routing algorithm works recursively. Assume that the key $\boldsymbol{l}$ is searched. Peer $\boldsymbol{u}$ finds $\boldsymbol{\alpha}$ peers which are closest to $\boldsymbol{l}$ by the meaning of the function $\boldsymbol{\delta}$ among its k-buckets. It asks them about the closest peer to the key $\boldsymbol{l}$. Peer $\boldsymbol{u}$ will send this message to the peers which are part of the response. This continues until $\boldsymbol{k}$ closest peers are found.

Parameter α is often called degree of parallelism because messages are send asynchronous. This improves the latency of a lookup.

3.5 Replication and Fault Tolerance

Each item is replicated to the k closest peers. These peers periodically checks whether any of them has disconnected. If so, item is replicated to the next peer. It means that at least k peers have to disconnect almost simultaneously to lose an item.

3.6 DHT Networks Comparison

We will compare basic properties of DHT networks at the end. Focus is on the main properties like average number of hops, size of routing information and costs of peer joining/leaving.

	CAN	Chord	Pastry	Kademlia
Average hops per routing	$\boldsymbol{O(d * N^{\frac{1}{d}})}$	$\boldsymbol{O(\log N)}$	$\boldsymbol{O(\log_B N)}$	$\boldsymbol{O(\log N)}$
Routing information	$\boldsymbol{2 * d}$	$\boldsymbol{\log N}$	$\boldsymbol{B * \log_B N + B}$	$\boldsymbol{k * \log N}$
Join/leave costs	$\boldsymbol{2 * d}$	$\boldsymbol{(\log N)^2}$	$\boldsymbol{\log_B N}$	-

Table 1: DHT networks comparison

Data is from (9)

Variables:

- $N =$ Size of the network
- $B = 2^b$
- $b =$ Pastry network parameter, see chapter 2.7.9
- $d =$ CAN network parameter, see chapter 2.7.7
- $k =$ Kademlia network parameter, see chapter 3

4 Trust Management

4.1 Weaknesses of P2P Networks

We discussed P2P networks advantages in the Chapter 2.2 and in this chapter we will discuss weaknesses of these systems. P2P networks are often used for file sharing therefore we will consider mainly this type of service.

The main weakness is absolute trust in each other. P2P networks are based on equality and trust therefore there are no special mechanisms how to control malicious peers. Peers can be bad in several ways. They can change sent data. They also can influence all the traffic going through it including routing messages.

Very interesting is the case of MediaDefender, the anti-pirate organization. This company decided to make downloading files inefficient. They put a lot of corrupted files or just scrambled files in the network. When user downloaded it, data were useless but it cost him time and other resources. Almost 25% of all downloaded files in some networks are fake files – they contain just corrupted data. (26)

Similar problems are so called free-riders – peers which just download data but do not share. That makes network imbalanced and in the worst case can cause its failure.

Viruses are not a difference. Because P2P networks do not have any security, everyone can connect and spread viruses. Imagine that you want to download an mp3 file and get a computer virus. Maybe you can block the sender, but other peers in the network will not.

These are the main challenges in the security of P2P networks. To ensure trust in such an environment is very difficult. We will discuss the problem more in the next chapters.

4.2 Trust Management

In information system and information technology, trust management is an abstract system that processes symbolic representations of social trust, usually to aid automated decision-making process. (27)

Trust management in P2P networks is a way how peers challenge threats mentioned in the previous chapter. It creates a network of trust. Peers know whom to trust, who can send them correct files. This influences routing a lot.

There are two main ways how to build trust management – policy based trust management and reputation based management.

4.3 Policy Based Trust Management

Policy based trust is often used in access control decisions. (28) Therefore the result is a boolean decision, access is granted or not. Peers need to specify precise access conditions what implies existence of well-defined semantics. Therefore decision should be dependent on attributes which are granted by certification authorities. Certification authorities are fully responsible for their decisions. This makes it very strong and useful in situations where are costly consequences – take electronic business for an example.

4.4 Reputation Based Trust Management

Reputation is the opinion (more technically, a social evaluation) of the group of entities toward another entity on a certain criterion. (29) We think this statement describes it very well. Whole reputation based trust management is based on others opinions. P2P network is like a group of people who try to make business together. They make a few trades and then share their experience because there are hustlers there. On the other hand, with a good system, they do not need to share experience – this is considered to be harder.

Reputation management is the process of tracking an entity's actions and other entities' opinions about those actions. (30) It means that there has to be a system of trading experience among peers. Such a system makes typical P2P network less chaotic.

Because of the nature of P2P networks, reputation based trust is the most common one. Our work is also built on reputation.

There are a lot of reputation based trust managements. We cannot describe them all therefore we chose three of them which we will shortly describe in the next chapters.

4.5 Eigentrust

Eigentrust (31) is the oldest algorithm of chosen ones. Its principle is very easy – to calculate trust on past interactions with given peer. It requires keeping history with all peers which actual peer u communicated with. If there is no history for given peer v, u tries to obtain it from other peers in the network. It actually asks them for their opinion. The result in normalized to be between 0 and 1.

In the first version of Eigentrust peers calculated theirs trust value by themselves. It is not very secure therefore distributed Eigentrust was presented. It brings new method of calculating trust values by distribute calculation. There is only one global value for each peer in the system. This implies that all peers trust the same way. All peers are calculating it together.

Big advantage of Eigentrust is its mathematical background. Algorithm properties are proofed what means it should work the same way even under not simulated circumstances. Other algorithms are based mostly on heuristics and presentation of good simulation results.

We see the problem of Eigentrust in data security – changing of locally stored trust values. The authors mentioned Secure Eigentrust in a short way by using so called score managers and their choosing by random.

Problem is also in the lack of simulations. Simulations were made only on small number of peers – a few dozens. (31) The network was Gnutella.

4.6 Xrep

Xrep (32) is an interesting protocol because it takes in consideration not even the peer reputation but also reputation of the file. This makes it possible to solve the problems with sharing of infected files by good peers (e.g. by virus infection).

Protocol does not calculate global trust value as Eigentrust. It asks random peers about their local opinion on file and peer which is providing it. These values are weighted and the decision is made. This has the advantage that peer cannot change its local trust value because it does not exist from the principle. There is only opinion on it by the peers it worked with.

Problem is with the lack of simulations, specifically any particular implementation because authors just analysed network traffic which was recorded in the form of logs – the source of the logs is not clear. Therefore authors did not present results of their protocol they just theoretically explained its advantages.

4.7 Trust Vectors

This algorithm (33) does not have any name but to be able to reference to it, we called it "Trust vectors". Algorithm uses only reputation of the peer.

Each peer holds a vector of trust evaluations for each peer it has come to contact. This vector represents history of the communication. Because the peer remembers history, it can better evaluate trustworthiness of other peers. The length of the vector is limited what gives algorithm a way how to forget. This is very important for reputation redemption because not all bad peers are still bad – e.g. good peer can be infected by virus and temporarily sends infected files.

Algorithm introduces two kinds of vectors – one for trust values and one for recommendation trust values. Reason is that the peer u can trust recommendations on other peers from peer v but does not need to trust its files and vice versa. This is very useful in a situation of coordinated attack where are two groups of a collective of malicious peers – ones which spread viruses and ones which share proper files and help increasing trust of the first group.

Authors presented results of the simulation with 1000 peers but only with 1000 unique items. We think that 1000 unique files is not too much but it is still much more than in the previous cases.

5 Simulation Environments

Simulation environment is very important part of every work on P2P networks. We cannot write a program for P2P sharing and run it on a few hundred or even on a few thousand computers. That is usually beyond researchers' possibilities. This is where simulators come into the play.

Simulator is simplified environment where we can observe behaviour of a lot of objects under certain circumstances. Each simulator has its advantages and disadvantages. It is very important to choose the right one.

Our searching can be divided into the three steps. Firstly, we searched for any possible simulators. Then we chose six best candidates according to their references and compared them. The third phase was about detail examination of two best environments. We describe the process in this chapter.

5.1 Most Common Used Simulators

This chapter is about the second phase of searching – brief description of six simulators according their reputation on internet.

5.1.1 P2PSim

Homepage: http://pdos.csail.mit.edu/p2psim/index.html

Language: C++

Current state: Last change was in the year 2005, still in alpha version.

5.1.2 FreePastry

Homepage: http://www.freepastry.org/FreePastry/

Language: Java

Current state: Last change 13.3.2009.

Protocols: Pastry

Advantages: Comprehensive tutorial.

Disadvantages: Only Pastry protocol.

5.1.3 PlanetSim

Homepage: http://projects-deim.urv.cat/trac/planetsim/

Language: Java

Current state: Last change 17.8.2009.

Protocols: Pastry, Chord, unstructured network

Advantages: Multilayer architecture. Nice graphical network topology output.

Disadvantages: Almost no statistics out of simulator; no tutorial.

5.1.4 Overlay Weaver

Homepage: http://overlayweaver.sourceforge.net/

Language: Java

Current state: Last change 11.5.2010.

Protocols: Pastry, Chord, Kademlia, Koorde, Tapestry.

Advantages: Multiple protocols support. Distributed computing.

Disadvantages: Almost no documentation. Simulator is in the form "take or leave" – very hard to extend it. No good statistics and no system of statistics gathering.

5.1.5 PeerSim

Homepage: http://peersim.sourceforge.net/

Language: Java

Current state: Last change 29.9.2009.

Protocols: Pastry, Chord, Kademlia, BitTorrent and more.

Advantages: High scalability. Cycle-based and event-based models. Strong statistical output system which is easily extensible.

Disadvantages: Does not count with details of the network layer.

5.1.6 OverSim

Homepage: http://www.oversim.org/

Language: C++

Current state: Last change 26.5.2010.

Protocols: Pastry, Chord, Kademlia and more.

Advantages: High scalability. Multitier architecture. Counts with entire network complexity.

Disadvantages: Complexity.

5.2 Best Simulators

We chose two best simulators of all analysed in the previous chapter. It is PeerSim and Oversim. These two simulators are quite different in a way of making simulations. Their comparison is made in this chapter. For process of choice, please see the Chapter 6.3.

5.3 Oversim

It is one of the most complex simulators or frameworks. We can say that Oversim is the only framework whereas others are just simulators. Oversim is not just a one program; it is a complex set of programs and IDE. Simulator itself is based on environment Omnetpp, specifically on its part INET.

5.3.1 IDE

Integrated development environment is very important here because framework uses its own language called NED. IDE is based on the Eclipse framework with extended possibilities of graph presentation and editing of language NED.

Eclipse foundation guaranties portability to different platforms. It also makes it extensible. This environment helps us with building and running everything from the IDE, we do not need external script such as makefile.

5.3.2 Installation

Installation is surprisingly simple for such complex framework with own simulation language. The only problem was with the version 4.1 which did not work therefore we had to use older version, specifically 4.0.

5.3.3 Architecture

Architecture is showed in the Figure 15. It is multitier architecture with many overlay networks which implement different communication protocols. DHT services are on the top of these communication protocols. DHT is therefore not bound to lower overlay networks. We can also build more tiers on top of the DHT one such as peer-to-peer name service. This approach is quite nice because of transparency of overlay layers.

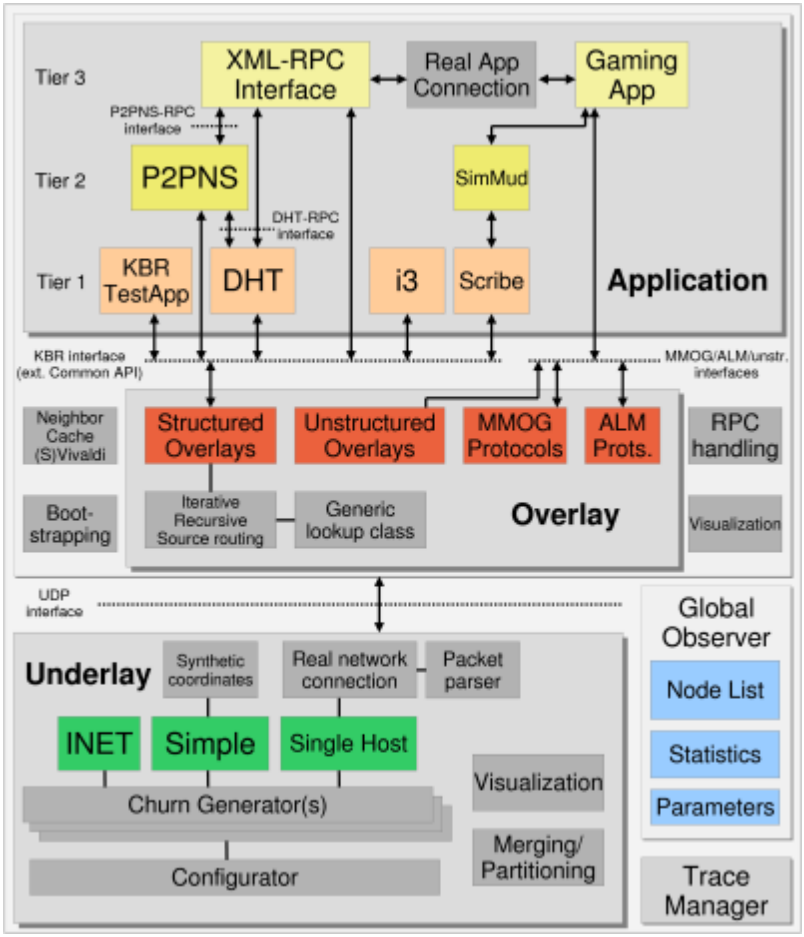


Figure 15: Oversim architecture

http://www.oversim.org/wiki/OverSimFeatures

5.3.4 Advantages

Complexity is the biggest advantage of this framework. It allows us to simulate peer failures, different types of overhead in the network, undelivered messages and other stuffs.

Many protocols like Kademlia, Chord, Pastry and others are already implemented with prepared DHT layer.

Multitier architecture is advantage too, because it allows us easy testing with different transport protocols.

Configuration of the simulation is in the configuration file what makes it easy to run different simulations without need to rebuild entire project.

Oversim has a reach system of statistics which are easily extensible. This is important because of result comparison.

Oversim is almost the only project which is currently in a development. Authors are very willing to help community. Answers to our questions were replied on the next day.

5.3.5 Disadvantages

Complexity of this simulator is also its very big disadvantage. Oversim can probably do much more than you really want. Overhead by managing, configuring and implementation of your own protocol could be huge.

Personal disadvantage is programming language C++ which we are familiar with but we prefer higher languages such as Java. Oversim suffers from some C++ problems like loss of type control and extensive usage of dynamic_cast<> what is a slow operation and it leads to opacity in the code.

Next disadvantage is still alpha version of DHT implementation and some transport protocols.

We found out documentation not so good. Explanation of framework functions is quite confusing. Documentation is just being created by a few people. There has been almost no change in the documentation for the past few years. This influences understanding of framework functions – we could not understand it even after couple of weeks. Communication is a nice example. Peers can communicate with messages, RPC calls and UDP packets but there is no specification which communication should be used in which case – at least, we could not find it.

5.4 PeerSim

PeerSim has different philosophy than Oversim. It is very simple simulator that simulates only the most necessary things, rest is left to the user.

It was started under European projects BISON and DELIS. (34)

5.4.1 IDE

There is no IDE for PeerSim. Everything is made by command line. PeerSim is built as library – you have to include it to your code. Than you can run it. There is just one makefile in the default downloaded package. It runs everything.

5.4.2 Architecture

Architecture or design is not unfortunately part of the documentation. There is nothing like class diagram there, just doxy documentation and tutorial. In spite of it, you can find out a very nice usage of design patterns Observer and Decorator.

Usage of the design patterns Decorator gives user very strong tool to implement almost any type of simulation. User can define network initialization and even set properties for every peer in the network.

Different properties are set in the layers. User can define as many layers as he wants. First layer can connect some peers in the network topology. Second layer can set latency among peers. Third layer can be used to simulate possible network failures. Peers' numbering and specialization can be done in the fourth layer. This is just brief example what can be done be this approach – many more examples can be found in (34).

PeerSim has two ways of simulation – cycle based and event based.

Cycle based simulation is very simple and light-weighted one. Peers are processed in the cycles. Each cycle begins with peer one and ends with the last peer according its number. You can imagine it as a token passing. Every peer can do its stuffs when it has a token. This is of course very easy to implement and very fast. Problem is that this is not a kind of behaviour we expect in our work. We assume P2P network with messages and many concurrent events. It is also the reason authors added event based simulation into the PeerSim.

Event based simulation is based on calendar of actions. Each action has its time and place – in this case time is a discrete time in simulation units and place is a peer. This kind of simulation is very close to our model because each message can be mapped to event. Even things like timeouts are easily done as events. Therefore we think that this is the better kind of simulation in PeerSim. This approach has its disadvantages either. It is more source consuming than cycle based one.

5.4.3 Advantages

Big advantage is simplicity of the simulator. It does not simulate everything in the network. It is just based on delivering events (in the case of event based simulation). Simulator itself does not count with latency, peers failure and so on. This is entirely

left in the hands of a programmer. This makes simulator very fast and easy to use. It simulates only things that you really want.

Lack of precision is not as bad as it may look. Latency and even peer failure can be implemented easily – in fact, it has already been implemented by a community. If you want to implement latency, just add one layer into the settings and it is done.

Community makes PeerSim very valuable. There are dozens of implementations of different protocols. (34) All you need is to use it.

We like a powerful configuration file. Almost everything can be set by this way. Each layer we mentioned above can be set in the configuration file.

Makefile is advantage either. It makes it easy to run repeatedly without need of clicking in the IDE.

5.4.4 Disadvantages

Simplicity of the simulator can be a two-edged sword. Simulation does not cover everything and therefore it is just up to programmer what will be covered. It requires skills and experience with simulations. One forgotten property can spell disaster. Take forgotten latency for an example. If we would not set latency and it remained zero then all messages would be delivered instantly and simulation would be worthless.

Lack of qualitative documentation on event based simulation makes it harder to use. This is problem especially in the beginning but later does not matter.

Smaller disadvantage is that PeerSim cannot be run parallel on more computers. On the other hand, we avoid problems and mistakes with synchronization.

System of statistics is quite advanced but it is still just in the form of logs. There is no IDE with graphs like in the Oversim. Therefore each log has to be processed after the simulation is over.

6 Choices

We present the most important decisions we had to make. It is particularly choice of DHT protocol, choice of algorithm and choice of simulator. Each choice is about to be explained in this chapter. These choices influence types of malicious nodes in the network therefore types of malicious nodes are described in this chapter either.

6.1 DHT Protocol

Goal of this thesis is to implement reputation based management into the protocol of structured P2P network and to measure its efficiency. That means we have to implement DHT protocol over P2P network. Next, we have to implement secure algorithm over this DHT protocol. Finally, we should try to implement this algorithm into the protocol core. Comparison of all these phases is important and required.

Structured P2P networks are used in very fluctuating environment with a lot of leaving and joining peers. Therefore our chosen protocol should handle this very well. This is our first requirement.

As mentioned above, we need to implement secure algorithm into the protocol, therefore protocol should allow it at least indirectly. This is our second requirement.

The third requirement is to have function implementation of the protocol in our chosen simulator. This is not a real need but it would save us time with implementing it from the scratch.

Most of the DHT protocols fulfil the first requirement but Kademlia seems to be the best. It is designed for really dynamic environment. Costs for joining and leaving the network are minimal. Please see the Chapter 3.6, Table 1 especially, for comparison.

The second requirement is more problematic because protocols like Chord cannot be easily changed without breaking its functionality. We prefer protocols like Pastry and Kademlia with its k-buckets.

The third requirement is not such a problem because both best simulators support a large scale of DHT protocols which are already implemented. See the Chapter 5.2 for closer details.

Finally, we chose the Kademlia protocol. It fulfils all requirements and therefore is the best for our purposes.

6.2 Algorithm

Secure algorithm is very important in our thesis. We have to find one which is very well described. Than we need to implement it. Algorithm evaluation follows after implementation. Here we test whether the algorithm is suitable for Kademlia network or not. If it is not working, new algorithm search is needed. After successful implementation of the algorithm, we need to find a way how to implement it to the protocol core of the Kademlia network. Algorithm has to be able to be used in all phases.

We had several options how to search for an algorithm – to use any of the existing algorithms or to figure out a new one. Existing algorithms can be split to several groups by their usage. They are generally designed for unstructured, structured or both kinds of networks. Therefore the first requirement is to have algorithm for structured P2P networks, especially for Kademlia which is our chosen one.

Algorithm should be very well described and tested, if possible. The more details provided about implementation, the better. The best scenario is to have function implementation. This is our second requirement.

The third requirement is to have an easily extensible algorithm. It is almost certain that we will need to change the algorithm or to extend it in some way. Chosen algorithm should be prepared for this.

The fourth requirement is so called nice-to-have. We would like to have some kind of theoretical estimation of its efficiency. This is very important to compare measured results with the theoretical ones. It allows us to have boundaries which we can converge to. On the other hand, it can stop us from further implementation when we see that maximal theoretical efficiency has been reached.

The first requirement is the easiest one. There are a lot of algorithms which fulfil at least compatibility with structured DHT networks.

Most of the published algorithms do not provide their implementation – they just provide description. Such examples are (31) (32) (33). This implies that we have to implement it by ourselves.

The third requirement is difficult to meet. A lot of existing algorithms are not easily extensible. Eigentrust calculates a global value for each peer but you cannot add new parameter without jeopardizing entire calculation. We think that trust vectors algorithm fulfils this requirement best.

The last requirement is the one with the least success. It is very difficult to provide the best theoretical efficiency. Most of the algorithms are based on heuristics and therefore does not provide estimates. Even Eigentrust, the algorithm with very good mathematical background, does not provide it.

Lack of the algorithms that meet our requirements forced us to write down our own algorithm. It is based on trust vectors, which is the most extensible algorithm for us. Trust vectors provides multiple levels of security, which number can be easily increased – please see Chapter 4.7 for details.

6.3 Simulator

Simulator is a thing which is used only at the end of a whole process. There is no such thing as the only right simulator. We can use multiple simulators but result depends on our requirements. Simulator will get program with our algorithm and then it just simulates.

Our chosen simulator should support multiple DHT networks, especially Kademlia which we chose as the most suitable protocol.

Quality of implementation should be at the first place therefore we prefer simulators which have strong background and are commonly used. This is the second requirement.

The third requirement is a personal one – that simulator is written in and controlled by a programming language we are familiar with. This will save us time and sources. It would help to improve quality of implementation of our algorithm.

The fourth requirement is to have simulator which is fast and which can be run in a parallel way. Parallel run is very important because we expect a lot of simulations which would take hundreds or even thousands of hours on one computer.

Our first requirement showed to be quite restrictive and just two best simulators fulfil it. The first is Oversim and the second one is PeerSim. Very surprising is that they are quite different. For more information about these simulators, read the Chapter 5.2 please.

The second requirement about strong background and community is easily done by both simulators – each one has dozens of citations and plenty of different protocol implementations written by community. (34) (35)

Oversim is written in C++ and requires learning of its own language. On the other hand, PeerSim is written in Java and uses simple property configuration file to parameterize simulation. We like Java and property file more therefore we consider PeerSim better at this point.

The fourth requirement is about parallel run. Oversim can use multiple computer cores for simulation. PeerSim cannot do it what could be bad, but configuration by property file removes this defect. In spite of that we cannot run one simulation in parallel, we can run multiple simulations on multiple computers. This is very important factor which makes PeerSim usable.

All requirements we have are fulfilled by the only simulator and that is PeerSim. We will use this simulator with its Kademlia implementation in our work. Please refer to the Chapter 5.4 for more information about PeerSim.

6.4 Malicious Peers

Types of malicious peers and their attacks depend on the network type. We are using Kademlia network therefore we will describe only attacks which are possible in this network.

There are two main kinds of attacks – individual or cooperation attacks. Individual attack is an attack of one node with any means possible. Cooperation attack is an attack with more nodes which help each other. We will describe basic individual attacks and then we will present their cooperation on meaningful attacks.

6.4.1 Individual attacks

1. Peer U puts hash of an infected file in all words it can find. There are plenty of mappings (word; infected file hash). Mapping of each possible word to given file increases the chance that this file is chosen to download.

2. Similar attack to 1 but after several successful attacks peer disconnects and afterwards reconnects in the network to avoid bad reputation. This is so called Sybil attack.

3. Similar attack to 1 but after sending of malicious file sends more good files to increase reputation. This attack is called Camouflage attack.

4. Malicious file hash planting or modification of entire mapping table of (word; file hash). This attack can be also called "lucky attack" because peer has to be chosen as storage for some interesting word such as porn, Matrix or some new film. Chosen as storage is meant that its distance is the closest in the network and therefore word's mapping is there. If the network generates IDs by random, this attack cannot be planned.

5. Proxy attack. Responses from higher peers (in the search hierarchy) are changed – owner of the word or pointer to the file hash can be falsified.

6. Proxy distributed denial of service (DDOS) attack. This is specific modification of the attack number 5 from above. Peer U sends all requests to peer A what causes A's congestion. In other words – peer U claims that any file hash is in peer A. The same can be done with word hashes. Our measurements show that surprising huge number of messages is going through every peer. All these messages can be changed in the case of Proxy DDOS attack.

6.4.2 Cooperation attacks

Each individual attack can be done by a group of peers and that way causes more damage. That makes five mass attacks. The most dangerous one is proxy DDOS which in the case of bigger group of malicious peers can be devastating for target peer.

There are other possible combinations of peers in a malicious group but only some of them have sense – they will increase attack strength. We chose combinations which are considered to be dangerous:

7. Combination of attack Sybil attack with Lucky attack. Peer C can change mapping of word – file hash to profit of its cooperatives.

8. Similar to 7, but instead of 4 is used the proxy attack.

9. There are two groups of peers – one infects honest peers and one increases reputation of the peers from the first group. This attack is response to the basic reputation management mechanism.

7 Algorithm

This chapter describes all defensive algorithms we have used. It is sorted by the increasing complexity of the algorithms. We provide deep description of each strategy and its name which is, as in all our work, the same as name in code and other parts of our thesis.

We implement algorithm using the strategy pattern which allows us easily change its implementation. The word “strategy” in the names of all algorithms is partially taken from this pattern.

Each strategy implements methods which help with downloading files and their routing. How exactly it is done, is up to the given strategy. They have only one thing in common – how to determine that the file is infected. We leave this up to user or the application. We assume that we know the file maliciousness in our simulation – i.e. when the file is downloaded, we can say whether it is malicious or not. We do not count on knowing this during any other part of the searching.

Basic search process is:

1. Random word is chosen to be searched - can be influenced by the rule 80-20 (80% of all searches is for 20% of the content.)
2. Random peer is chosen as the start peer. This peer will search for chosen word.
3. List of file hashes which contains given word is obtained by searching in the Kademlia.
4. Start peer decides which file and from which peer to download.

7.1 DefensiveStrategyRandom

Every simulation needs some kind of reference values to be able to compare results. This strategy is our default implementation of such behaviour.

As the name says, it picks random files which are downloaded. Everything is done randomly using pseudorandom generator. This strategy is our reference which we compare other results to.

7.2 DefensiveStrategyForbidAll

This strategy is based on simple assumption – all we need is to forbid all peers which have sent at least one malicious file. There are plenty of other peers therefore our searching abilities should not be compromised.

The strategy checks the file maliciousness and then makes a decision. If the file is malicious then the peer is added to the ban list. Peer cannot be removed from this list, it can be only added.

When the file is about to be downloaded, peer checks its ban list and removes all files that are in the peers from this list.

This strategy is very simple. It is the very first thing that most people think about. It has some advantages and some disadvantages which are presented in the chapter Results.

We chose this strategy to have comparison with more advance strategies.

7.3 DefensiveStrategyTrustVectorsA

This strategy is our first usable attempt to create more sophisticated defence. It is based on the Trust vectors algorithm – please see Chapter 4.7 for detail information.

It has a memory what can be imagine as vector of certain length. If a downloaded file is malicious than a value one is added to the first position in the vector otherwise zero is added to the first position. Memory has the ability to forget therefore after every download the vector is shifted to the right by one position (it is divided by two). The length of the vector is the length of the memory. It is settable attribute but values about eight are good enough. Memory allows peers to regain a good reputation in the case of accidentally sent of malicious file. It takes eight (in our case) good files to send to regain full trust. The trust is represented as a score. Full trust corresponds to a zero score. The higher the score, the bigger the distrust is.

Downloading a file is done by choosing a peer according to its score. Only the peers with the lowest score are chosen and one of them is randomly picked. We will download a file from this peer.

The thing we have added and which is not in the Trust vectors is a negative based approach. It means that each new peer we have not communicated before is

considered to be bad and it has to prove its cleanness. It is done by adding a small discriminating value to its score. This way no peer has zero score at the beginning. Most of the current algorithms have positive based approach what means that each new peer has a zero score because it has not done anything yet. Then the score is changed whether in positive or negative way.

Negative based approach has a disadvantage that it could shrink peer's search radius or at least it could prolong a search. It could look so because the peers we have met before are the ones we download from the most. This is not entirely true because if that peers do not have our file, we can download it from other peers with slightly bigger score. This guaranties us that we will always find our file, if it exists.

This approach makes some kind of connections or highways in the network. These highways are only on logical layer so far. They are one-way what means that we can trust a peer but it does not have to trust us. There is no such thing as a global trust value; each peer has its own opinion about everything.

Typical discriminating value is 1. This allows all peers to clean themselves with just one good file. If a malicious peer wants to clean itself after sending a malicious file, it has to send at least eight clean files. This is a very big difference which can be even higher if needed.

Advantage of the negative based approach is clear – it very well works against free-riders and reconnecting malicious peers. Reconnection is not an option because it will not clear the score.

7.4 DefensiveStrategyTrustVectorsB

We have a potentially successful strategy against many malicious peers but we do not have anything that could handle Proxy DDOS attack. This strategy is the answer for it. Strategy TrustVectorsB enhances strategy TrustVectorsA therefore it has the same

Our second strategy behaves the same way as the first one but it adds a new layer for handling the DDOS attack. This layer is similar to the first one. It is a memory which holds score for DDOS attacks. If a DDOS attack has been done, the peer is added here and its DDOS score increases. This approach has also a small discriminating value for new peers.

The second layer gives us an ability to distinguish between attacks and possibly ban a peer for DDOS-ing but still using its files or vice versa.

Complicated part is distinguishing between a DDOS attack and a normal routing.

Let peer A search for a word W which hash $h(W)$ lies in peer B. Mapping for the word W in peer B has been inserted by peer C. Let peer T be the target for DDOS attack. Situation is shown in the Figure 16. Let assume that A is a clean peer – there are four possibilities what can happen. We discuss them all. We assume, for now, that peer T can say that some search query is a DDOS attack and other peers would believe it.

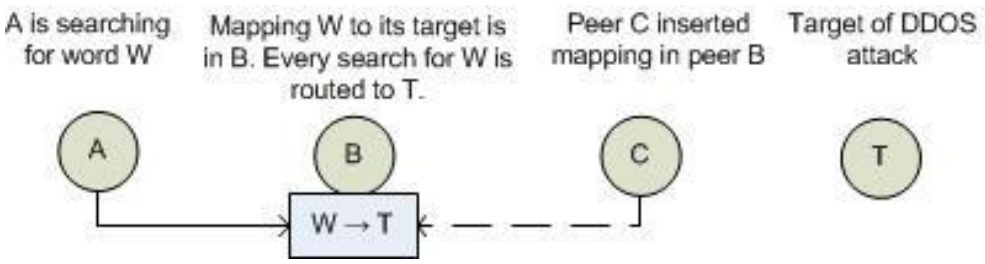


Figure 16: DDOS attack principle in the Kademlia network.

a) B is clean, C is malicious

This scenario shows typical problem of structured P2P networks – B cannot be responsible for mappings which are there. Peer C can easily put mappings for all possible words to peer T in other peers. Mapping modification has to be allowed that network should provide its function – please see the Chapter 3 for more information.

Peer A does not know who put bad mapping there – it could be B or C. C could be the only one malicious peer but can potentially infect all mappings and B does not have any potential defence. We think this is the biggest problem.

Solution is to keep track who inserted which mapping in any peer.

b) B is malicious, C is clean

This is similar case as above but in reverse. A still does not know who is responsible for the DDOS attack. It can be B or another peer C which is mostly unknown. We can keep track who inserted what in any peer. This is problematic because B should blame innocent peer. We still do not have proof that B is telling us the truth.

c) B is malicious, C is malicious either

Very simple scenario which is not more powerful than any of the cases before. If we can solve both scenarios above then we will solve this scenario either.

d) B is clean, C is clean either

This is the best scenario where everything is ok and there is nothing to decide.

We have solved this by shifting the responsibility for mappings to the peer C which inserted it. The problem is still that B can blame C. This is solved by adding a digital signs. There are systems based on RSA which can be used. (32)

This allows us to use digital sign for verifying who inserted what and where. B cannot falsify digital sign of C therefore C is in safe. Peer B is safe in case a) either because malicious peer C can be identified by its digital sign.

There is still one problem – target peer T can lie. Even this is solved by digital signs and logic in the DHT networks. Peer T has to prove that someone is making a DDOS attack by showing a signed mapping. This mapping should be signed by peer C and pointing to peer T. If $T \neq C$ than T is telling a truth because C could not insert any other mappings than for words which are located in C. This means that mappings pointing to the peer C should be signed only by the peer C. Any other combination is a sign of an attack.

We implemented this kind of signed mappings and logic into our system and algorithm against DDOS attacks was done.

7.5 DefensiveStrategyTrustVectorsD

The last strategy we have tried solves the problem with possible network shrinking. It is based on strategy DefensiveStrategyTrustVectorsB which downloads files from peers with the best (mostly zero) score. Strategy D adds a chance that other peers will be included.

New threshold is introduced – peers with score lower or equal than this threshold will be included with a certain chance. Threshold and probability can be configured to achieve best results for different types of networks.

High chance means that the network would not shrink due to negative based approach. Lower chance helps to create subnetworks of trust which we discussed in the previous chapter.

7.6 Algorithm in the Protocol Core

The goal of this thesis is to try to implement trust management into the routing protocol. We did this by changing the order of peers in the k-buckets. We did preserve algorithms based on trust vectors and enhanced its power into the routing mechanism.

Sorting of peers in the k-buckets in this case is firstly done by peers' score and secondly by time. This preserves time ordering in the case of the score absence. We wanted to test whether this mechanism would help to create clean subnetworks better or not. These subnetworks are kind of transitive trust where we do not expect distributed calculations. They are not created intentionally by cooperation, they are just used on the principle – "I believe because you believe but I always check it". If the subnetwork is created, all good peers in it will benefit from it.

8 Our Settings

We will describe all our settings and decisions in this chapter. These settings are mostly about our simulations. We will refer to our work simply with one word as simulation or as program.

As described in the chapter above, we decided to use Kademlia network implementation in the PeerSim simulator. We downloaded this simulation from the project homepage. The simulation is under the GPL license. Therefore our work is also under the GPL license. This simulation had to be strongly customized to match our expectations.

8.1 Program Separation

First change we have done is separation of network generation from simulation itself. This is important from two points of view. Firstly, it spares time because we do not need to generate network for every simulation we are running. All we need is one network with given parameters for a bunch of simulations. Secondly, it is logically better – we can change network generation without worrying of code change in the simulation. This allows us to try more types of networks – even types we did not see in the past.

There are two programs now – Kademlia simulation and Network generator. You can find source codes of both in the attached CD. Time spare is about several hours per simulation in big networks. We measured time of network generation and it is approximately three hours for network of 5000 peers and 50 000 items. We have done several thousand simulations so time saving is huge.

8.2 Routing

Routing is simple according to the Kademlia but we write a short description to keep reader in the picture. At first, we describe how new files are added to the network.

8.2.1 Adding a New File

Assume that there is a file with name A which can be split into the words $S_1..S_n$ – mostly according to the whitespaces. The peer P wants to add this file therefore it makes hash $h(A)$ and P finds the closest peer R to $h(A)$. Now the peer P needs to propagate the file name that others peers could search with words only and do not need to know exact hashes.

Peer P hashes each word S and gets hash $h(S)$. P finds the closest neighbour Q as in the case of searching for file hash. The hash $h(S)$ and mapping to R are stored in the Q. In the case of redundancy, the file and words hashes are stored in more peers according to the given algorithm – in our case in the neighbours.

8.2.2 Searching

Assume that peer P wants to download a file with name A. The name is divided into the words $S_1..S_n$.

1. Searching is performed for each word. Response for a word search is a list of file hashes which contain given word in their name.
2. Basic set operations are used to obtain a set of file hashes which fulfils our search query. These operations are mostly joining and intersection.
3. Peer chooses one hash from obtained set.
4. The hash is known therefore peer P finds its owner.
5. Information about owner are gathered, peer P can verify it from responses from more peers.
6. Chosen file is downloaded.
7. Maliciousness of the file is determined – whether the file is malicious or not.

Searched values are often cached – so called proxy. It is important especially for points 1 and 4. Proxying influences a security of the network therefore we have to cover this threat in our work.

8.3 Turbulence

Turbulence should be a part of every P2P network simulation. Turbulence means that peers are dynamically connecting and disconnecting during the simulation. This brings some changes into the network behaviour. We use Kademlia network which is very effective with dealing with this behaviour. We will describe joining and leaving the network in the next two subchapters. Ratio of joined and left peers is discussed in the Chapter Parameters.

8.3.1 Peer Leaving

Peer leaving is done in two layers – the network layer and the application layer which is Kademlia. Peer has to be deactivated in both layers otherwise will be in inconsistent state.

Deactivation is done by setting a flag in the peer and removing it from the network. Copy of peer is done to be used later on. This copy is done without routing information and peer ID. Copying preserves maliciousness of the peer.

8.3.2 Peer Joining

Our simulation does not care about generating of new words or files after the initial phase. This affects the way new peers are generated. When the peer is leaving, its copy is made and held in the memory. It means that peer has already some files assigned. This memory holds all such peers. The memory holds only Kademlia layer peer information.

When the network decides to generate and join a new peer, new network peer is made and one of the peers in the memory is chosen as the Kademlia layer. This peer is removed from the memory.

When the memory is empty, no new peer is added in the network. If it was added, it would have empty file list and serve just for routing information resent and as a new requests generator without file sharing – so called free-rider. We can simulate free-riders other way therefore this is not allowed.

Peer keeps its role after activation because deactivation preserves maliciousness of the peer.

8.4 Implemented Attacks

This chapter presents all implemented attacks and their characteristics. These attacks were tested in simulations and results are presented in the chapter Results. Their names are according to the ones used in program therefore you can easily look at results or code and still be in the picture.

8.4.1 MaliciousNodeInfectAllItems

This is the most primitive attack of all. Malicious peer M infects all files which are downloaded. Success of this attack depends on peers defence. If any reasonable defence is implemented, this attack is without a chance.

On the other hand – without defence it is very cost effective because it does nothing but infecting.

8.4.2 MaliciousNodeInfectNthItem

Malicious peer M sends both good and malicious files. Good files are sent to regain positive reputation and attack the basic principle of trust management – trust evaluation. We implemented parameter N which tells us how often are malicious files sent. This is a variant of Camouflage attack from the Chapter 6.4.1.

This attack can be quite successful. Good defence is a big memory – a history of sent and received files from other peers. Another good defence is a total prohibition of this malicious peer.

8.4.3 MaliciousNodeInfectSomeItemsReconnect

This is a better form of MaliciousNodeInfectAllItems attack. Malicious peer M tries to send several infected files (settable parameter) and then it disconnects and reconnects with a different ID. This is a variant of Sybil attack from the Chapter 6.4.1.

This attack is very hard to break because peer with new ID is considered to be a new member in the network and therefore it has no bad history.

Disadvantage of this attack is the fact that often changes of ID makes the peer really quickly isolated because it is not put in the routing tables of a lot of other peers. This fact decreases the efficiency of infecting.

Another disadvantage is that reconnecting may take a while when the peer is not in the network and cannot infect other peers.

8.4.4 MaliciousNodeDDosWordResponses

This is a variant of Proxy DDOS attack from the Chapter 6.4.1. All routing messages that go through the malicious peers are changed to point to the target peer. Our

results show that the traffic increase at target is really high and can cause the peer congestion.

8.5 Parameters

Our simulation has a lot of configuration parameters. Each parameter has its reason and is important in some way. We present main parameters in this chapter.

We had to set some parameters in advance because of their huge number – testing all their combinations would be impossible in reasonable time and with possible sources. Values of these parameters are also presented in this chapter.

All parameters are presented with the names used in the configuration file so reader can easily stay in the picture.

networkSize

Parameter networkSize is the number of all peers in the network. This includes both malicious and clean peers.

We chose values 500 and 5000 as the testing values. Network of size 500 is a smaller one and is used to test basic premises. If any dependence appears, test with the bigger network is done.

words

Parameter words is the number of all words used in the simulation. It represents all words which are contained in all shared file names.

Number of words in the network is approximately the same therefore we decided to use only one value. This value is set to 1000 words. We tried some experiments with different sizes and the results were very similar.

items

Parameter items is the number of all items in all peers in the network. It is obvious that it depends on the number of peers therefore we set it to 10*networkSize.

seed

Parameter seed is the random generator seed. There is only one seed for entire simulation. More seeds are used to achieve balanced results which are not dependent on one lucky coincidence.

redundancy

Parameter redundancy is a replication factor for files and words. It determines how many copies of each word and file are there in the network. We set this factor to five what is common value in the Kademlia network.

numberOfSearches

Parameter numberOfSearches is the number of search queries which are about to be run. Each such a search query is for a file that contains particular word.

This parameter influences the length of simulation the most therefore we have used 50 thousand searches as the quick overview. This value was used mostly with the networks of size five hundred. We chose 500 thousand searches as better simulation and 5 million searches per simulation in specific experiments where the results should have been precise.

numberOfMaliciousNodes

Parameter numberOfMaliciousNodes is the number of malicious peers in the simulation. This number is valid in the beginning because malicious peers can leave and join the network and their number therefore fluctuates. Values of this parameter are chosen as percentage values of the network size. This implies that the number of clean peers is networkSize minus numberOfMaliciousNodes – in some cases it can be less than number of malicious peers.

maliciousNodeType

Parameter maliciousNodeType is the type of malicious peers. We did not use defence based on a collective effort therefore we did not try to test with different malicious groups. We only used one big malicious collective with peers of one type. These peers did not cooperate among them but MaliciousNodeDDosWordResponses which concentrates effort of all peers to one target. The list of possible attacks is in the Chapter 8.4.

defensiveStrategyType

Parameter defensiveStrategyType is the type of the defence which is used. Type of the defence is used in all clean peers. This means that we do not test network where are peers with different defensive strategies. The list of defences is in the Chapter 7.

kBucketItemComparator

Parameter kBucketItemComparator is the type of the comparator used in k-buckets. Standard is the time comparator which is used in Kademlia. We test this comparator with other comparator which is implemented in our defences. More about k-bucket comparators is in the Chapter 7.6.

numberOfExperiments

Parameter numberOfExperiments is the number of experiments which are done by PeerSim. This is originally the PeerSim parameter which is there to balance simulation results and clean it from lucky coincidence. We set is to one and we simulated differences with the seed parameter.

simulationTime

Parameter simulationTime is the total time of simulation in simulation units. This parameter influences how long does the simulation run. Simulation is forcibly ended after the time is reached. We set it to 36 million of simulation units.

transportLayer

Parameter transportLayer is a type of the transport layer. We have two main types of the transport layer. First one is UniformRandomTransport which randomly generates latency between two peers every time the latency is needed. Latency values are limited between two borders – please see next parameter explanation.

The second type of the transport layer is E2ETransport which stands for End-to-end transport. This transport layer allows us to define latencies between every two peers in the simulation. We can even choose whether the latency is symmetrical or not. Values of this type of latency are firmly set and do not change during the simulation.

minDelay

Parameter minDelay is bound to parameter transportLayer. It is the minimal possible value of the latency between any two peers in the entire simulation.

maxDelay

Parameter maxDelay is bound to parameter transportLayer. It is the maximal possible value of the latency between any two peers in the entire simulation.

turbulenceTimes

Parameter turbulenceTimes is the number of changes in the network. There is turbulence in the network as mentioned in the chapter 8.3. Number of leaving and joining peers is regulated by this parameter. It is the number of all leaves and joins together. We mostly used value of 10% of all peers. It can be interpreted like that the network experience 10% fluctuation. This does not include fluctuation caused by the malicious peers with their strategy – mostly with "reconnect" strategy described in the chapter 8.4.3.

idle

Parameter idle is the fraction of turbulences which causes nothing. We set this parameter to zero because otherwise it is counterproductive.

add

Parameter add is the fraction of turbulences which causes a new peer joining. The system of joining is described in the chapter 8.3.2. We set this parameter to 0.5.

remove

Parameter remove is the fraction of turbulences which causes a peer leaving from the network. The system of leaving is described in the chapter 8.3.1. We commonly set this parameter to 0.5.

The sum of idle, add and remove parameters has to be exactly one. We mostly used combination of 0.0-0.5-0.5 to have balanced network.

observerStep

Parameter observerStep is the step of observer in simulation units. Observer is a logging module of the simulation. It is responsible for almost all statistics. It does not influence results in any way. We used values of 1 million simulation units and 100 thousand of simulation units in case of closer look.

trustInNewNodesInPercents

Parameter trustInNewNodesInPercents is important only when using with strategy DefensiveStrategyTrustVectorsD. Its purpose is described in the Chapter 7.5. Values of this parameter are in the percentages.

popularWords

Parameter popularWords is a fraction of words which are considered to be popular. These words are downloaded more often than others because traffic in the real networks in not linearly spread but it has its specifics. There is a rule 80-20. 80% of all searches are for 20% of the content. (36) This rule is very important and we had to implement this behaviour – it is controlled by parameters popularWords and popularSearches.

We mostly set this parameter to 0.2 but we tried it with value 0.1 as the simulation in networks for sharing mainly new content.

popularSearches

Parameter popularSearches is bound to the parameter popularWords. It is the fraction of all searches which are only for popular words. We set it mostly to 0.8 to satisfy 80-20 rule. We also tried value 0.9 as the simulation in networks for sharing mainly new content.

9 Results

We present all our experiment results in this chapter. Results are split by the defensive strategy that was used. All strategies are compared and the results are finally summed up.

Our simulations had a huge number of parameters therefore we will not mark each simulation with its full name but only with necessary parameters and its id which allow exact simulation parameters to be found in the attached CD.

9.1 DefensiveStrategyRandom

This is the basic strategy where peers download files without any security assistance.

Graph of number of infected file downloads during the time is shown in the Figure 17. We can see that it is linearly spread across the time. That is expected behaviour because there can be no improvement without no security and learning from the past. This holds even under the simplest attack where malicious peers infect all files.

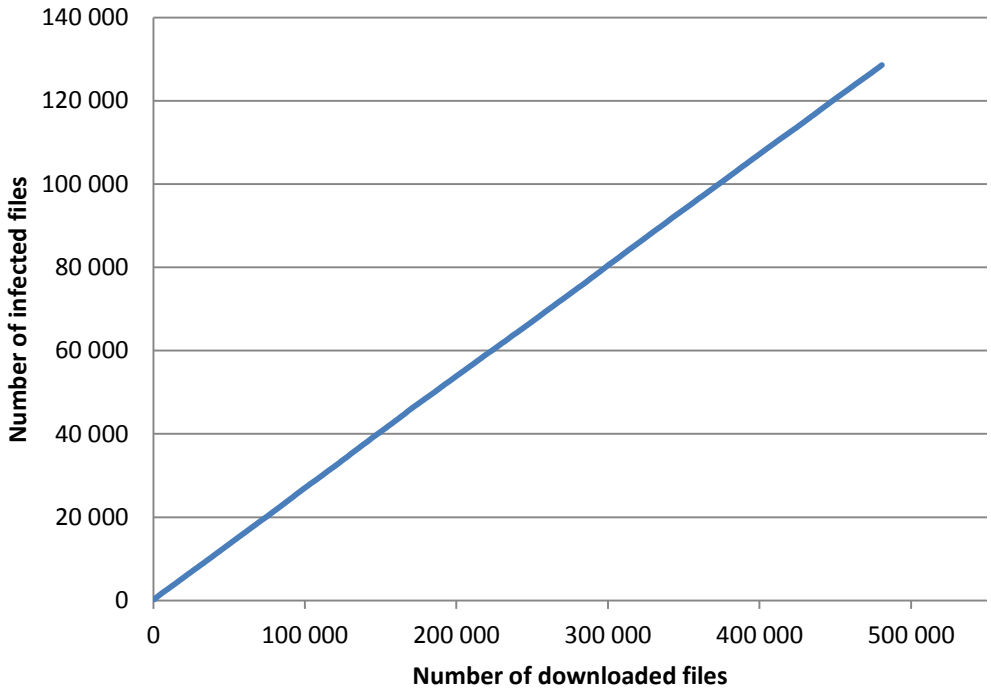


Figure 17: Random strategy under attack InfectAll. Network size is 500 peers where 5 are malicious.

We will use this strategy to point out some interesting observations such as influence of different seed of the pseudorandom generator. This biggest difference we could find is shown in the Figure 18. We can see that it is still linear dependency – the only difference is the gradient. This is the reason why we will not post graphs with different seeds – the results are always very similar.

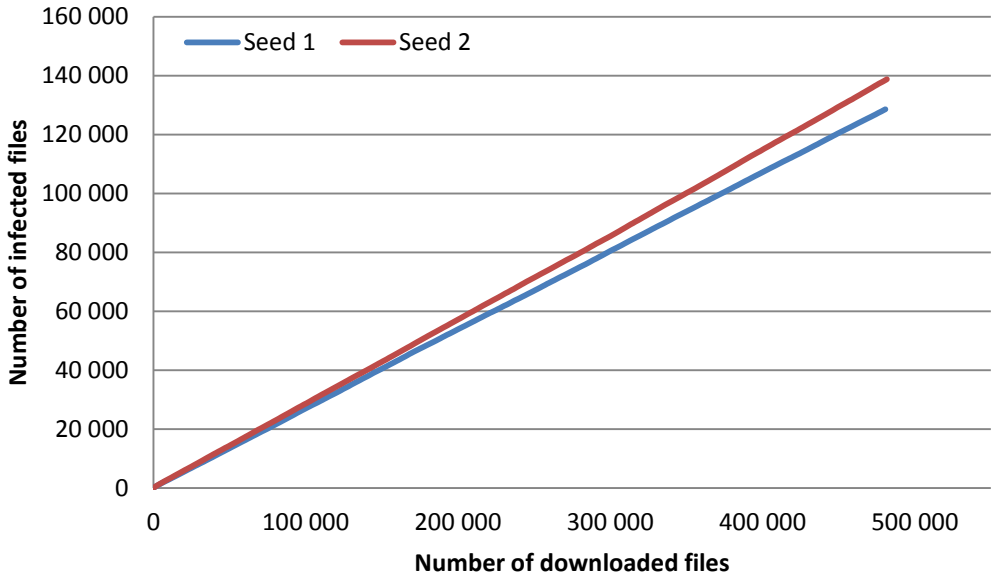


Figure 18: Comparison of how different seeds influence results. Network has size 500 peers with 5 malicious ones. Defensive strategy is random and attack is InfectAll.

Graph in the Figure 19 shows us a dependency of number of downloaded infected files on different number of malicious peers. The network size was 500 peers where M were malicious.

We can see that increase of number of infected files is up to 150 malicious peers what is 30% of the network size. This is the point where it starts to fall down. This is because of too big saturation of malicious peers and a small number of clean peers. There are 450 malicious peers and only 50 clean ones in the extreme case. This could not have good results in term of efficiency – we can see that number of infected files is almost the same for 150 and 200 malicious peers. This is the reason why we tested mostly with 1%, 5% and 10% of malicious peers in the network.

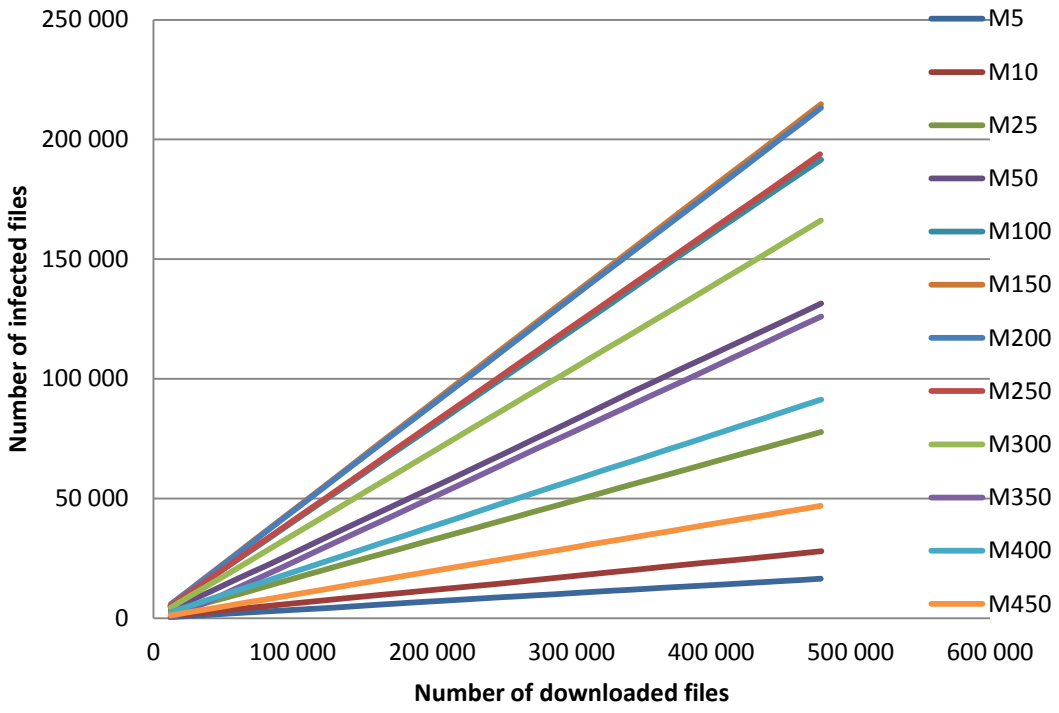


Figure 19: Dependence of a number of downloaded infected files on number of malicious peers in the network of size 500. Defensive strategy Random, attack InfectAll. Different number of malicious peers.

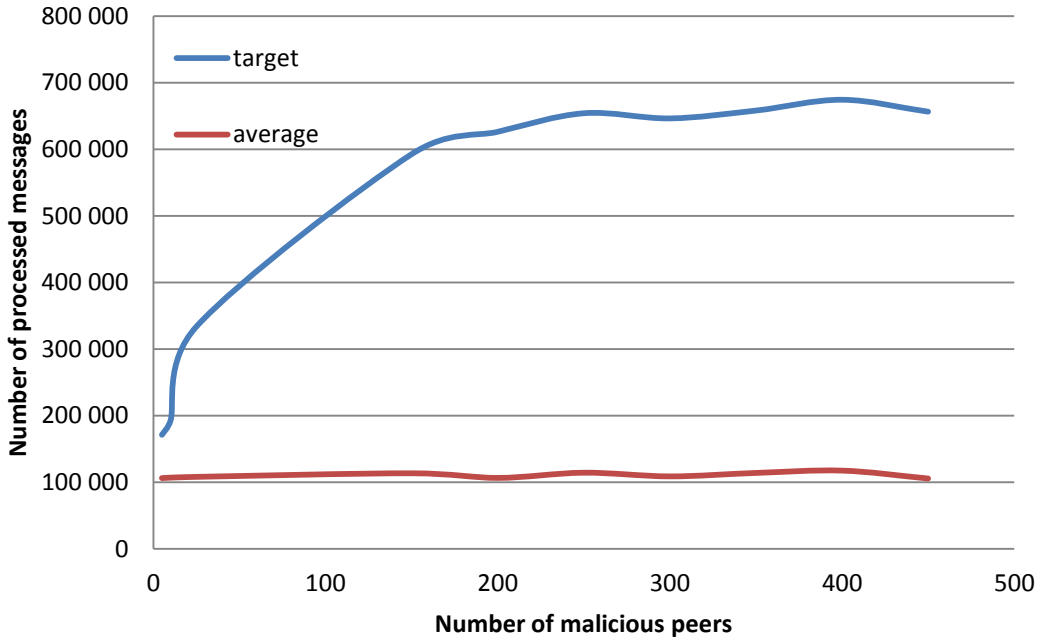


Figure 20: DDOS attack and Random strategy. Network size si 500. Comparison of average message count and number of messages processed by the target peer.

Figure 20 shows us a successful proxy DDOS attack. Every malicious peer is sending all traffic to the target peer what causes its congestion. The graph shows the number of processed messages – average for the network and the ones processed by the target peer. We can see that the increase is really big. Maximum starts at about 200 malicious peers. The attack is successful because the target processed almost seven times more messages than is the average.

9.2 DefensiveStrategyForbidAll

This strategy represents basic defence. Its results should be better than in the case of Random strategy. Comparison of these two strategies under InfectAll attack is in the Figure 21. We can see that ForbidAll is a very good defence for this type of attack.

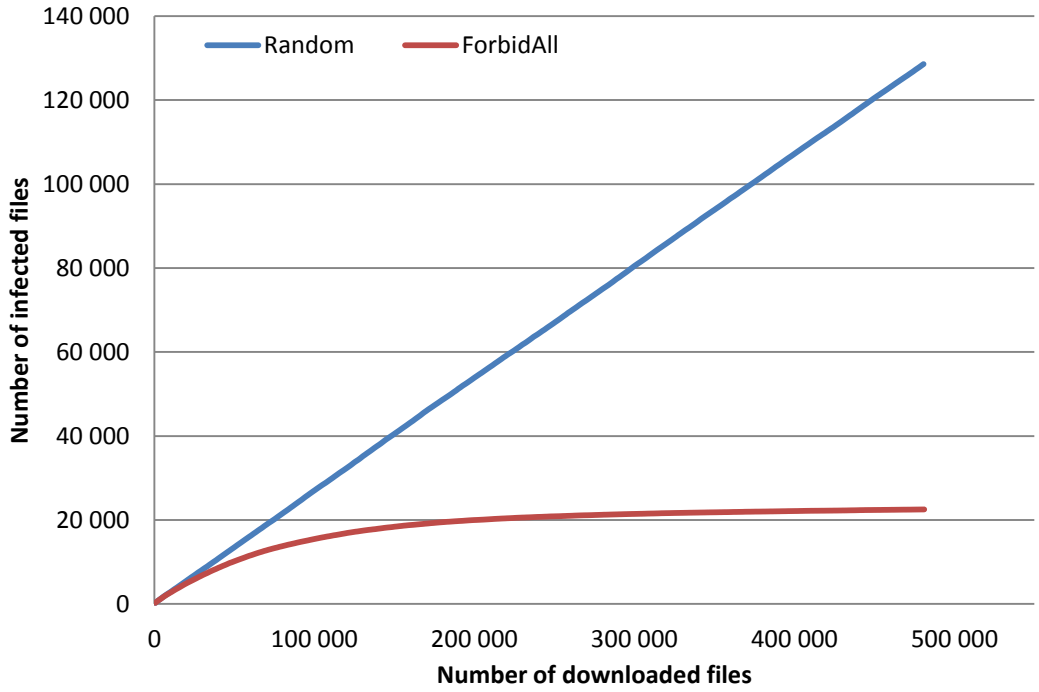


Figure 21: Effect of ForbidAll strategy against attack InfectAll. Network size is 500 and number of malicious peers is 50.

Even more, we can estimate the maximum number of sent infected files. ForbidAll strategy forbids every peer that sends an infected file. There is no cooperation among peers therefore each peer has do it itself. Maximal number of infected files is then $K = M * (N - M) + \alpha * (N - M)$, where N is the network size, M is number of malicious peers and α is fluctuation parameter in percentages.

We visualized this estimation and the result is in the Figure 22.

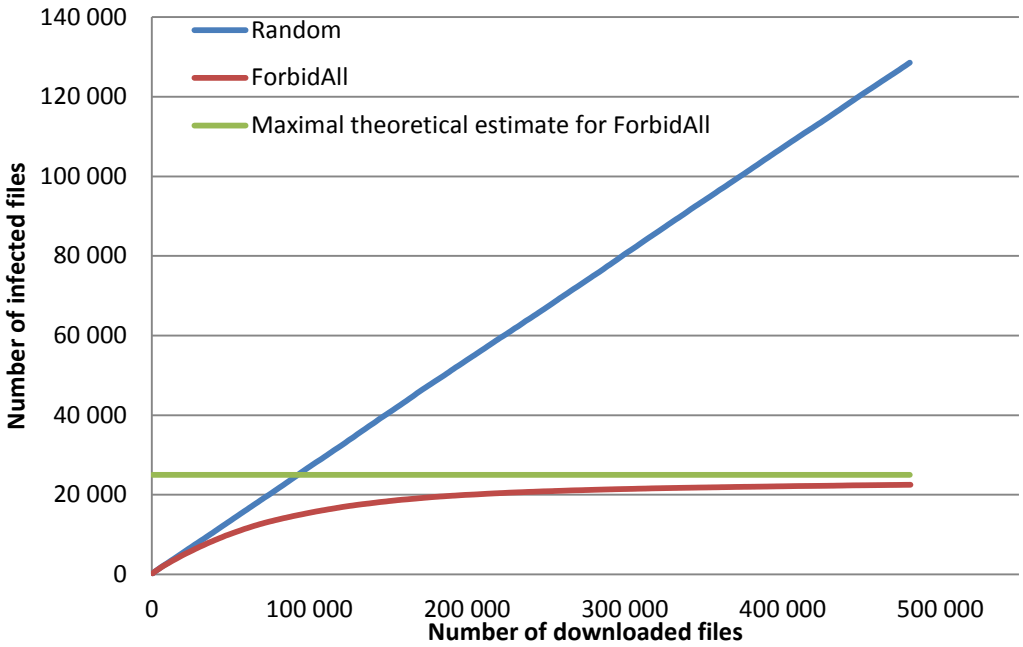


Figure 22: Comparison of measured data with theoretical estimation for ForbidAll. Network size is 500 where 50 are malicious peers with InfectAll strategy.

We can see that number of infected files is approaching the theoretical limit but will never reach it. This is important factor because our own strategy cannot be worse than this one.

We see that ForbidAll strategy is great against InfectAll attack but it has its problems either. The size of forbid tables is still growing. The solution could be some time limit after which they are erased or fixed size of the table. Another problem is that clean peer can send infected file by an accident and it cannot clean itself so peer is forbid for all time.

The biggest problem of ForbidAll strategy is with peers which are new coming, i.e. reconnecting. The peer forbids one malicious peer but cannot forbid a new one because they did not have any communication before. That makes reconnecting very successful strategy. One new coming peer can infect all clean peers. The result of our experiment is in the Figure 23. We can see that ForbidAll is really good against InfectAll attack but Reconnect attack is very successful regardless. Strategy Random is not shown because its values are too high.

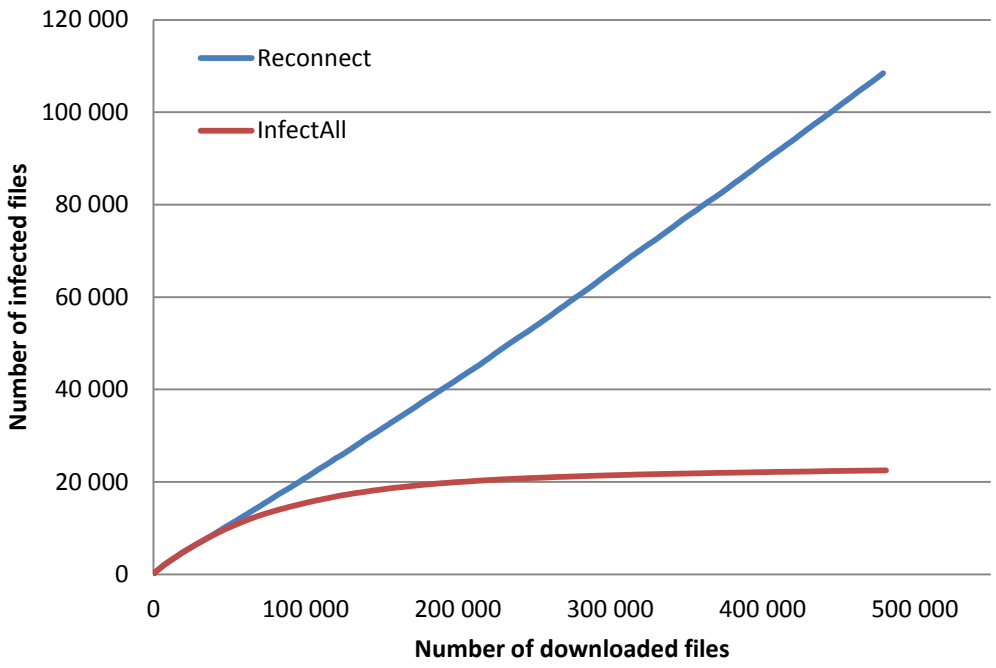


Figure 23: ForbidAll strategy under different types of attacks. Network size is 500 where 50 are malicious peers.

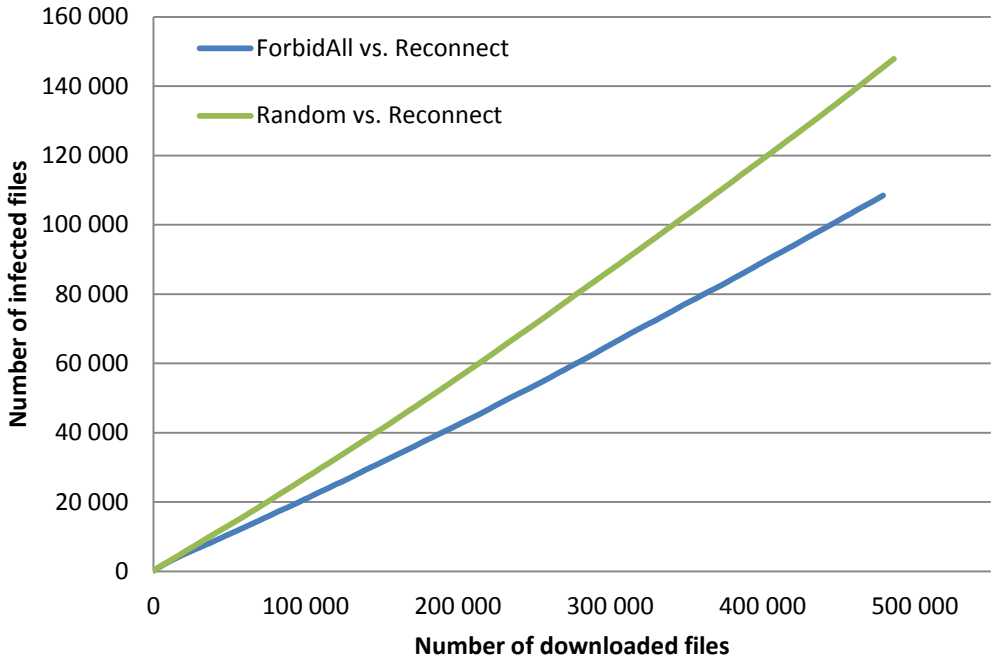


Figure 24: Comparison of different defences against Reconnect attack. Network size is 500 where 50 are malicious peers.

Interesting comparison is in the Figure 24 which shows us efficiency of different strategies against Reconnect attack. We can see that ForbidAll strategy is not good enough but it is still better than Random strategy.

9.3 DefensiveStrategyTrustVectorsB – notCore

Our first interest was in comparison with ForbidAll strategy which is very good against InfectAll attack. This is done in the Figure 25. Graph shows that strategy VectorsB without core implementation is even better than ForbidAll. It is better approximately about 30% what is really good result.

This result is a proof of success of negative based approach. ForbidAll strategy has only two states of known peers – trust or distrust. VectorsB has three states – trust, distrust and unknown. These states are not equal – it is shown in the Figure 26.

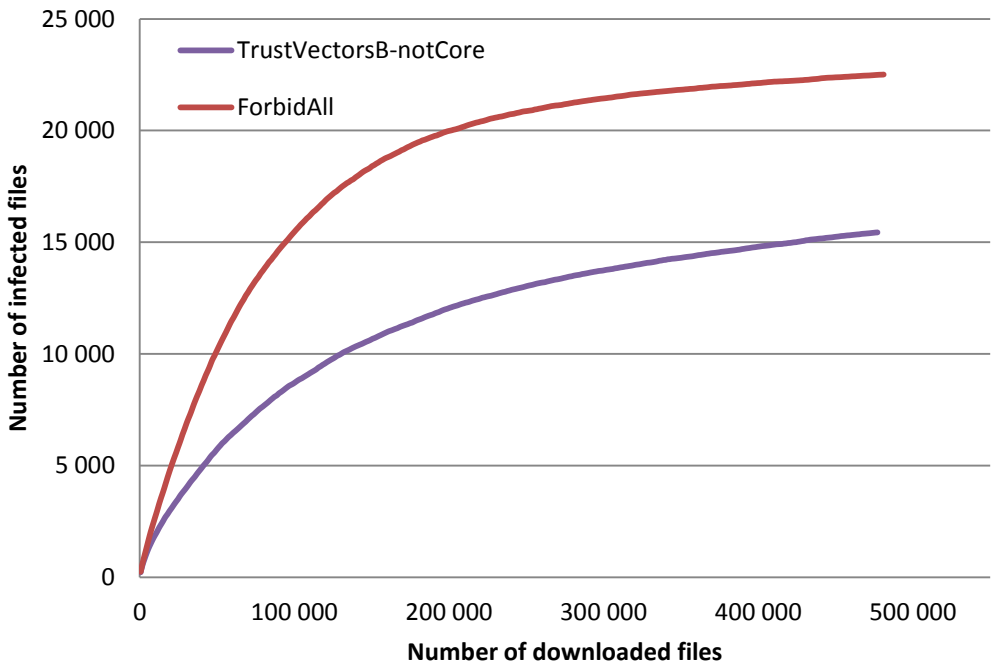


Figure 25: Comparison of different strategies against InfectAll attack. Network has size of 500 peers where 50 are malicious.

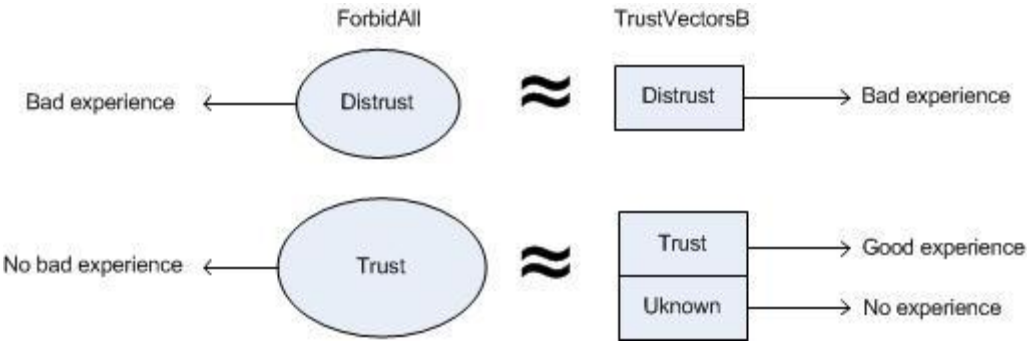


Figure 26: Comparison of VectorsB and ForbidAll strategies. VectorsB has three states whereas ForbidAll has only two states. These states are not equal.

The next testing was with Reconnect attack which proofed to be effective against ForbidAll strategy. Results are in the Figure 27. We can see that VectorsB is doing well – it is approximately five times better than ForbidAll strategy. In spite of this success, the increase of downloaded infected files is not slowing too fast. The curve is almost linear and it takes a long time to slow it down. Our experiments suggest that the break is after approximately two thousand downloaded files per peer. This is a high number which is not always satisfied.

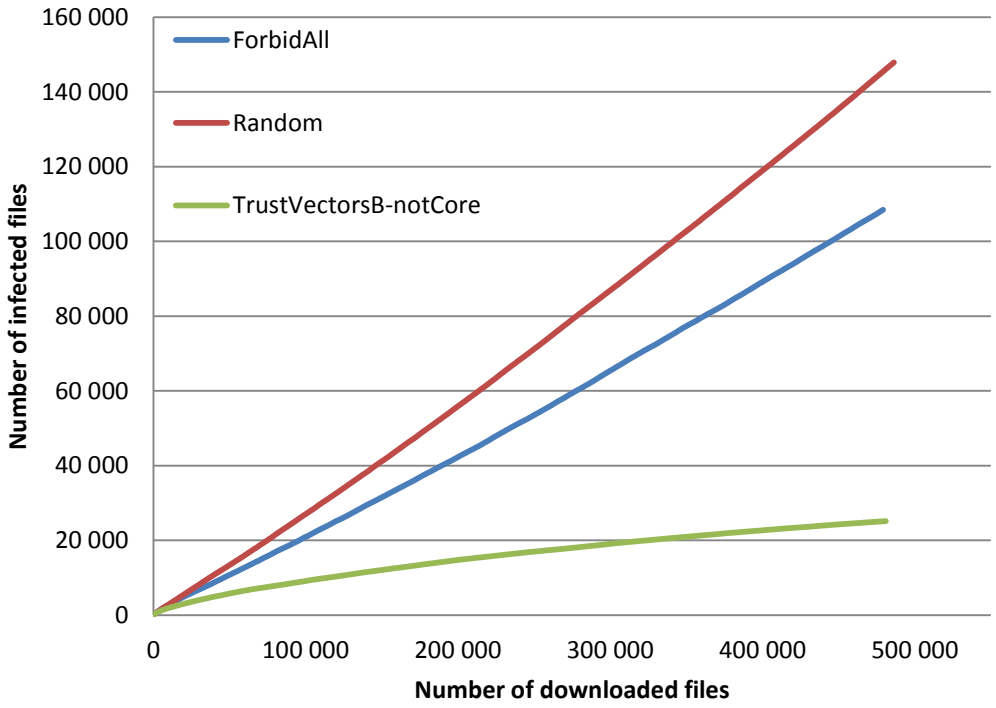


Figure 27: Comparison of different strategies against Reconnect attack. Network has size of 500 peers where 50 are malicious.

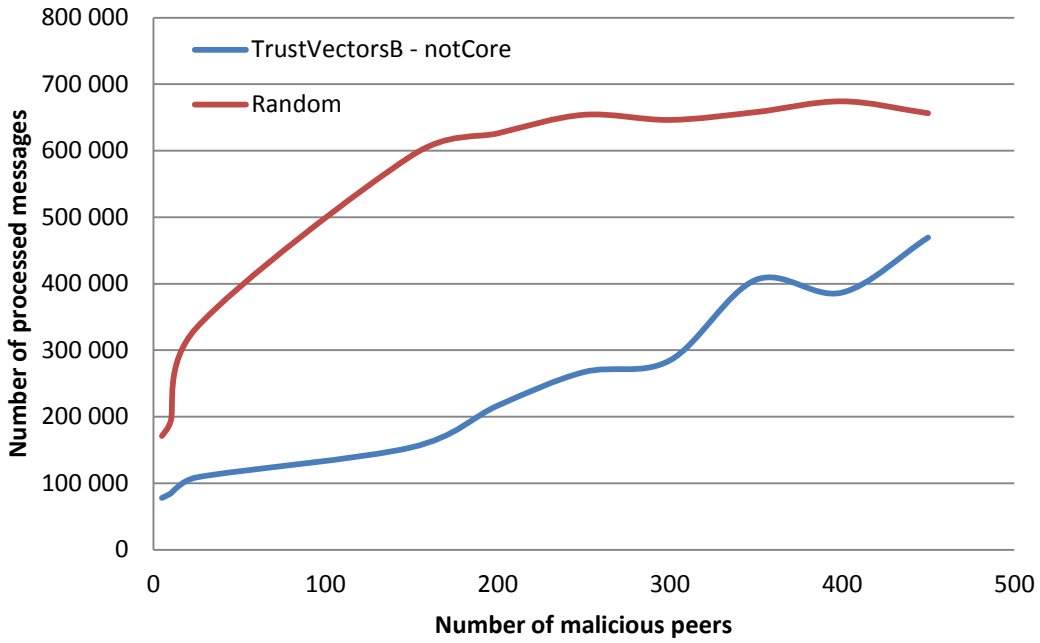


Figure 28: Different strategies under DDOS attack. Graph shows dependency of processed messages by target peer on number of malicious peers in the network of size 500 peers.

Graph in the Figure 28 shows us dependency of processed messages of target peer on number of malicious peers. Attack is DDOS. We can see that strategy VectorsB is doing much better than Random strategy.

Closer look at this DDOS attack is in the Figure 29. We can see the average number of processed messages, the target's number and the maximum number of processed messages by any peer. We see that VectorsB is doing well even when there are 150 malicious peers. When this limit is reached, the efficiency becomes lower. It means that our strategy is effective even for 30% of malicious peers in the network.

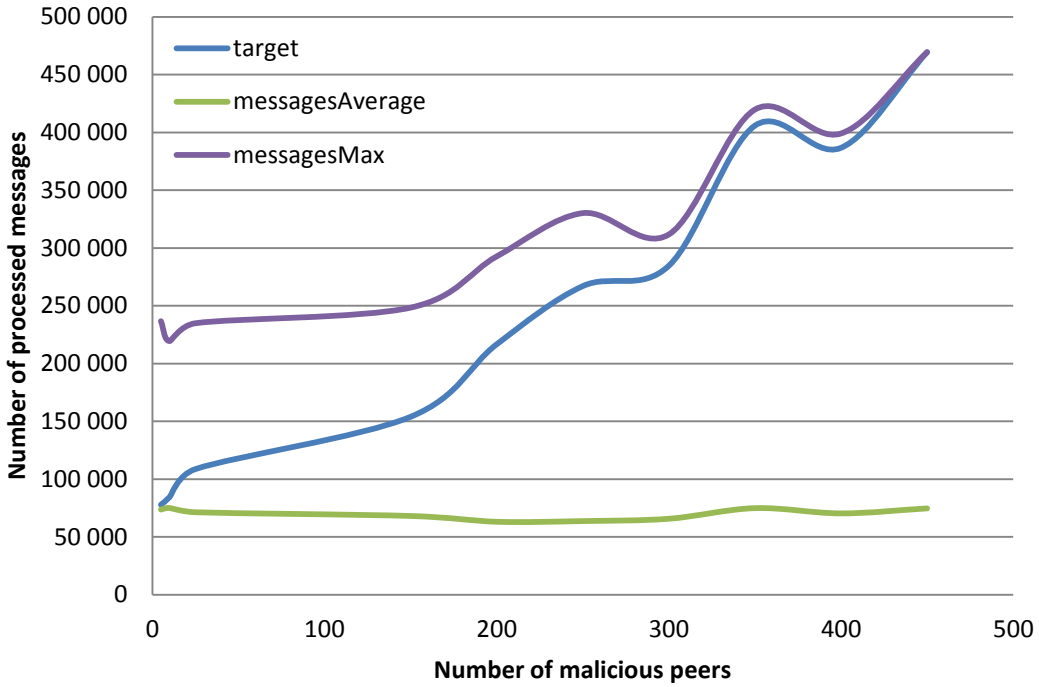


Figure 29: Analysis of DDOS attack and VectorsB strategy. Graph shows dependency of processed messages by target on number of malicious peers.

9.4 DefensiveStrategyTrustVectorsD – notCore

The goal of this strategy was to enhance the possibilities of the strategy VectorsB. It should have been more flexible and soften negative based approach. Results of our simulation with this strategy are in this chapter.

We introduce only one graph where the results are presented. This graph is in the Figure 30. We can see that strategy VectorsD creates a smooth transition between the VectorsB and ForbidAll strategies.

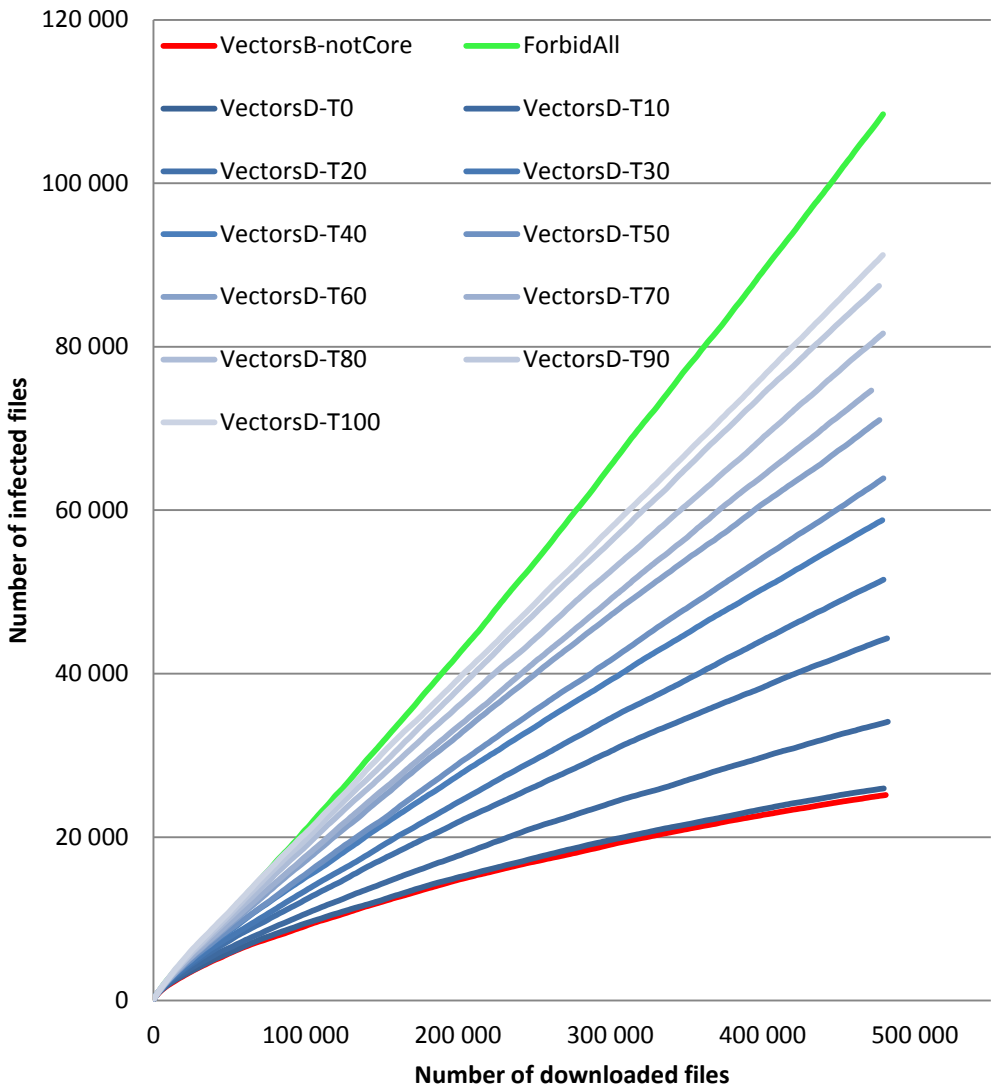


Figure 30: Comparison of VectorsD strategy with other strategies. Network has the size of 500 peers where 50 are malicious. Attack is Reconnect. Parameter T represents trust in the new peers in percentages.

9.5 DefensiveStrategyTrustVectorsB – Core

This defence has the strategy implemented into the core of the routing protocol. Because it was the desired strategy, we compared it with other strategies.

First comparison is in the Figure 31 which shows us different strategies under InfectAll attack. We can see that VectorsB in the core implementation is much better than any other strategy. Core implementation is almost two times better than implementation without the core.

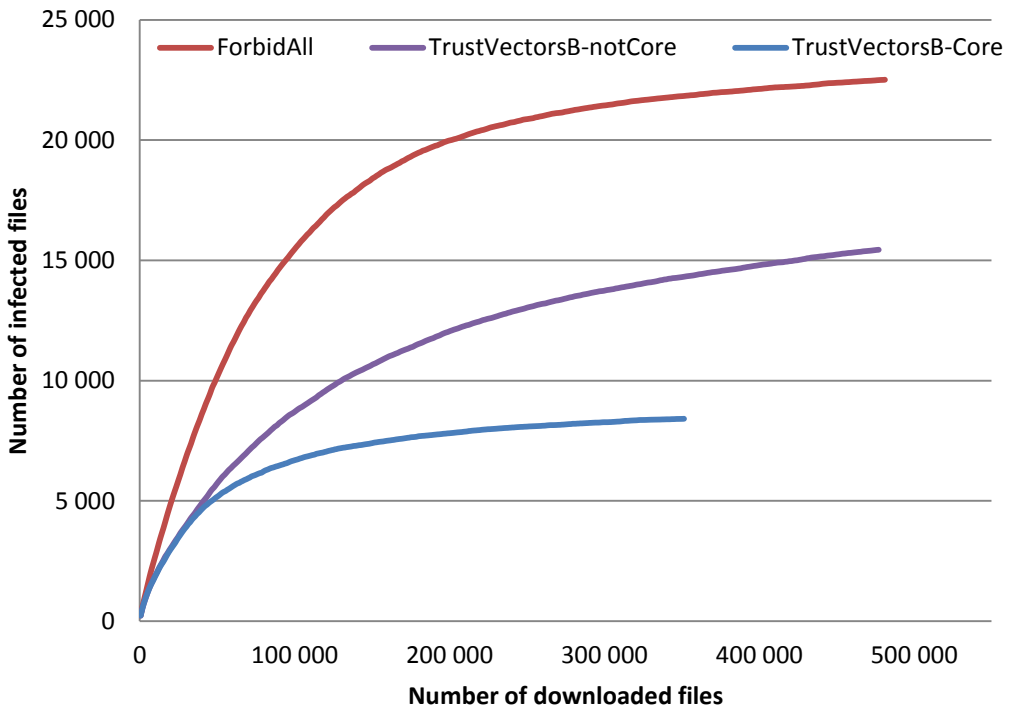


Figure 31: Comparison of different strategies under the InfectAll attack. The network size is 500 peers where 50 are malicious.

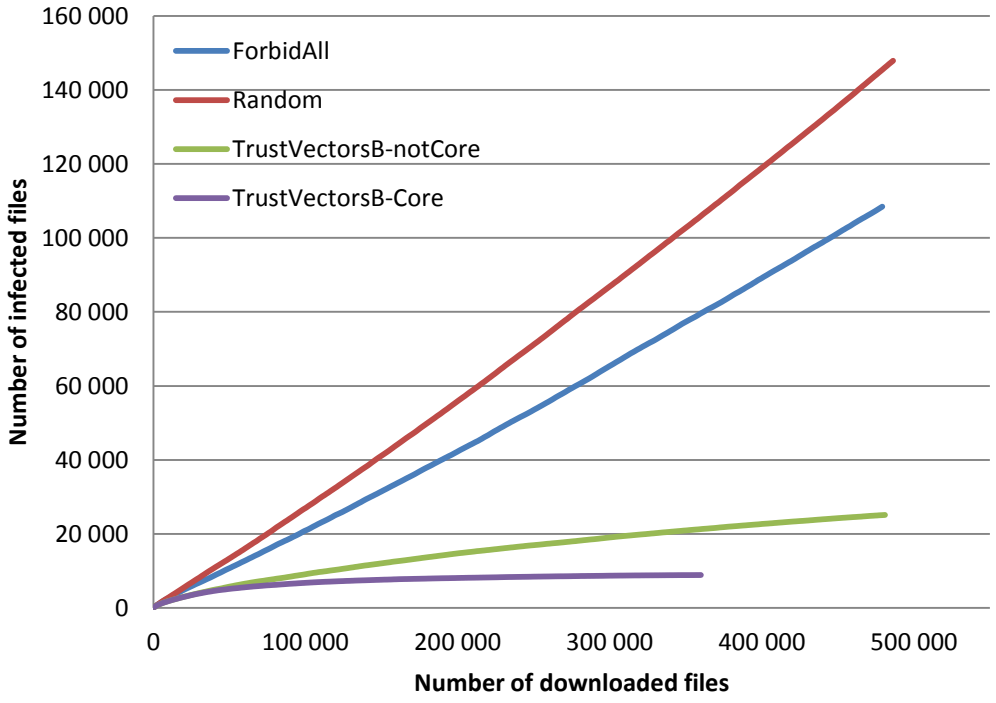


Figure 32: Comparison of different strategies against Reconnect attack. Network size is of 500 peers where 50 are malicious.

Next experiment we did is showed in the Figure 32. It is the comparison of different defensive strategies against Reconnect attack. We can see that VectorsB-Core is again the best strategy. It is almost three times better (in the term of number of downloaded infected files) than VectorsB-notCore and approximately sixteen times better than Random strategy.

We took closer look at the difference between strategies VectorsB-Core and VectorsB-notCore. Result of simulations with bigger network is in the Figure 33. We can see that core implementation is almost ten times better.

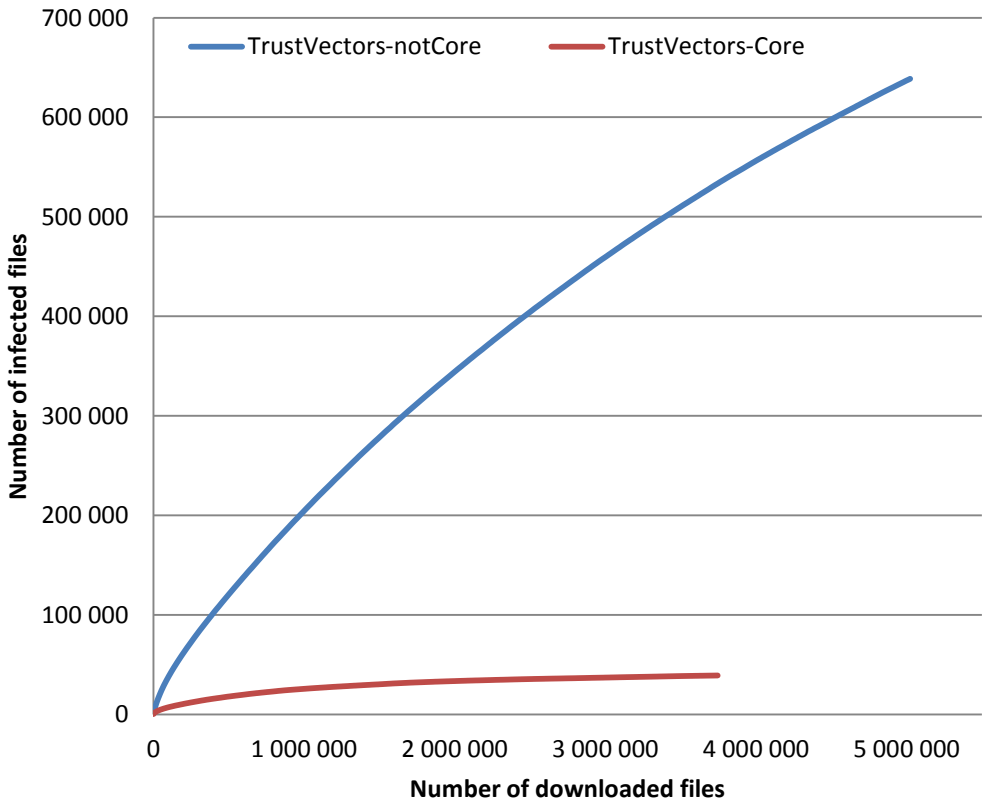


Figure 33: Difference between the VectorsB-Core and VectorsB-notCore strategies against Reconnect attack. Network size is of 5000 peers where 500 are malicious.

Even more, graph for strategy VectorsB-Core has different shape – it is not linear anymore. It can be seen in the Figure 34. It is the same experiment as in the Figure 33 but this time without VectorsB-notCore strategy because the difference is too big to see all the details.

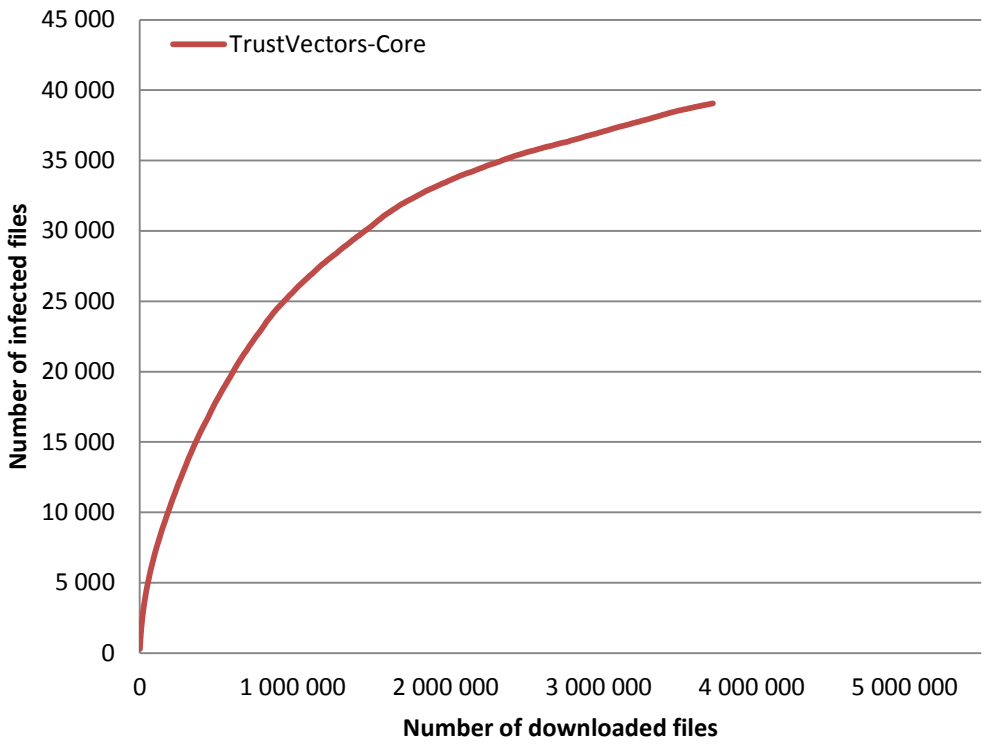


Figure 34: Strategy VectorsB-Core against Reconnect attack. Network size is of 5000 peers where 500 are malicious.

9.6 Summary

The strategies we used have results which are expected. Best example is strategy ForbidAll which results are easily predicted – it is very good against InfectAll attack but is miserable against Reconnect attack. Behaviour of Random strategy was predicted to be simple – dependency of infected files is linear on number of all downloaded files. This behaviour was confirmed by our experiments.

Then we tested our own strategies. We are very pleased about our negative based approach which seems to be very successful. It is well illustrated by results where all our strategies based on this approach have better results than other strategies – even against such simple attacks like InfectAll which is easily defended by ForbidAll and has its theoretical limit.

Afterwards, we did simulations with implementation of our algorithm into the routing protocol core. This showed to be very powerful and effective way of dealing with malicious peers. Even the worst possible attack – Reconnect – was easily

blocked. Core implementation seems to be several times better than implementation of the same algorithm without the core. Ratio of the infected downloaded files for the network where 10% are infected peers is close to one percent and is decreasing over the time. This can be seen in the Figure 34. This makes our strategy very strong and able to compete with strategies based on distributed calculations. (31) (33)

Our protocol has also uniqueness – it is able to deal with DDOS attack. We did not find any other trust management strategy that would be able to do it.

Finally, we did more than 1400 simulation with different parameters. Results presented in this chapter are just the peak of it. We tried to choose representative experiments which illustrate main dependencies. For more results, check the attached CD.

10 Conclusion

This chapter contains information about what was done and what can be done next to continue with the research. It sums all our work and possible future enhancements.

At first, we took look at peer-to-peer networks and theirs types. We assumed that this is not very known problem therefore we described it in detail. Different types of networks were presented and the most common implementations of distributed hash tables were mentioned either. Focus was on the Kademlia network which is the network we have been using in our work.

Our work is closely related to the trust management therefore its definition and brief description were made. We presented several known techniques which are commonly used together with their weaknesses and strengths. Because of need of simulation environment, we had to examine various simulation frameworks. We worked with each of two best for couple of weeks to be sure which one is the best for us. We did change chosen simulator to better suit our needs. All this work is now freely available under the GPL license.

The choices we had to make are in the Chapter 6. We described all our choices in detail. This chapter is important for all who would want to continue in our work. Exhausted description of our own trust management algorithm has been done either. The inspiration was taken from one existing algorithm but we strongly customized it for our work. We designed and implemented its routing core implementation. Settings we used during our simulations are also described and well documented.

One of the biggest chapters is Results. We provided not only all simulation results in the attached CD but we processed them in the form of graphs which are more readable than tables. We did postulate summaries from these measurements which suggest that implementation of trust management into the routing protocol core is very powerful and the results are impressive. This fulfilled the goal of this thesis which was to implement a trust management into the mechanisms of the routing protocol and to compare results with normal trust management.

Finally, we write something about the future enhancements. They are very well possible and even required. It would be nice to visualized created subnetworks in the case of core implementation of negative based approach. It could lead to

improvement of the algorithm. Other possible enhancement could be merging of negative based approach with distributed calculations.

11 Bibliography

1. Peer-to-peer. *Wikipedia.* [Online] Wikimedia foundation, 13 June 2011. [Cited: 14 June 2011.] http://en.wikipedia.org/wiki/Peer-to-peer.

2. **Dooley, Kevin.** *Designing Large Scale LANs.* s.l. : O'Reilly Media, 2001. ISBN: 978-0-596-00150-6.

3. Official Google Enterprise Blog: Destination: Dial Tone -- Getting Google Apps to 99.99%. *Blogspot.com.* [Online] Googleenterprise, 14 January 2011. [Cited: 14 June 2011.] http://googleenterprise.blogspot.com/2011/01/destination-dial-tone-getting-google.html.

4. **Bhagwan, Ranjita, Savage, Stefan and Voelker, Geoffrey M.** Understanding Availability. *In Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03).* San Diego : University of California, 2003.

5. **Google.** Press statistics. *Youtube.* [Online] Google, 2011. [Cited: 15 June 2011.] http://www.youtube.com/t/press_statistics.

6. **McCourt, T. and Burkart, P.** When Creators, Corporations and Consumers Collide: Napster and the Development of On-line Music Distribution. *Media Culture & Society.* 2003, 25, pp. 333-350.

7. Napster. *Wikipedia.* [Online] Wikimedia Foundation, 4 June 2011. [Cited: 16 June 2011.] http://en.wikipedia.org/wiki/Napster.

8. Overlay network. *Wikipedia.* [Online] Wikimedia Foundation, 4 April 2011. [Cited: 18 June 2011.] http://en.wikipedia.org/wiki/Overlay_network.

9. **Novotný, Miroslav.** *Peer-to-Peer sítě.* [DHT Lecture] Prague : s.n., 3 December 2009.

10. Skype. *Wikipedia.* [Online] Wikimedia Foundation, 17 June 2011. [Cited: 18 June 2011.] http://en.wikipedia.org/wiki/Skype.

11. Supernode (networking). *Wikipedia.* [Online] Wikimedia Foundation, 9 April 2011. [Cited: 16 June 2011.] http://en.wikipedia.org/wiki/Supernode_%28networking%29.

12. Gnutella. *Wikipedia.* [Online] Wikimedia Foundation, 29 May 2011. [Cited: 16 June 2011.] http://en.wikipedia.org/wiki/Gnutella.

13. **Ritter, Jordan.** Why Gnutella Can't Scale. No, Really. *Rice.edu.* [Online] February 2001. [Cited: 17 June 2011.] http://www.cs.rice.edu/~alc/old/comp520/papers/ritter01gnutella-cant-scale.pdf.

14. Gnutella2. *Wikipedia.* [Online] Wikimedia Foundation, 18 March 2011. [Cited: 17 June 2011.] http://en.wikipedia.org/wiki/Gnutella2.

15. Hash table. *Wikipedia.* [Online] Wikimedia Foundation, 18 June 2011. [Cited: 18 June 2011.] http://en.wikipedia.org/wiki/Hash_table.

16. Collision resolution. *Wikipedia.* [Online] Wikimedia Foundation, 18 June 2011. [Cited: 18 June 2011.] http://en.wikipedia.org/wiki/Hash_table#Collision_resolution.

17. Consistent hashing. *Wikipedia.* [Online] Wikimedia Foundation, 1 May 2011. [Cited: 18 June 2011.] http://en.wikipedia.org/wiki/Consistent_hashing.

18. Distributed hash table. *Wikipedia.* [Online] Wikimedia Foundation, 6 June 2011. [Cited: 17 June 2011.] http://en.wikipedia.org/wiki/Distributed_hash_table.

19. **Gupta, Anjali, Liskov, Barbara and Rodrigues, Rodrigo.** *One Hop Lookups for Peer-to-Peer Overlays.* Lihue : The USENIX Association, 2003.

20. Content addressable network. *Wikipedia.* [Online] Wikimedia Foundation, 7 May 2010. [Cited: 17 June 2011.] http://en.wikipedia.org/wiki/Content_Addressable_Network.

21. **El-Ansary, Sameh and Haridi2, Seif.** *An Overview of Structured P2P Overlay Networks.* [Pdf document] Sweden : Swedish Institute of Computer Science, 2004.

22. Chord (peer-to-peer). *Wikipedia.* [Online] Wikimedia Foundation, 30 May 2011. [Cited: 18 June 2011.] http://en.wikipedia.org/wiki/Chord_%28DHT%29.

23. Pastry (DHT). *Wikipedia.* [Online] Wikimedia Foundation, 8 March 2011. [Cited: 19 June 2011.] http://en.wikipedia.org/wiki/Pastry_%28DHT%29.

24. Kademlia. *Wikipedia.* [Online] Wikimedia Foundation, 30 May 2011. [Cited: 20 June 2011.] http://en.wikipedia.org/wiki/Kademlia.

25. **Maymounkov, Petar and Mazieres, David.** *Kademlia: A Peer-to-peer Information System Based on the XOR Metric.* New York : Springer, 2002.

26. **Merkel, Ryan.** 25% of Torrent Files Downloaded Are Fake as Companies Continue to Battle Piracy. *Culturetease.* [Online] Culture Tease, 25 January 2011. [Cited: 22 June 2011.] http://culturetease.com/2011/01/25-of-torrent-files-downloaded-are-fake-as-companies-continue-to-battle-piracy/.

27. Trust management (information system). *Wikipedia.* [Online] Wikimedia Foundation, 13 April 2011. [Cited: 25 June 2011.] http://en.wikipedia.org/wiki/Trust_management_%28information_system%29.

28. **Bonatti, Piero, et al., et al.** *An Integration of Reputation-based and Policy-based Trust Management.* [SemanticWeb Policy Workshop in conjunction with 4th International SemanticWeb Conference] Galway, Ireland : Citeseer, 2005.

29. Reputation. *Wikipedia.* [Online] Wikimedia Foundation, 27 May 2011. [Cited: 25 June 2011.] http://en.wikipedia.org/wiki/Reputation.

30. Reputation management. *Wikipedia.* [Online] Wikimedia Foundation, 24 June 2011. [Cited: 25 June 2011.] http://en.wikipedia.org/wiki/Reputation_management.

31. **Kamvar, Sepandar, Schlosser, Mario and Molina, Hector Garcia.** *The EigenTrust Algorithm for Reputation Management inP2P Networks.* [Proceedings of the Twelfth InternationalWorld Wide Web Conference] Budapest : Elsevier, 2003.

32. **Damiani, Ernesto, di Vimercati, De Capitani and Paraboschi, Stefano.** *A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks.* [Proceedings of the 9th ACM conference on Computer andCommunications Security (CCS'02)] Washington, DC, USA : ACM, 2002.

33. **Selcuk, Ali Aydin, Uzun, Ersin and Pariente, Mark.** A Reputation-based Trust Management System for P2P Networks. *International Journal of Network Security.* May 2008, Vol. 6, 3.

34. **Jesi, Gian Paolo.** PeerSim: A Peer-to-Peer Simulator. *Sourceforge.* [Online] 10 March 2009. [Cited: 20 June 2011.] http://peersim.sourceforge.net.

35. Oversim publications. *Oversim.* [Online] Institut für Telematik, Karlsruher Institut für Technologie, 2010. [Cited: 20 June 2011.] http://www.oversim.org/wiki/OverSimPublications.

36. **Leibowitz, Nathaniel, et al., et al.** *ARE FILE SWAPPING NETWORKS CACHEABLE? CHARACTERIZING P2P TRAFFIC.* Tel-Aviv, Israel : Expand Networks, 2002.

List of Figures

List of Tables

Attachments

1. CD with simulation program, thesis in PDF and simulation results.