Scalability Analysis of Dynamic Name Resolution Mechanisms

THOMAS FÄGERHALL

MIKAEL HALLNE

Master of Science Thesis
Stockholm, Sweden 2007

ICT/ECS-2007-14

Scalability Analysis of Dynamic Name Resolution Mechanisms

THOMAS FÄGERHALL

MIKAEL HALLNE

Supervisor
Martin Johnsson
Ericsson AB

Examiner
Assoc. Prof. Vladimir Vlassov
ECS/ICT/KTH

Master of Science Thesis
Stockholm, Sweden 2007

ICT/ECS-2007-14

Abstract

As the coming of a new technology such as laptops and handheld computers the need for mobility in networks has increased dramatically, and this puts a new demand on naming systems that are used world wide to provide to make devices able to connect to each other using symbolical names. In case of a device changing its point of attachment to the network, with today's naming system on the Internet, the DNS, it can take up to 24 hours before the change is visible in the whole system, and that is not a viable solution. To counter these problems a new method called Path-Couple was invented by Martin Johnsson at Ericsson. However, this is not the only new idea, using distributed hash tables has been proposed by the research community and this idea is gaining ground. A previous work has been conducted measuring accuracy and connection times and the Path-Coupled method showed great results. The next step in proving if this method is viable for a real-world use is to do a scalability analysis, meaning to see how well the method can handle a growing amount of roaming users. So in this thesis these three methods are compared against one another to show how Path-Coupled performs against them.

To determine how well the methods scaled in comparison, simulation was chosen as the tool. A few different candidates was examined but OMNeT++ ended up being elected because of its features, documentation and the fact that it comes with open source and being free for academic use. When simulating these naming systems mobile hosts are needed, to do this a mobility model had to be used to make the simulation model as close to reality as possible. Some mobility models were investigated and it turned out that all were suffering from problems, so random walk was chosen to be used because of its ease to be implemented and its popular use in the research community. In random walk a client walks a random distance for a random time then changes its direction to a new random direction and walks a random distance and keeps doing this loop until the end of simulation.

For the simulations there had to be a topology of the network to be used and an underlying map of autonomous systems. A few topology network generators were considered, but it was quickly realized that none of them were good, so instead a real network map were used and scaled to fit our simulation environment, on top of this map extra topology was added to model the connectivity and delay to the naming system and between the autonomous systems. The methods were then implemented into this simulation environment where they were run with the same settings.

At a thorough analysis it was showed that the Path-Coupled method scales well in comparison to both the DNS method and the DHT-based method on all points except for the load on the common Resource Location Register. It was however investigated if the RLR could be changed from a monolithic structure to a DHT or a DNS based solution. And it was found that using a DHT would be the best option so as to spread the load and make it scale.

Acknowledgements

We would like to thank our supervisors, Martin Johnsson at Ericsson and Vladimir Vlassov at KTH, for their help during the work of this thesis.

Table of Contents

List of Figures

List of Tables

1 Introduction

The task for this master thesis is to investigate the scalability aspect, during high mobility, for a newly proposed name-to-address resolution mechanism. The performance of this method will be compared against the existing (dynamic) DNS solution, and a DHT-based method. This work is the continuation of a previous master thesis work, conducted in the spring of 2006. The evaluation will be done by implementing the different protocols in a simulation environment and from the results obtained; one or more conclusions should be drawn.

This work is a cooperative work by Thomas Fägerhall and Mikael Hallne, both Master of Science students at KTH (Royal Institute of Technology), Stockholm. The work has been conducted at Ericsson Research premises in Kista, Sweden.

1.1 Background

In the beginning of the Internet there were numbers, just long strings of numbers such as IP addresses. These numbers were quite hard to remember for a human mind so a name space with names easy-to-remember and to manage was introduced, and which then resulted in the need for name-to-address mechanisms. How ever these name-to-address resolution mechanisms are designed in such a way that it assumes the addressee remains on the same address forever or at least does not change often.

The idea mentioned above which assumes fairly static bindings between names and addresses is no longer the case in modern networks. Because an addressee might change address, i.e. location, every day or hour or even minute (seconds!), and these updates are not being propagated through the network fast enough. Take the example of the Domain Name System (DNS) where it might take up to 24 hours before a change is seen. This means that DNS could be completely useless for an addressee that changes address every hour. What is needed is a new name-to-address resolution mechanism that propagates the changes much quicker, preferable instantly.

The need for mobility, i.e. having connectivity while moving around, led up to Martin Johnsson's idea of a new mechanism, a novel path-coupled name-to-address resolution scheme, hereafter referred to as the Path-Coupled method. To see if it is a viable alternative, there are a number of properties that it should have such as overall low connection establishment time, good scalability and more. The viability is compared against two other name-to-address mechanisms, Dynamic DNS, and a mechanism based on Distributed Hash Tables (DHT). Three master students spent the spring of 2006 to investigate two of these important properties, connection time and how well it performed in a network with highly mobile hosts. Now it is time for us to investigate one of the other properties, namely scalability. The scalability aspect that is investigated is regarding the load on the systems and the performance of the methods as the number of users/clients in the system increases.

1.2 Motivation

This is the second step in the ongoing verification of the novel path-coupled name-to-address resolution protocol. The results obtained with this thesis will work as a fundament for further input to the Ambient Networks Project. The Ambient

Networks Project is co-sponsored by the European Commission. The projects goal according to their web site is: *"…create the network solutions for mobile and wireless systems beyond 3G."* [44] Ericsson AB is coordinator of the project.

The innovation, proposed by Martin Johnsson at Ericsson Research, is one out of many contributions from Ericsson to the Ambient Networks Project. The Path-Coupled method should be able to handle and is indeed specifically addressing the support for a highly dynamic network environment.

1.3 Problem Statement

The three methods that will be compared are: Dynamic DNS, a DHT-based name-to-address resolution method, and the newly proposed Path-Coupled method.

The objective of this thesis is to examine the scalability of the three different name-to-address resolution methods. Due to the expected high mobility in the network the methods will perform differently based on the differences in the three techniques with respect to how they handle name spaces, how lookups and updates are performed, and their different needs regarding memory and CPU. Some of these differences are seen in previous work (Dynamic Name-to-Address Resolution. Evaluation of a novel name-to-address resolution method, MSc thesis, ICT/ECS-2006/89, School of Information and Communication Technology, Royal Inst of Technology (KTH), Stockholm, 2006. Mathias Johansson, Kristian Olsson, and Patrik Åkerlund) regarding the same protocols. Our work is to see how they behave when the system grows, e.g. when a large number of mobile hosts is introduced that move around in a network topology according to a predefined model. A large network will be simulated, with as many users and terminal entities as possible, that will perform name-to-address resolution lookups and registrations as part of their normal network usage.

The result from the simulation will show how the different methods perform under increasing amounts of load. The result will be evaluated and the three different methods will be compared to each other. The result will be presented in this report.

1.4 Evaluation

It is expected to determine how well the evaluated name-to-address resolution mechanisms scale. There are two aspects to be considered in the scalability, first it is the network traffic, how much traffic will be user traffic and how will it compare to the amount of control traffic.

The second aspect to consider is how the need for memory and processing power in the machines scales with regard to the number of users.

Since the Resource Location Register (RLR), in the Path-Coupled method, contains all information about all the hosts/users in the network the distribution aspect of the RLR needs to be evaluated.

These criteria will be evaluated for the Path-Coupled method in comparison to the other methods.

1.5 Existing Solutions

Today the DNS system is in charge on the Internet. It takes care of the task of mapping between symbolic names, i.e. *www.ericsson.com*, to IP addresses. The design of the DNS system is rather old, and the conditions on the Internet have drastically changed since it was designed. A resource that needs to be reached via Internet is more mobile today, and the trend is that the mobility will increase. [40]

In the GSM system the E.164 number plan is used, by recommendation from the ITU-T. The E.164 name space defines the format of telephone numbers and can handle numbers with up to 15 digits. The protocol that handles number mapping is ENUM, which is a DNS-based solution. [43]

1.6 Related Work

There has been some work done in the area of DHT-based naming systems. This is a quite popular approach and there exists several research articles that evaluate a DHT-based solution that could replace the existing DNS solution. The idea is to replace the existing hierarchal structure with a flat name space, and also to improve the availability and load balancing in the system. [39]

One existing solution is the CoDoNS (Cooperative Domain Name System) [41] that is deployed on the peer-to-peer overlay Beehive [42]. Beehive is a DHT that provides an O(1) lookup time, fast and optimized replications, and according to the projects web page adaptive behavior that lets the system respond to sudden changes in the network.

1.7 Personal Contribution

Since this project is group effort, the work has been divided amongst the two of us. Thomas will study and evaluate simulator candidates as well as studying how to measure CPU and memory usage for the different methods, Mikael will study traffic models, traffic generation, mobility models, topology generation, and the distribution aspect of the RLR. During the implementation and evaluation phase Thomas has been in charge of the DNS method and the Path-Coupled method, and Mikael has taken care of the DHT/Chord-method.

1.8 Structure of the Report

Section 2 contains a background over naming systems in general, and particular the DNS system, DHT-based systems, and the novel Path-Coupled method. Further this section contains background information on the available simulators. Section 2 continues with a background for the simulation framework, containing information about telephone traffic and Internet traffic. Different mobility models are presented and evaluated. The section ends with information regarding the distribution aspects on the RLR in the Path-Coupled method. Section 3 contains the method and describes the design choices regarding the models built. The next section, section 4, describes the implementations done. It describes how the three mechanisms have been implemented and how the simulation model is developed for the simulations. Section 5 contains the analysis of the result from the simulations. The results from each name-to-address method is analyzed and presented. The last section is section 6 Conclusions and Future Work.

2 Background

This Section of the document explains the three different methods that will be investigated in general terms. It also provides information about traffic patterns, traffic generation, mobility models, and topology creation considerations. To round off, the section is ended with background information regarding the distribution aspects and the deployment aspects of the Resource Location Register.

2.1 Scalability Aspects

In this section follows an explanation of what scalability means, followed by a brief presentation of the aspects investigated in this thesis.

Defining what scalability means is quite hard. The web site dictionary.com gives the following definition: "*How well a solution to some problem will work when the size of the problem increases.*" This definition is a good starting point, but does not give a full explanation. The compliment needed is the aspect how well a problem gets solved when the scope of the solution is increased. There is two ways to scale a system, horizontally or vertically according to Wikipedia.org's article on Scalability. Horizontally refers to a systems ability to expand in size, while vertically refers to the ability to expand the capacity of the system. In this thesis the horizontally aspect is investigated, due to the fact that capacity is not achieved by the method itself but the underlying hardware or implementation.

It is not an easy task determining how well something scales since there often are many parameters. Some systems might accept more then one behavior, and how all those behaviors correlates with each other might be very complicated. Therefore it becomes very important to limit the parameters and to understand what is being measured and what consequences it has for a system. When understanding the measurements it is not very interesting to look at precise measurements, it is more interesting to compare trends between different measurements and in this way gain an intuition of how it would work.

This thesis will evaluate how different parts of the naming systems are loaded when the amount of users increases. It is interesting to find out how the different systems work in comparison to each other. In this thesis the corresponding parts of each method will be compared to each other concerning how different parts (mainly servers) in the network is loaded as well as how storage of information is distributed among them. More information about this will be given in section 3.2 and 3.3.

2.2 Naming Systems

The primary function of a naming system is to translate a name into a reference of an object. This can be done in many ways; the most instinctively is to have a list of names and their corresponding references, and this was also how the first naming system was constructed. However, this system quickly suffered from problems as more and more people started to use it, since the shared list is hard to keep it up to date while distributing it to everyone. Therefore more complex naming systems were created.

Today naming systems are primarily used to help humans remembering address on the Internet, and the most used and known system is DNS, which there will be more written about in section 2.2.1.

Providing name to addresses translation in an accurate and efficient manner is very important, to achieve this, it is important to have a correct and consistent directory of mappings. To do this it was agreed upon that a naming system should at least provide the following functions to do so: Bind, Rebind, Lookup, List binds and Remove bind. [23]

It is not only important that such a system is correct, it is also very important that such a system scales well; meaning that it balances the increased load on the system well. A more detailed definition of scalability can be found in section 2.1

The different naming systems have different namespaces depending on how they are constructed. A namespace is all the valid names in a naming system. One example could be all words less then eight characters using only lowercase characters. Then there are two different kinds of ways to organize namespaces: Flat namespaces or Hierarchical namespaces. In a flat namespace the data remains unorganized while in hierarchical it is somehow organized, DNS for example, which is most known naming system, uses a kind of tree structure to organize all the names.

2.2.1 DNS

DNS is the biggest and most widely used naming system on the Internet today. It was invented in 1983 by Paul Mockapetris [40] and has over the years had numerous add-ons as different needs have emerged.

DNS stands for Domain Name System, this is due to how the namespace is portioned into domains, and each domain is managed by one or more DNS servers. This is to distribute the load as well as distributing the management of systems, making it maintainable.

DNS is organized in a hierarchical structure, a tree as can bee seen in Figure 1 below. To explain this, an example address will be used: *it.kth.se*. It can be observed that the address is split into four different parts separated by dots. The different parts of the address represent different levels of the structure. First is the empty string that is to the right of the last dot, this represents root which is the highest level of the structure and is static, meaning it never changes location. The root only stores information about the next layer of the hierarchy. These sublevels in their turn hold information of what is under them. This continues in the same manner until the lowest level has been reached. The picture below illustrates how the hierarchy is organized and the example of *it.kth.se* can be located. [23]

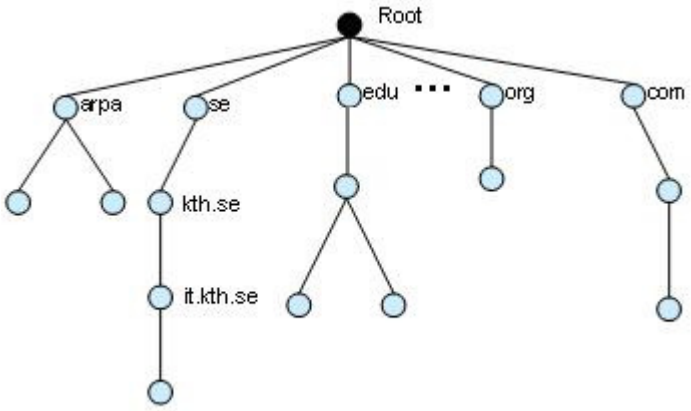

Figure 1: Hierarchical structure of the DNS system

When using a DNS system from a device, such as a PC, the application sends its lookup request to the local DNS server. The local DNS server is often a server located at the ISP. In case this DNS server does not know the answer already through caching, two methods can be used during the rest of the resolving process, either iterative or recursive. In an iterative process the local DNS server will first ask the root for information of where *it.kth.se* is located, root would not know better then to give a reference to *.se* that would know more. The local DNS server would then ask *.se* the same question, "where is *it.kth.se?",* it will carry this on until the request has been resolved or it is shown impossible to resolve. It would then send back the result to the initiator. The recursive method works in a different manner, it would also ask root for where to find *it.kth.se,* but instead of replying, root would send the request further to *.se* which would in its turn send it to *kth.se* and so on until it would be resolved, and when it would reach the end it would be sent back the same way it came, all the way back to the initiator. Figure 2 shows the lookup process when using the iterative method, and Figure 3 shows how a lookup is performed when using the recursive method.

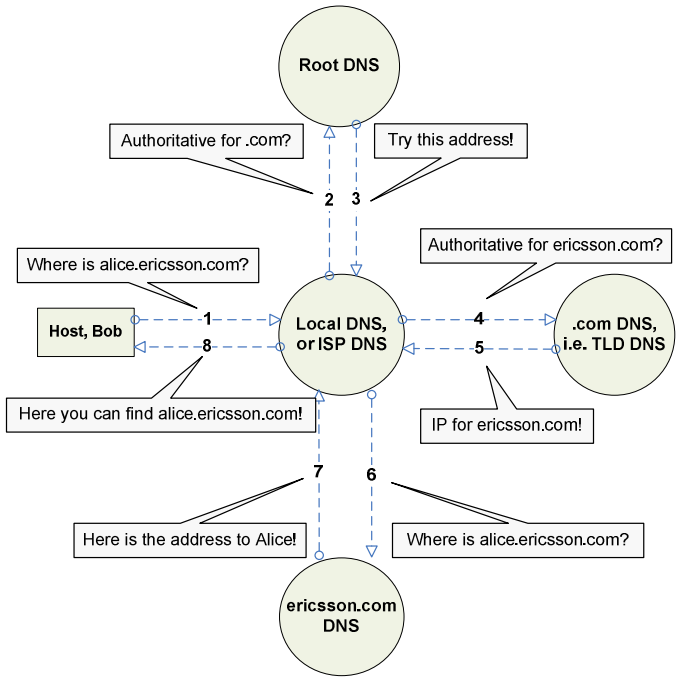

Figure 2: Lookup process using an iterative DNS procedure

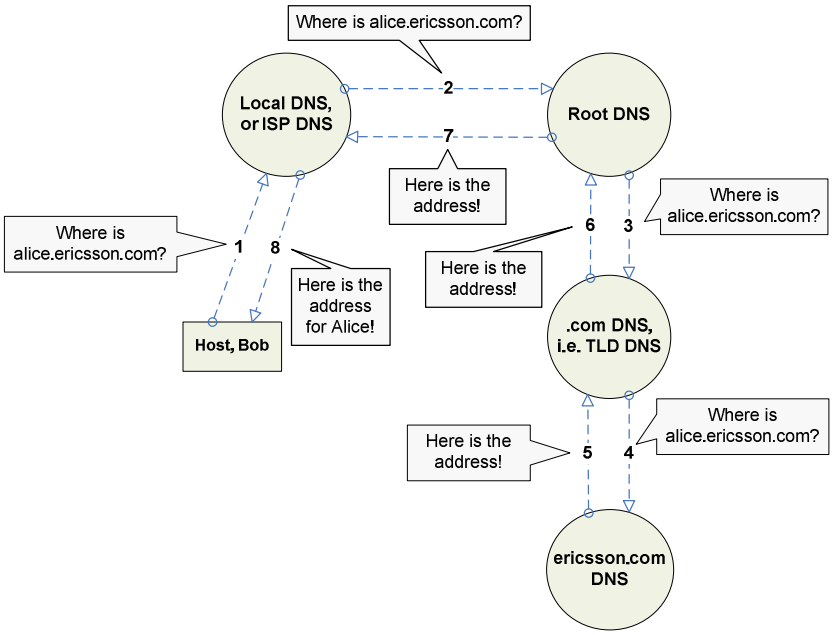

Figure 3: Lookup process using a recursive DNS procedure

When updating a record in DNS the server in charge of the address is contacted and the update is made. Then the DNS server either tells its secondary servers of the change that has been made or the secondary servers are polling the primary for changes in even intervals. Eventually the change will be seen. It is often the case that a server finds out about a change when the timer for caching the reference has been expired.

A problem with DNS is that a change is not visible in the whole DNS system instantly, and this is because of caching. When a local DNS server resolves an address it will cache all the information it comes by, to invalidate this information there is a Time To Live (TTL) value associated with each record. Not until the record has expired the value will updated from the servers. Because of this mechanism there is a chance that a change in a DNS record will not be visible to all users until after perhaps 24 hours, which depends on the TTL value of the cache. This can sometimes have big consequences.

2.2.2 DHT

DHT works in a radically different way from DNS. Instead of having a hierarchical namespace it has a flat namespace. To store all name records DHT uses a distributed hash table. A distributed hash table is an overlay network that looks like a ring and all the nodes have a unique number associated with it and the stored records are spread out using a hash function. All the nodes are connected with their neighbors, their predecessor and their successor, see Figure 4.

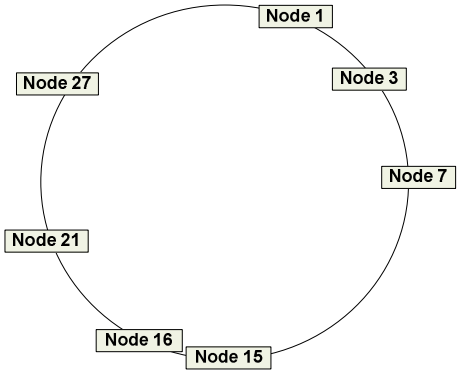

Figure 4: Shows a DHT ring where all nodes are connected to its successor and predecessor

When a node joins a DHT ring it is given an ID and becomes responsible for all IDs between itself and its predecessor including its own. The node is also given knowledge of who its successor and predecessor in the ring is.

Storing information in a DHT is done by hashing the identifier for the information wished to be stored. This hash value is then used to identify which node in the ring that should store it.

When retrieving information from the DHT the sought identifier is hashed and then the node responsible for this hash value is contacted. When contacting a node the request is sent in the ring over each node in that direction, coming closer and closer to the sought node, the information can then be passed back recursively or it is passed back directly to the user depending on the implementation.

There are many implementations of the DHT, Chord is one of them. Chord is extended from the basic DHT. One of its extra features is a finger table; this table contains information about nodes further ahead in the ring. [46]

A Clarifying Example

The network is portioned in such a way that every node has its own unique ID in the DHT rings number range. If the ring has a range of 32 numbers and there is seven nodes in the DHT at the numbers 1, 3, 7, 15, 16, 21 and 27. Then node 3 will be responsible for the numbers 2-3 and node 16 will be responsible for number 16. If a person makes a call for a name that turns out to be number 22 and starts asking at any node but in this example it starts at node 15, node 15 will ask at node 16 that will ask at node 21 that will ask at node 27 that will have the answer that will be sent back to the node that initiated the lookup inside the ring, in our example node 15. This node will then send the information back to the user that made the lookup. In Figure 5 there is a more detailed illustration of this.

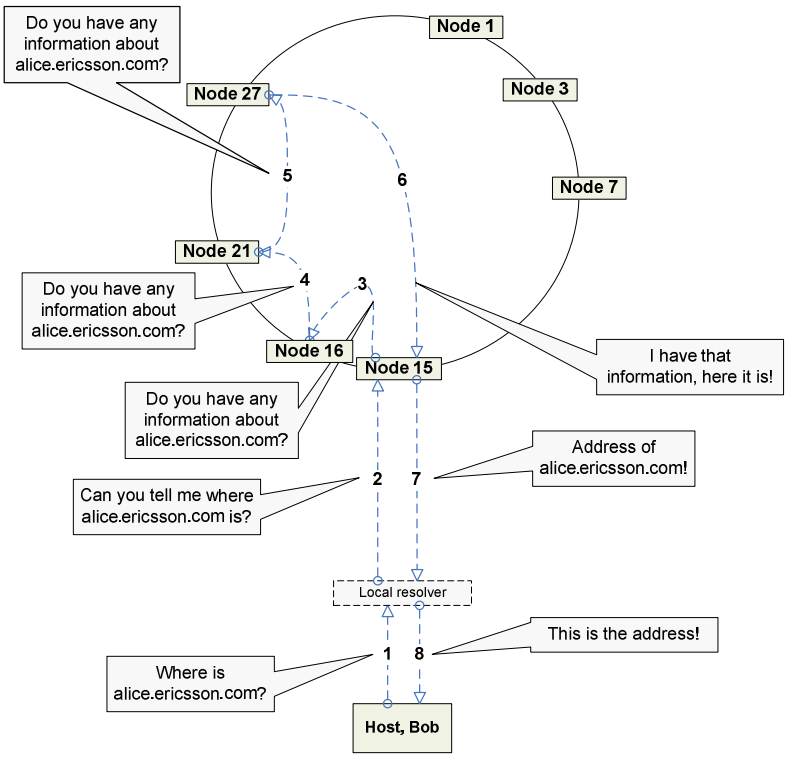

Figure 5: A recursive DHT-based name-to-address resolution procedure

The DHT nodes also have a finger table. The size of the finger table is equal to the number of bits in the key space. In the finger table there are some entries of nodes that are much further ahead in the ring. This is to make it faster for a lookup to take place, because a jump through the finger table will take you much further in the circle. And it is perfectly safe since one will only jump to nodes that have a number less then the one you are looking for. In Figure 6 there is an illustration of this.

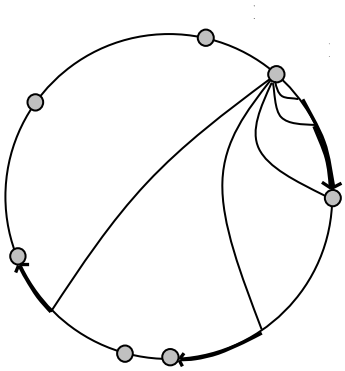

Figure 6: Shows a node's finger table connections to other nodes

In DHT when updating a record the node in charge of the name is contacted and the name is changed. Then the change need to be propagated to update the cached values, if there is any.

There are some different methods suggested for caching. First is for recursive lookup, the data is propagate back towards the node where the user made its request. Along this way the information can be cached. Second method would be to cache all information from one node to its predecessor. A third way would be in the iterative version where one node does all the questions in the ring for the resolving client; this node can cache the data that is being sent back. This would mean that any data can be cached at any node, since lookups can be done from any node.

2.2.3 Path-Coupled

Path-Coupled name resolving takes on a completely different approach. Instead of first looking up an address and then making the connection it makes the connection at the same time as looking up the name. This means that the address resolution is made in small steps, every step taking you close to the end destination. The next paragraph will show the structure.

The Path-Coupled method is organized as a tree and at the root there is the Resource Location Register (RLR) that holds records of all the names in the system and assigns them a Resolution Transaction Code (RTC), the RTC is a reference code, it is used to refer to the name and is used to hide the real name of the host, and this is then what is stored in the level beneath, a new RTC is used for every level. At that level (RS of level 1) the RTC is mapped to another RTC and an address to the responsible RS of level 2. It keeps going like this until the lowest levels of RS has been reached; there the RTC is just mapped to the address of the sought host. This is illustrated in Figure 7. [23]

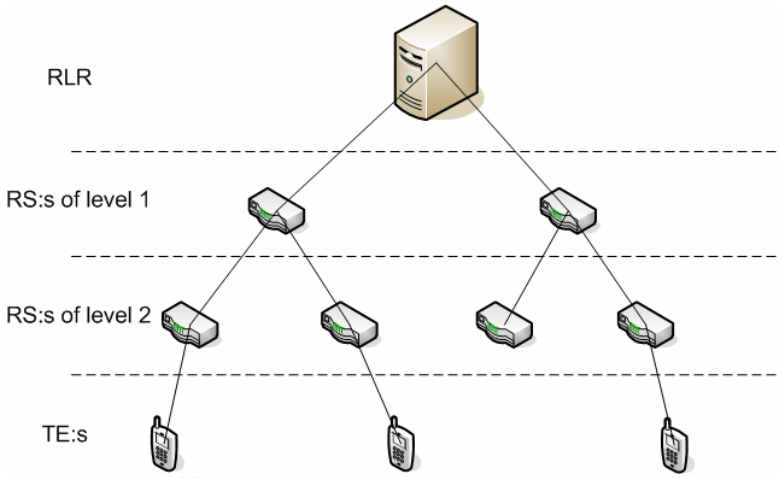

Figure 7: The logical view of the Path-Coupled hierarchy

To make a connection using the Path-Coupled method would work as follow:
The connection initiation goes from a caller to its closest router, that router then asks the RLR to find out to in which highest level RS-area the sought host is in. The packet is then sent from that router towards that area, the first router in that area will pick the packet up and then make another lookup to its area RS to find out where to route this packet next. When this information is found the router then routes the packet further, it continues on the same manner until the area which holds the sought host has been reached, it then instead forwards the packet to the sought host, and the connection is starting to initialize. How the connection then is really made is up to the implementation, it can either be done by sending the handshake back the

same way, or doing a lookup to go back. Figure 8, below, shows how a lookup is made together with a stateful reverse path. [23]

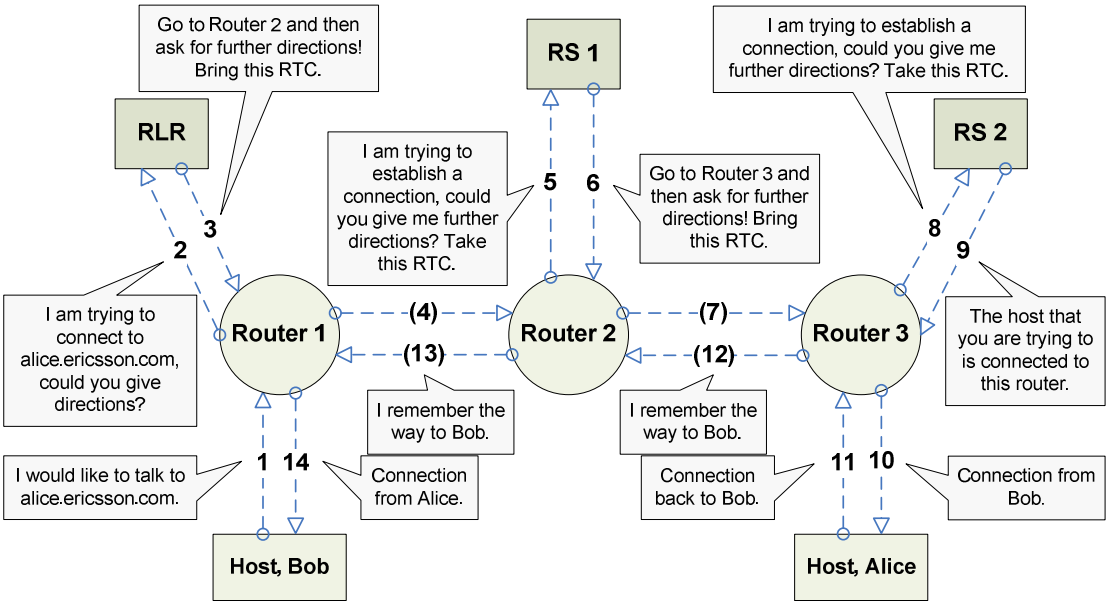

Figure 8: Path-Coupled name-to-address resolution procedure

Updates in the Path-Coupled method are divided into different categories depending on how high up in the hierarchy it has affects. The highest is "full hierarchical movement" and the others are named "hierarchical movement of level X" where X is number of the highest effected level, levels can be found in Figure 7.

When a host has moved from one RS to another the host contacts its new RS and gives the name of the old RS and its old RTC. The new RS then creates a new RTC for this host, and send it along with the update message to the RS above, the new RTC is stored there, and sent higher, if this was the highest affected node in the hierarchy it sends a message to delete the old RTC records or alternatively they are timed out. After the update is done a change can instantly be seen in the system, this is great for mobility since they will be reachable directly. [23]

2.3 Simulator Evaluation

2.3.1 Evaluation Criteria

In this section a number of different simulators, simulation-frameworks and emulator software's will be evaluated to find the most suitable one for this thesis. First the important criteria will be investigated. This will be done in two sections: functional and non functional. Then an overview evaluation of simulation/emulation software will be presented, followed by a more detailed description of the relevant ones. And last a more thorough look at the chosen simulator.

Primary Functional Evaluation Criteria

When performing a scalability analysis the number of independent nodes in the simulation is the a very important factor, however, its not just the ability to have many nodes that is the key it is also how high throughput the simulator has since it does not matter if it can have 100000 nodes but not finish within a reasonable amount of time.

The random number generation is also an important factor when choosing a simulator. There are a number of properties that should be fulfilled to classify a random number generator as good. However, random numbers are a science itself and will not be addressed more thoroughly in this thesis than what aspects that needs to be taken into consideration. Random numbers have to be "good", meaning they are proved to be random enough with certain properties. Also, the numbers generated has to be independent of each other, the software has to support that all different random number users can have different streams to take numbers from. Otherwise the results would not be reproducible and thereby not credible. This is due to the non-deterministic properties of distributed simulations. The non-deterministic behavior comes from that different nodes of the system will not always act in the same order and therefore the same trace of events will not always occur. However, this is a science on its own, and will not be dealt with further in this thesis. The random sequences have to be long enough so they do not wraparound and starts to use the same numbers again. [24]

Support for statistics gathering is also important, since simulations are pointless if they can not be measured in the correct way. There is often many different ways to collect data.

The possibility to model mobility is important because the thesis focuses on methods that have a high support for mobility, therefore it has to support mobility or at least have some ability to mimic/simulate it.

Secondary Functional Evaluation Criteria

There are other contributing aspects as well, such as if a simulator has already pre-developed models that can be used to save time.

As mentioned previously a simulation result should be reproducible. This is to ensure that the simulator gives the same answer for the same input. [25]

Non-functional Evaluation Criteria

The license of the software is a non functional criterion. The license of the software has to be a free license. This limits us to software that is either free for students, for researchers or for everyone.

Documentation is another important non-functional criterion. It speeds up developments because learning goes faster and also there are resources to read when things have gone bad.

To Summarize

What is needed for this thesis is a scalable simulator, with good support for random numbers, distributed simulations, good statistics collection and preferable it should have pre-developed models that can be used. Also, the results should be reproducible and the simulator should be well documented and be available with academic license.

2.3.2 Primary overview of candidate simulators

In this section an overview of the interesting simulators found will be presented. There where several more simulators looked at but they are not mentioned, because

they are too many. These simulators have been discarded either because they where obvious not to fulfill our demands, information about them was too hard to come by, or they where commercial.

Ns-2

Ns-2 is a very popular simulator used for network simulations. The previous thesis had been performed using this simulator. However, the memory consumption is a big problem is ns-2. In the previous thesis they used about 2 GB of memory to simulate around 4000 nodes [23]. This makes it very difficult to even consider ns-2. There is a distributed version of it, but it still suffers from the same memory consumption problems. So ns-2 is not any real choice.

Opnet

Opnet is the biggest commercially available simulator, but under the academic licensing can only simulate around 20 core node and a maximum of 50 million events. Also, there was no clear way of how to do distributed simulations with it. Therefore it is not a candidate. [26]

OMNeT++

OMNeT++ is an open source C++ implementation that is widely used by researchers around the world. It is free of charge for academic and non-profit use and it is also widely documented and has a lot of pre developed models that can be used. It can also handle distributed simulations. It has a documented good random number generator. So OMNeT++ is clearly a very good candidate. [27]

Prime SSF

The version 1.0 was recently released, 24 august 2006. It is a real-time large-scale network simulator. It seems like a very interesting simulator, it has support for parallel and distributed simulation. Since it was released this recently it could be problematic to use it. Therefore Prime SSF is not an option. [28]

ModelNet

ModelNet is an emulator rather then a simulator and it works quite differently. In this you would develop your application and then test it using ModelNet as a dummy network. There is little said about efficiency, but at some point it said that you can model 100 Gnutella clients on a dual 1 GHz CPU. This makes us wonder how well it would work for us. Our clients might be simplified a lot and only act the behavior, not really be a working application. But still be programmed as a stand alone application. Also, ModelNet has the ability to simplify what it is emulating, for example remove Ethernet when only interested in TCP/IP. This makes it a candidate that has to be examined more closely. It also has some topology generators. [29]

JiST/Swans

JiST is quite different from other simulators. It runs Java code and makes use of Java Virtual Machine as a simulation engine. JiST is extremely memory efficient. They have measured that the overhead for simulating one million nodes is less then 2 GB of memory. It is also very easy to create distributed simulations. JiST has an add-on called Swans that has some pre-developed models that might be able to be used by us. Also the fact that it uses Java makes it faster to develop a model for it. [30]. However, JiST seems to be bad at collecting statistics because no information was

found about how to do it, and one presentation claims it is not good at it [37]. And therefore is not an option for us.

Summary

Ns-2 was discarded because it is too inefficient with memory. Opnet because its limitations in the student-license. Prime SSF because of the big risk related to its recent release and JiST/Swans because of its poor statistics gathering capabilities. All are serious problems and therefore they could not be used. ModelNet and OMNeT++ are still interesting after the overview and will be investigated in the following section.

2.3.3 A Closer Look at the Candidates

ModelNet Facts

There are several factors that are in favor for ModelNet, amongst all of them some are the built in topology generator, the possibility for network abstraction, and the stand alone application development possibility. But when the pros are compared to the problems with ModelNet it is found that it is no longer a candidate for this work. Some of the things found that could cause problems are that random number generation and mobility of nodes has to be implemented manually, since there does not exist such support for that for the moment. Another problem with ModelNet is that lack of assisted statistics gathering. The emulator software has a built in measurement points, but it is hard to define own measurement points. It also lacks the capability to model background traffic. [36]

But the major issue for not choosing the ModelNet emulator software is the limitations due to the fact that it is running in real time. If it does not have capacity to deal with a packet within a deadline the packet will be dropped. There is two ways a packet could be dropped, either physically before it enters the core nodes, or virtually, inside the core nodes. If packet dropping within the core nodes start to occur the results would get very strange since any packet anywhere in the virtual network might be dropped, this might result in packets being dropped on links that are not loaded at all. So, the conclusion is; ModelNet is a too big of a risk for us to use. [31]

OMNeT++ Facts

There are several reasons for choosing the OMNeT++ simulator environment together with the INET framework. OMNeT++ is flexible, scalable, and accurate. It also has a good support for random number generation. Further it has a well documented environment and a lot of resources that might be reused for this work. The simulator environment also has features as automatic network configuration and the possibility to counter the problem of warm up periods by finding a stable state before it starts to record statistics. [32]

The problems that might occur when running the simulations are that the simulator might be to slow, and that it could be to memory consuming.

Memory consumptions are at the first glance quite fair. There is records showing that about 10000 nodes takes about 325 MB[34], other records shows that about 11000, nodes where 5500 nodes are routers, takes up 1.5 GB[35]. These numbers give a hint that this simulator software has an acceptable memory usage.

The random number generator primarily used in OMNeT++ is called Mersenne Twister; it is fast and has a huge period of numbers. The random number generator has been tested in [24], and they found that it works well as long as you stay away from some seeds that are correlated. Using correlated seeds for our size of simulation is highly improbable. Since the random number generator gives a possibility to use several separate random number streams it fulfills the previously stated need for reproducibility and independent streams.

OMNeT++ has support for performing parallel and distributed simulations fairly easily. There are some things that are not allowed to do be done in the simulation model, for example global variables can not be used. [33]

Because of all the positive information found about OMNeT++ it will be used in this thesis, below follows a more detailed description of the chosen simulator.

Detailed Description of OMNeT++

OMNeT++ is an open source discrete event simulator. It builds its simulation models in two separate parts. One part is the architecture of how models are connected; this is done using a language called NED (NEtwork Description). The other part is the models themselves, they are coded in C++ using the OMNeT++ API.

A module in OMNeT++ can consist of several other modules. These modules that consist of others are called compound modules, and the other are called simple modules. A compound model can also consist of other compound modules. These modules communicate between each other using message passing.

When programming, data collection can be specified, i.e. what to collect and when to collect it. OMNeT++ will then collect the specified data during the run and store them in two files. Why there are two files is because there are two different kinds of values OMNeT++ can record; vectors and scalars. The OMNeT++ package comes with two tools for viewing the collected statistical data, Plove and Scalars. Plove is used to display the results of collected vectors and Scalars to displaying results from gathered scalars.

OMNeT++ has a graphical user interface that can be used to display simulation models. This is great when validating if a model works; because of the possibility to follow messages step by step through the system, as well as inspect their values along the way.

OMNeT++ has previously been used in a number of research papers and also used in education at universities.

2.4 Framework for Simulation

2.4.1 Telephone Traffic

The number of calls that arrives at a fixed point in the telephone network during a finite time period is called the arrival rate. The calls arrive at random and are independent of each other, thus forming a Poisson process. The Poisson process is

the mostly used process to describe the distribution of all the arriving calls. To denote the arrival rate in mathematical expressions the lambda letter (λ) is used. [18]

The Poisson process, described by the MathWorld web site, looks like following:

$$P(n) = \frac{(\lambda t)^n e^{-\lambda t}}{n!} \quad (1)$$

$P(n)$ = Probability of n arrivals in a finite interval of time
n = Number of arrivals in a finite interval of time
λ = The arrival rate
t = Average holding time

Holding time is the term used to describe the total length of a call. This includes the talking time, queuing time if any, and the time to manage setup and tear down of call. The distribution of holding times is exponential. The inter arrival time of calls is also exponentially distributed. [18]

When making a call and the system does not have the capacity to handle that call, it will be blocked. When the call is blocked the caller will receive a busy signal. Most service providers can not dimension a system so that it can handle all the subscribers at the same time. Therefore blocking will occur.

Some systems incorporate queuing as a way to let the subscribe wait for a service to become available.

The load on the telephone network is related to the time of day. The activity in the network is at its highest level in the morning when people come to work, but it decreases when the day gets close to lunch hour. In the afternoon the traffic increases slightly. The third peak seen in the traffic pattern is in the evening around 19 o'clock. From this data a Busy Hour can be defined as the 60 minutes that has the highest traffic load under a longer time period.

Figure 9, taken from [21], shows the predicted traffic demand per subscriber in a WCDMA network for the years 2006-2010. The data in the figure is an average of the demands, based on information gathered from several operators. Other information in [21] shows that the busy hour traffic share is 10 percent relative the total amount of circuit switched traffic in the network. In the figure below a subscribers traffic volume in milli-Erlang can be read on the left axis. On the right axis the Minutes of Usage (MoU) per subscriber and month can be read. Each column shows the total amount of traffic and is divided into three categories: the lower part (bluish color) shows the amount of speech traffic, the middle part (purple color) shows the amount of transparent data (not explained in this text), and lastly the upper part (green color) shows the amount of non transparent data (not explained in this text).

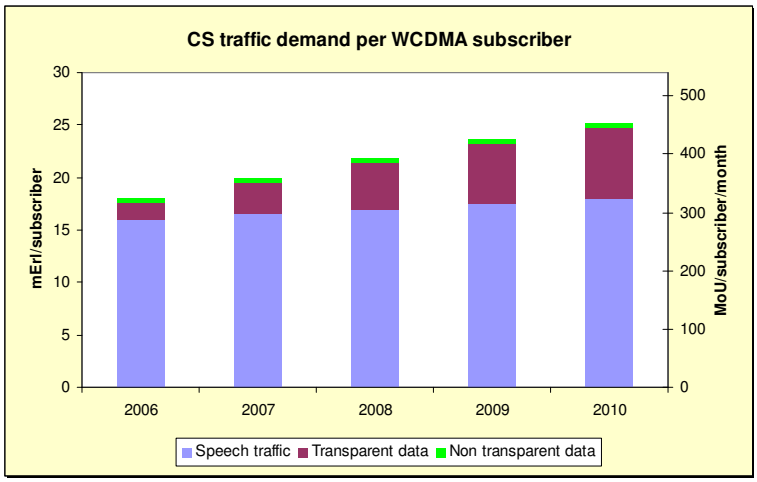

Figure 9: Circuit Switched traffic demands per WCDMA subscriber

2.4.2 Internet Traffic in the Simulator

In the simulations that where conducted in this project, the need for a realistic model is important. In order to design a proper one, the need to understand the behavior of different kinds of network traffic is important. Even though one has to make severe simplifications, the traffic generated in the simulations should be representative.

In the case of telephone traffic Poisson processes and exponential distributions are all that is needed to model telephone traffic networks in a satisfying way. But in computer networks, as the Internet, things get more complicated. Different techniques exist on the Internet for sending data thru and forth. All these differences give different traffic patterns, neither of them that could be described with the same methods as in the case of telephone traffic.

Data found in [21], and that is shown in Figure 10, shows that web traffic (WWW/HTTP) has more than a 50 percent share of the total amount of PS-related traffic in a WCDMA packet switched network. The other major application is streaming traffic. Together this two applications account for nearly 75 percent of the total traffic in the WCDMA network. To our understanding these numbers represents both user plane traffic and control plane traffic.

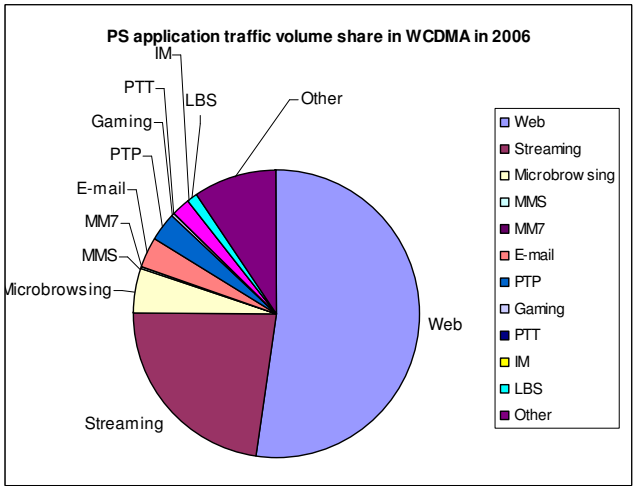

Figure 10: Traffic volume share of PS-applications in a WCDMA network

Web traffic

Surfing the World Wide Web is a very popular thing to do when using the Internet, and as stated above it accounts for the major part of the PS-related traffic seen in cellular networks. The characteristics of web traffic are that when a user visits a web site, there is not one TCP connection that is active between the web server and the user's terminal. For HTTP/1.0, defined in RFC 1945, there is one connection per object to download from the web page, e.g. a web page that contains text and five pictures results in one connection to download the requested file, and five TCP connections to download the pictures. Figure 14 illustrates the nature of web sessions, showing that each visited web page can consist of several TCP connections.

A lot of effort has been put in the area of describing the nature of web traffic. In [13] the authors find that the arrivals of web connections (WWW) can not be modeled by Poisson processes. Current research shows that web traffic has a so called self-similar nature, i.e. web traffic shows a bursty behavior over different time scales.

The characteristics of network traffic are so called self-similar. An object that is self-similar is roughly looking the same as a part of the object. An example taken from Wikipedia says that a curve is self-similar [10]. In Figure 11 the properties of self-similarity is shown. The enlargement of a part of the curve (to the right) shows that it has roughly the same look as the whole curve.

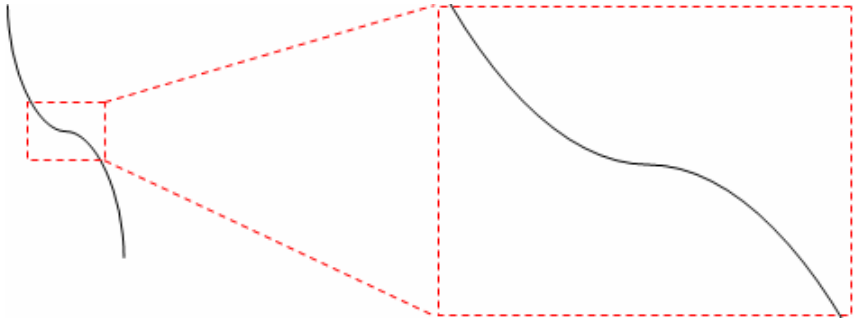

Figure 11: Properties of self-similarity, a part of the curve is similar to the whole curve

Crovella and Bestavros define self-similarity in [11], and they explain it in the following way: "A self-similar time series has the property that when aggregated (leading to a shorter time series in which each point is the sum of multiple original points) the new series has the same autocorrelation function as the original." They continue to explain possible causes for why web traffic is self-similar. They say that it depends on for example the distribution of the document sizes of web pages, effects from caching, the users "look at" time, and the superposition of several WWW transfers on the local area network.

In Figure 12, from [11], shows the self-similar behavior. The upper left diagram shows web traffic during an hour, and clearly shows the bursty behavior. The upper right figure, which is a sub set of the upper left figure, shows the characteristics of web traffic during a six minutes period which is clearly bursty. The lower left figure is in turn a sub set of the upper right figure showing the bursty behavior over a 50 seconds period. The last part of Figure 12, the lower right figure, shows a subset of the lower left figure over a period of six seconds, depicting the bursty nature of web traffic.

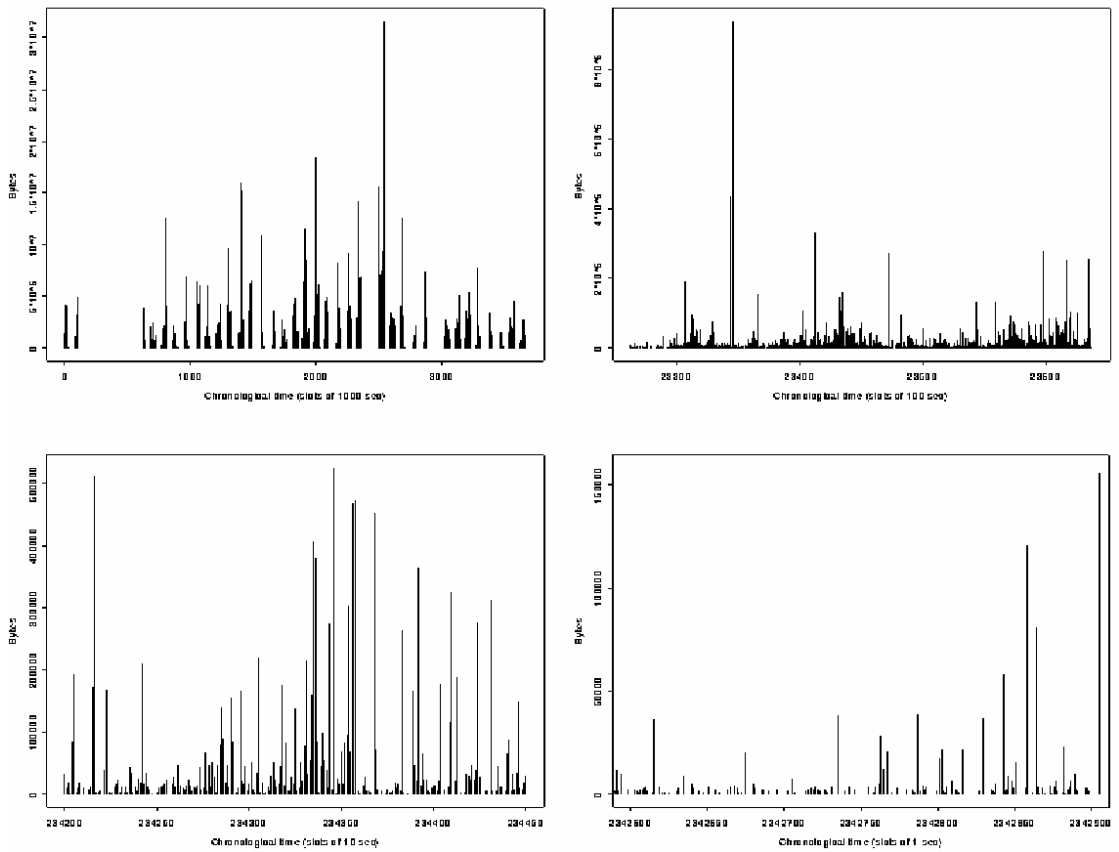

Figure 12: The bursty behavior of web traffic during different time scales

Due to this self-similar property the traffic models used to describe data traffic is not capable to describe the behavior of network traffic. Traffic in internets tends to be bursty, and the self-similar property keeps this behavior over larger periods of time, whereas traffic modeled by Poisson-processes smoothes the "curve" as more sources contributes. E. Leland et al. showed in their paper [12], which is based on four years of Ethernet traffic measurements, that "*the presence of "burstiness" across an extremely wide range of time scales: traffic "spikes" ride on longer-term "ripples", that in turn ride on still longer term "swells", etc*". So no matter how many sources there is, the nature of Ethernet traffic is bursty and aggregated Ethernet traffic normally keeps this characteristics [12]. These findings are verified in [13]. The effect of self-similarity over different time periods can be seen in Figure 12, where each subset of a period shows a similar behavior as the whole set.

These findings show that it is a highly complex matter to model the traffic on the Internet. Inter-arrival times, packet sizes, queuing, congestion, different protocols encapsulated in other protocols, and more are some of the factors behind this complex behavior.

How to Simulate Web Traffic Behavior

Several suggestions to explain the origin of self-similarity in web traffic exist. Crovella and Bestavros use the following model in [11] to explain it: Web traffic could be seen as multiple aggregated ON/OFF sources, where a source could be for instance a workstation in a LAN, either receiving data or is idle. If the distribution of ON/OFF periods for each process is heavy-tailed, then the time series will be self-similar. ON periods is characterized by WWW traffic, e.g. the transmission of data

for individual files. There are two different types of OFF periods; Active OFF time and Inactive OFF time. The Active OFF time is the OFF periods during an ON period, e.g. when a web page is downloaded and the web browser parses the information retrieved in order to start download embedded objects. These Active OFF periods is rather short, 1 millisecond to 1 second according to [11]. The Inactive OFF times is the users "think time", e.g. the user have retrieved the page and is reading the contents or looking at a picture. These time periods is assumed to be greater than 30 seconds. So to sum it; Active OFF times is machine induced, while Inactive OFF times is user generated delays.

The following figure (Figure 13), taken from [11], illustrates the different periods. To the left is the users active time, which corresponds to her web browser fetching a web page containing several embedded objects. To the right the inactive time that corresponds to the users look at time.

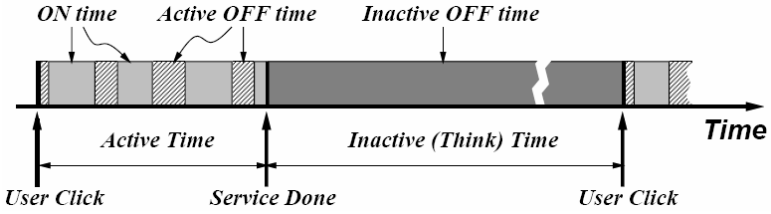

Figure 13: Differences between ON times and OFF times

A user's web session could be depicted as in Figure 14, which comes from [21]. During a session a user visits a number of different web pages (Page 1… Page N), each page containing several embedded objects creating a TCP connection for each object (TCP 1… TCP N). Figure 14 shows all the components of a WWW session, in contrast to the figure above that only shows the pattern when visiting a single page and the corresponding think time. The time between the first user click and the second user click, seen in the figure above, is the same time that is called page IAT (Inter Arrival Time) in the figure below.

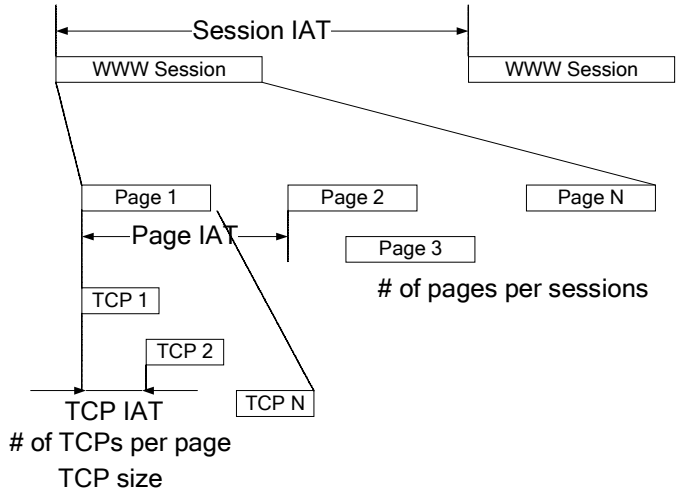

Figure 14: The components of a WWW session

To generate web traffic that corresponds to the pattern seen in Figure 14, one could think of a web session as a process that retrieves data during the ON-period and sleeps during the OFF-period. The active time in Figure 13 corresponds to Page 1, seen in Figure 14. In [20] the authors describe page sizes, request sizes, and

embedded references as things to pay attention to. Each of these could be modeled using different distributions, and should so be according to their findings.

Information about how web traffic is implemented in the simulator is found in 3.9.1 and in 4.5.2.

Streaming Traffic

Streaming music is quite popular at the Internet, and has also a rather large popularity in telecommunications networks as mentioned earlier.

A user that wants to start a streaming session, containing either video or audio, usually visits a web site that offers this service. The user clicks on a link containing a URL to the streaming server, a URL which is connected to by the user's media player. After that the media player handles the communication with the server. The media player uses the RTSP-protocol, defined in RFC 2326, to control the stream server and thus letting the user interact with the stream. The actual transportation of the data is handled by the RTP-protocol, RFC 3550. RTSP uses TCP to transfer data since the control channel is established at both ends during the whole session. RTP usually sends its data using UDP, but it could use TCP if needed but avoids it due to the TCP congestion control.

Studies done on RealAudio traffic shows that their streaming traffic pattern is different from web traffic. The behavior is that data sent seems to be constant looking over time periods ranging over tens of seconds. But if the traffic is inspected at smaller time scales the RealAudio flow have bursty ON/OFF behavior, with OFF periods of 1.8 seconds [19]. The mentioned paper also shows that an average session is roughly ten minutes.

Information regarding the implementation of streaming traffic in the simulator can be found in section 3.9.2 and in 4.5.2.

2.4.3 Different Models for Mobility Patterns

Since this work focus on the behavior of the system from a mobility perspective, it is necessary to model the movement of users and their terminal entities. The following section presents a subset of the existing mobility models that try to model this behavior. There exist a lot of mobility models, all trying to address different issues regarding mobility. But it has also been shown that none of the models is really that good, as shall be seen in later sections.

In order to solve the issue with moving hosts/mobility in the network, a need to use a mobility model is necessary. This model will tell the simulation environment how hosts move in the network. The models are based on mathematical theories and assumptions about the scenario they are used in. The problem with the available models is that they are made to fit a specific scenario. There exist models that model every movement independently, and others that account for group behavior. Different models handle hosts that reach the boundary of the simulation area differently.

Figure 15 shows some of the existing mobility models, and how they partitioned based on their properties. The following sections will describe the characteristics of random mobility models, as seen in the branch of Random Models below, and the

Gauss-Markov model. The provided text should not be considered a thoroughly description of these models, but a brief overview. The effects of using mobility models will also be discussed.

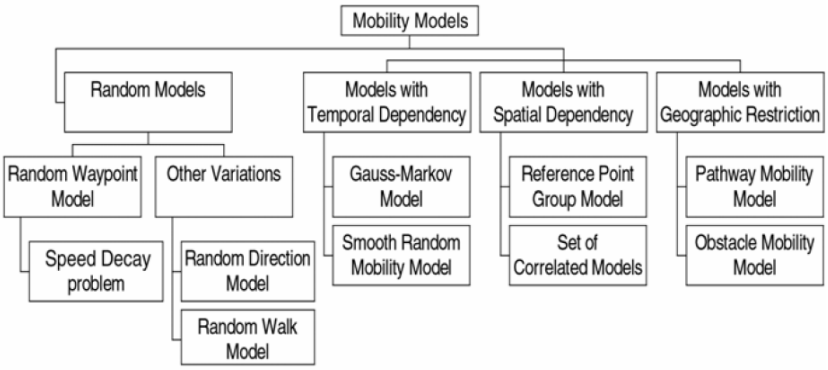

Figure 15: Different mobility models and their correlation according to Bai et al.

Random Waypoint Mobility Model

In the Random Waypoint Model (RWP) a node (mobile host, mobile entity) chooses a point in the simulation area that it moves toward, the node then randomly selects a movement speed from a uniform distribution [*minspeed, maxspeed*]. The direction of the movement is not changed and the velocity is constant during the travel. When the node reaches the waypoint it pauses there for a random amount of time. The node then repeats the process, and continues to move during the whole simulation process. [2]

In Figure 16, from [2], the result of a node moving according to the RWP model is depicted. The mobile host starts at a random position in the model, in this case (133, 180), and lets the mobility model steer its direction and velocity. The dots show where the mobile host stops and pauses, and then makes a new waypoint and speed decision.

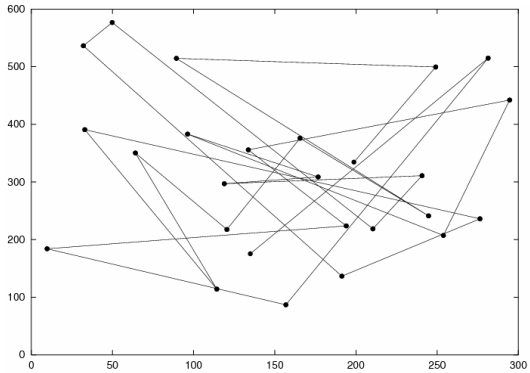

Figure 16: Movement pattern of a node using the Random Waypoint Mobility Model

The main components that affect the mobility model are the maximum speed v_{max} and the pause time T_{pause}. If the value of maximum speed is kept low and the pause time is quite large, then a stable simulation is achieved e.g. the topology of the model does not change that much. But if the speed is large and the pause time is small the topology will be quite dynamic. [1]

One of the problems with this model is discussed in the paper from Bai et al. [1]. The problem is due to the spatial node distribution, and how a uniform node

distribution changes to a non-uniform node distribution during the simulation time. After a long time the distribution reaches a stable state and the concentration of nodes is in the center of the simulation area, whereas the node density is close to zero at the borders of the simulation area. Another problem with the mobility model is that the average number of node neighbors varies over time. This is due to node movements that occur through the center of the area or towards the area center. This phenomenon is called a *density wave*. [1]

Bettstetter et al. showed in their work the reason for this behavior. They found that the probability to make a travel that is directed towards a border is only 12.5% and that the mobile host moves towards the center of the area with a probability of 61.4% [3].

Random Walk Mobility Model

The Random Walk Model (RWK) is sometimes referred to as *Brownian Motion*. This is due to the similarities in the models regarding the unpredictable movement of entities in the model [1]. In RWK the mobile hosts change their speed and direction at every time interval *t*, or after a certain distance *d*. To determine the new speed a value is chosen from [*speedmin, speedmax*], and a direction is chosen from [0*,* 2π]. If a node reaches a border of the simulation area, the node bounces and continues in a direction that is a function of the incoming angle. [2]

In Figure 17, taken from [2], the movement of a mobile host using the RWK mobility model is shown. The host starts in the center of the area and moves with respect of the model. The dots show where the mobile host makes a new speed decision and direction decision.

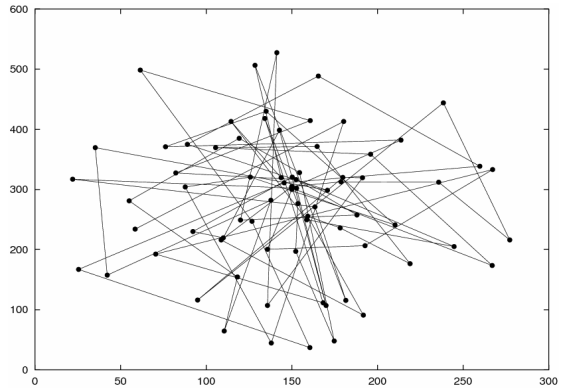

Figure 17: Movement pattern of a node using the Random Walk Mobility Model

The problem with this model is that if the predefined distance *d* or the time interval *t* is set small, the area that the node roams will be rather small. This leads to node that only moves in a small sub area of the simulation model [2]. Another problem with the RWK model is that due to the lack of memory in the function the behavior of a mobile node is rather unrealistic. [4]

Random Direction Mobility Model

In the Random Direction Mobility Model (RDM) a node chooses a direction which to follow during its travel. This is similar to the RWK model, but in RDM a node continues its travel until it reaches a border. When a node reaches a border it sleeps

for a random amount of time and then chooses a new direction. The direction is chosen from the interval [0°, 360°]. [2]

The pattern of a node moving using the RDM model is showed in Figure 18, which comes from [2]. In the figure a mobile hosts starts its travel in the center of the simulation area. The dots at the borders depict a node that have reached the border, sleeps, and then makes a new direction choice.

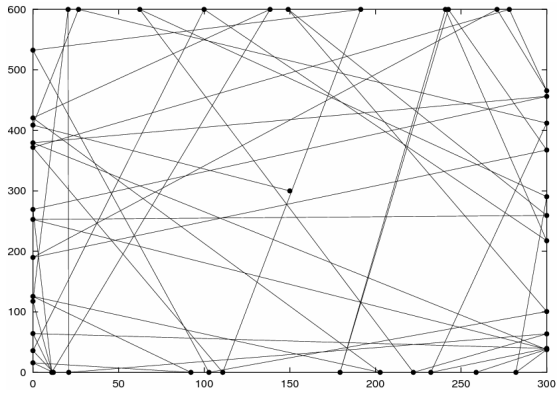

Figure 18: Movement pattern of a node using the Random Direction Mobility Model

The RDM model was developed to address the problems found in the RWP model with density waves and the non-uniform distribution of nodes. In RDM the number of neighbors is kept at a nearly constant number throughout the simulation. One problem with this model is that mobile hosts reside at the borders of the simulation area. [2]

Gauss-Markov Mobility Model

The Gauss-Markov Mobility Model (GM) tries to affect the future movement by considering the current velocity and direction. The GM model has velocity dependence, at each time interval *t* the new speed and new direction v_t^x and v_t^y is calculated as follows:

$$\begin{aligned} v_t^x &= \alpha\, v_{t-1}^x + (1-\alpha)\upsilon^x + \sigma^x\sqrt{(1-\alpha^2)}\,w_{t-1}^x \\ v_t^y &= \alpha\, v_{t-1}^y + (1-\alpha)\upsilon^y + \sigma^y\sqrt{(1-\alpha^2)}\,w_{t-1}^y \end{aligned} \quad (2)$$

α is a tuning parameter that varies the randomness of the functions, where $0 \leq \alpha \leq 1$. The mean value of the speed and direction is denoted by υ^x and υ^y. The previous speed and direction is denoted by v_{t-1}^x and v_{t-1}^y. The w_{t-1}^x and w_{t-1}^y are random values with a Gaussian distribution, that has a mean equal to zero and a variance of σ^2. σ^x and σ^y represents the standard deviation. [1]

The parameter α shows how much memory the function has. If α is equal to zero, you will get a memory less function according to Bai et al. The memory less function is the same as the Random Walk Mobility Model. The function would then look like this:

$$v_t^x = \upsilon^x + \sigma^x w_{t-1}^x$$
$$v_t^y = \upsilon^y + \sigma^y w_{t-1}^y \quad (3)$$

Going the other way and letting α take a value of 1, the function becomes linear. It only depends on the previous speed and previous direction. It would then look like this:

$$v_t^x = \alpha\, v_{t-1}^x$$
$$v_t^y = \alpha\, v_{t-1}^y \quad (4)$$

But letting the value of α be varied between 0 and 1 ($0 < \alpha < 1$) then tells how much memory the function has. An increased value of α makes the function more influenced by the previous values. Whereas a low value of α makes the function less dependent on its previous speed, and its previous direction. [1]

There exist different implementations of the GM model. One specific implementation tries to avoid letting the hosts reside near the border of the simulation area for a longer time. This is achieved by modifying the direction of the node if it is to close to a border, or if it has reached a border. The system have predefined values to change the direction to depending on which of the area borders the mobile host are getting close to. [2]

Figure 19 shows the movement of a node using the GM model. The mobile host begins its travel in the center of the model. In the figure taken from [2] the values needed the produce this walk are; t is 1 second, α is 0.75, w_{t-1}^x and w_{t-1}^y are both chosen from random Gaussian distribution with mean that is zero and the standard deviation is one, υ^x is static and has been chosen to 10 m/s, and the initial value of υ^y is 90°.

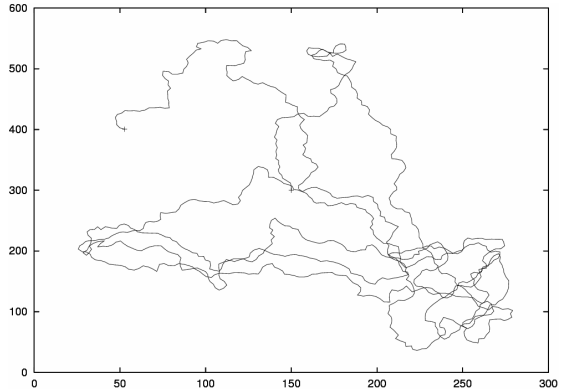

Figure 19: Movement pattern of a node using the Gauss-Markov Mobility Model

A Probabilistic version of Random Walk

The Probabilistic version of Random Walk [9] uses a probability matrix to determine whether a mobile host should move or not. The probability can be changed for each node by modifying *P*. The probability matrix used looks like:

$$P = \begin{bmatrix} P(0,0) & P(0,1) & P(0,2) \\ P(1,0) & P(1,1) & P(1,2) \\ P(2,0) & P(2,1) & P(2,2) \end{bmatrix} \quad (5)$$

Where state 0 is the current position, state 1 is the previous position, and state 2 is next position for the mobile node. This model is said to produce a more realistic movement pattern, if the probabilities is chosen correct, as a person does not tend to randomly walk around until they reaches their destination. In Figure 20 the movement pattern produced by a node using the probabilistic model is depicted, the figure is taken from [22].

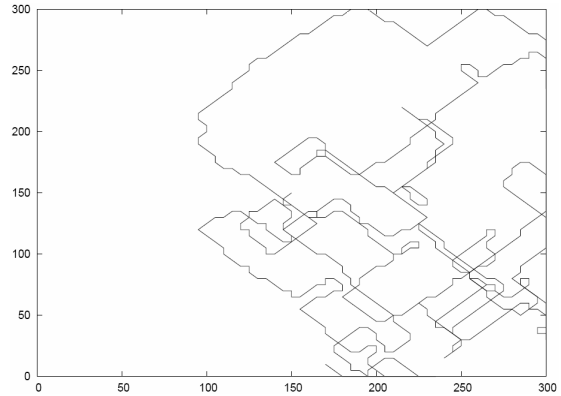

Figure 20: A nodes movement using the Probabilistic version of Random Walk

2.4.4 Problems with the Existing Mobility Models

Existing random models all try to address the issue of mobile hosts, and how they should roam in a simulation area. All these models are very simple models of how mobile hosts move in the real world, but according to Bai et al. in [1] all the random models fail to model some essential characteristics that would be found in a mobile network. They say that the random models generate extreme mobility behavior due to their memory less properties. This is since the speed and direction does not depend on the previous speed and direction, as seen above. The random models also fail in the aspect of spatial dependency, since all nodes are model independently of each other, hence lacking group behavior. There exist cases when the mobility of a group of nodes shows correlated mobility. The third limitation that Bai et al. describes is the lack of geographic restrictions. The random models can not model movements in an urban area, where the limitations of waypoints or directions are caused by buildings, crossings, and freeways.

Other research done in the area of routing in mobile ad hoc networks (MANETs) has looked at the impact of using different mobility models. One of these is the IMPORTANT project [5], that is an evaluation framework for the study of "Impact of Mobility Patterns On RouTing in Ad-hoc NeTworks". The question of their project was; *if mobility matters*, and *how much it does matter*. They performed simulations in ns-2, comparing the performance of three different ad hoc routing protocols (DSR, AODV, and DSDV). For each of the protocols they used four different mobility patterns, namely: Random Waypoint, Reference Point Group Model (RPGM), Freeway, and Manhattan Grid. For the RPGM model they used two versions, one with single groups and one with multiple groups. Some of their results are shown in Figure 21 and Figure 22. In Figure 21 the routing throughput is a

function of the maximum speed, and the figure clearly shows that the DSR-routing protocol has a really poor throughput when using the Manhattan mobility model and the speed is increased. It is also seen in the figure that the throughput decreases some when using the Random Waypoint model and the Freeway model, whereas the RPGM model seems to have no effect on the throughput. In Figure 22 the routing overhead is a function of the maximum speed, and the routing overhead increases severely for the DSR-protocol when nodes travel at higher speeds and using the Manhattan model. The Freeway model and the Random Waypoint model also affect the routing protocols performance, whereas the Reference Point Group Model does not affect the performance of the DSR-protocol.

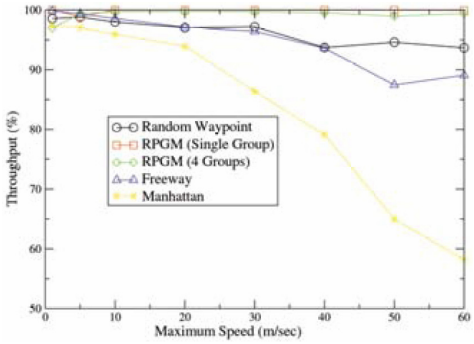

Figure 21: Throughput in a simulation using different mobility models and the DSR routing protocol

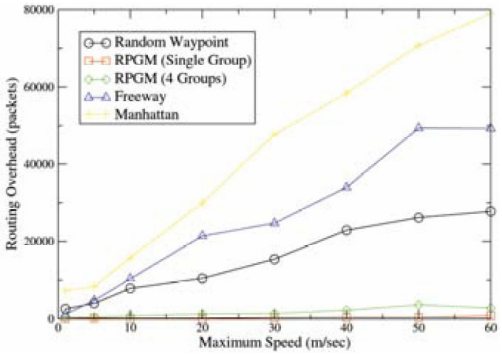

Figure 22: Routing overhead experienced using different mobility models and the DSR routing protocol

Another test they performed was to evaluate the routing protocols relative to each other using a specific mobility model. In Figure 23, where throughput is a function of maximum speed, it is seen that the AODV-protocol has the best throughput when using the Manhattan mobility model and the DSR-protocol performs rather bad. In Figure 24, where throughput is a function of maximum speed, the Random Waypoint mobility model is used. It is seen that DSR is the best performing protocol, whereas AODV experiences a smaller throughput, and DSDV performs rather bad.

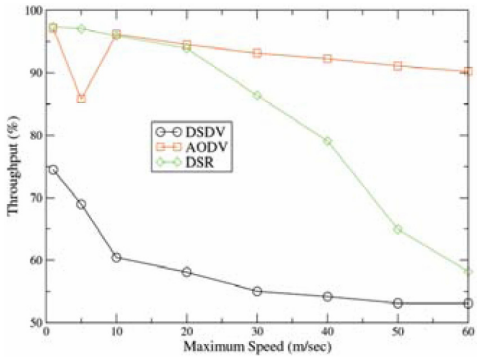

Figure 23: Throughput performance of routing protocols using Manhattan model

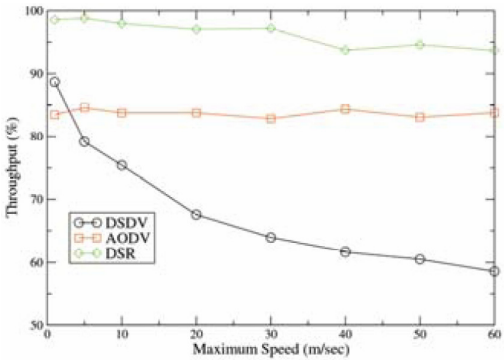

Figure 24: Throughput performance of routing protocols using Random Waypoint

Another problem that is experienced in the Random Waypoint model is the so called *Speed Decay* problem. This problem is observed in most of the mobility models according to Yoon et al. in [7], and causes the average speed to decay and converge to a long-term average that is lower than the initial average speed. This could result

in erroneous data, if the researcher using these models is not aware of the problem, wrong conclusions might be drawn from the result.

In Figure 25, taken from [8], the speed decay is shown as the average speed decreases as simulation time goes. Yoon et al. showed the results both in [7] and in [8]. In the figure below the results is from a simulation using the Random Waypoint with zero pause time and a speed distribution in the interval [0, 20] m/s. They also show that average speed will drop to zero over a longer time period. The solution proposed by Yoon et al. is that the minimum speed should not be set to 0 m/s, because this is reason that makes the simulation not to reach a stable state. They tried to set the speed interval to [1, 19] m/s, and find that the average speed soon reaches a stable state of 6.11 m/s.

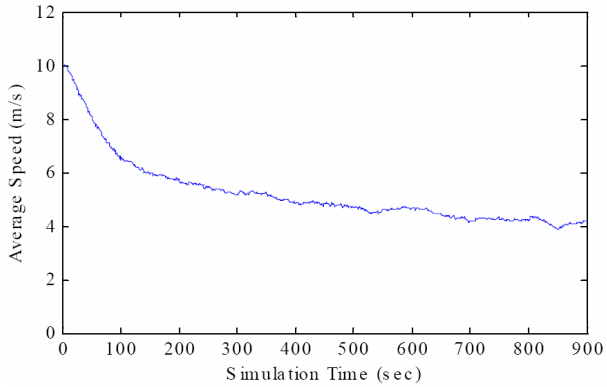

Figure 25: The effect on the average speed due to speed decay

2.4.5 Conclusions Regarding Mobility Models

After looking at these mobility models and seeing the massive critique of them, it is hard to know which mobility model to use. It is shown that almost all of the models suffer from some kind of problem, and that they all lack in modeling the real world accurately. How will the lack of sufficient mobility models affect the data gathered during simulations? How viable will the results be when examining the mobility aspect when no good mobility scenario exists?

It has been decided that the Random Walk Mobility Model will be used in this thesis work. Even tough the drawbacks of the model, it is widely used in research community, and that is the motivation for using it here. The speed interval that is going to be used during the simulations should be set so that the speed decay problem is avoided, and so that the average speed converges to a sufficient value. In the Random Walk model there is the possibility to affect the interval used for calculating how far to travel between each new direction decision. This gives that the entities moving in the simulation area are likely to cover most of the area, and not tending to stay in the middle of the simulation area as described in the section about Random Walk Mobility Model.

Since the performance of naming systems is to be examined the unrealistic movement pattern gotten from the Random Walk model may not be a problem. This is due to that the main issue during the evaluation is that there are a lot of lookups and updates. So, if a client moves using a non deterministic pattern instead of moving in predefined and realistic paths, the effect from updates sent to the naming system will still be valid.

Further information regarding the mobility model is found in sections 3.9.4 and 4.5.3.

2.4.6 Topology Generation Considerations

The Internet is a large network that consists of numerous inter-connected networks. When investigating the structure of the Internet you could divide it into several layers: at the highest level service providers offers a world wide communication network that Internet Service Providers (ISP) can connect to. The ISP in turn provides Internet access to companies, that in turn have their own networks, or to end-users. This is a very simplified and limited description of the Internet, but it gives an idea of its structure, which is depicted in Figure 26.

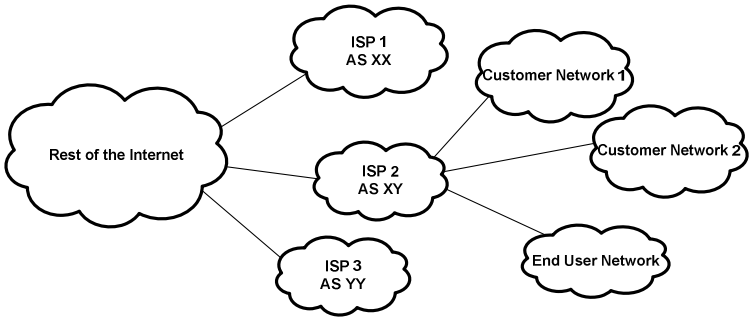

Figure 26: The fundamental characteristics of inter-connected networks

Each cloud in the figure above represents a network, which could be of any size. The ISP networks are often Wide Area Networks (WAN), covering a whole country or even larger. A customer network could either be a single Local Area Network (LAN), or several inter-connected LANs. A network of an ISP could be organized so that every major city has a couple of larger routers covering the city, whereas smaller cities could have only one router. A customer network is often connected to an ISP with several links, thus getting redundancy. Figure 27 shows a simplified picture of an ISP network, only depicting routers. The network consists of border routers at the edge of the cloud, which handles connections to and from the ISPs peers. The routers inside the cloud are the core of their network, handling routing inside the network. The border routers are often connected to a large number of customers or other ISPs, and should be able to handle large amounts of traffic.

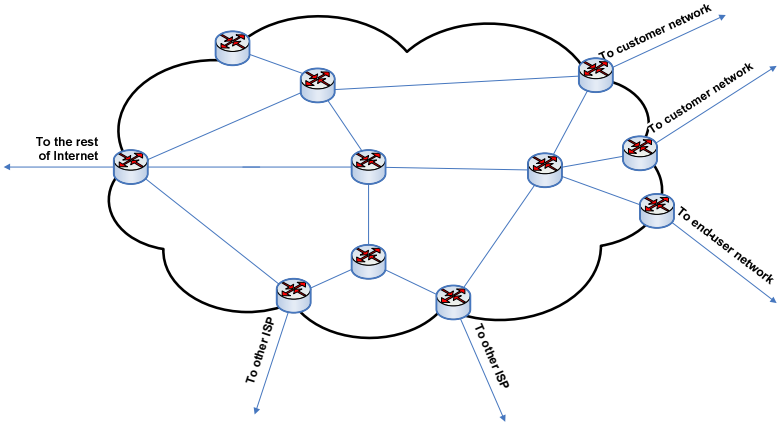

Figure 27: A simplified view of the routing structure inside an ISP network

In order to create a realistic topology there exist a number of helpful tools. Following is a brief description of some of them, namely: BRITE, PSgen, GT-ITM, Inet.

BRITE implements one method for creating the topology, but dependent on the configuration of the parameters the behavior differs. One configuration of the parameters constructs a topology according to the Waxman method; that interconnect nodes based on their relative distance in the simulation area. Another configuration could yield a model where the network is constructed incrementally, i.e. a top down approach, and the nodes interconnect towards nodes that have a higher degree [16].

PSGen is based on the BRITE topology generator, and has explicit support for OMNeT++. It has all the capabilities that BRITE has, according to the limited documentation found on the Internet [17].

GT-ITM generates a transit-stub model which consist of three levels of hierarchy; transit domains, stub domains, and LANs attached to stub nodes, according to the authors of [14]. The method creates the graph in steps, in each step creating a new domain. When creating the model it is possible to adjust a set of parameters to control the average number of created nodes in the different domains, it is also possible to adjust the parameters controlling the connectivity between domains and between nodes. In Figure 28, taken from [14], a typical transit-stub network is showed. It shows the transition domains in the grayed areas, stub domains are in the white circles, but the figure does no depict the LANs that are attached to the stub nodes.

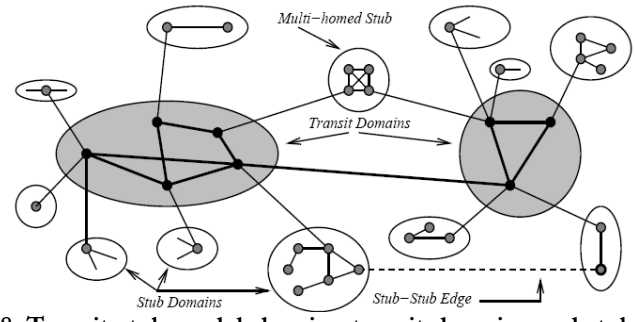

Figure 28: Transit-stub model showing transit-domains and stub-domains

The Inet-topology generator [15] tries to create a topology by for each node connecting it with *n* other nodes, where *n* is taken from a distribution that has power-law properties. The generator then continues by connecting different nodes with each other according to a specific rule.

Further information regarding the topology and how it is constructed can be found in sections 3.9.5 and 4.5.1.

2.5 Distribution Aspects of the RLR

In the Path Couple method the RLR has a central role in the naming system. During the work of this thesis the distribution aspect of the RLR will be investigated. The functions of the RLR could be implemented in a modified DNS fashion, a DHT-based method could be used, or another method that could offer the wanted

features. The reason for wanting to represent the RLR in this way is to spread the load on the root in the Path-Coupled model, since the RLR will receive a lot of queries as described in section 2.2.3. The reason for distributing the RLR in one of the above mentioned ways is several, namely; load balance issues, lookup times, redundancy, and so on. All the features that are regarded as necessary in today's naming systems, to ensure a reliable performance.

2.6 Summary

In this section it is shown what the scalability aspects of this thesis are. It is also shown what is measured in this thesis, which is how the system is loaded in different aspects while having an increasing amount of users. The three name-to-address resolution methods, DNS, DHT-based implemented using Chord, and the Path-Coupled method, were presented, so that that their behavior could be understood.

One simulation framework was chosen as being used for this project after reviewing several alternatives and found that the chosen one, OMNeT++, had several nice features that would cover the needs of this thesis. Several modules for the simulator were investigated, them being: traffic generation patterns, mobility models and topology generation. The parameters needed to create a traffic generation pattern was found and presented. Mobility models were all found to be suffering from the same problems and therefore random walk was selected because of its ease of implementation, and also because it has become widely used within the research community.. Concerning topologies it was chosen to not use any topology generator because of their lack of realism, instead a simplified model of a real network topology was chosen to be used.

Last the RLR distribution aspect was introduced and it was seen that the RLR in the Path-Coupled method has to be distributed somehow and this will be investigated in this thesis.

3 Method

This part of the document provides the reader with information about the evaluation of the project and how it is done. This section also describes some of the design choices made for the future implementation.

3.1 Introduction

Throughout the work of this thesis some limitations is imposed, which is due to the work done in the previous thesis. The reason for inheriting these limitations are the advantages gained from reusing the design in the previous. Some of the limitations on the DNS method is for instance the TTL-values that are being used, the iterative lookup method, and the use of a local resolver that is separated from the client. In the Chord-based method the limitations imposed are for instance the methods used when setting up the ring and the partitioning of the key space. In the Path-Coupled method there are no limitations inherited since the method is not based on previous code.

3.2 Network

When looking at the network it is very important to look into what sort of load the method puts on it, if there is any area that will be heavily loaded and how the lookup traffic relates to the effective traffic put through the system.

When measuring a network concerning scalability is not about measuring the network per say, but instead measuring if there is any node in the system that will be heavily loaded, or any specific area, so that they would not be placed to close to each other, since it would give cause for congestion if the traffic volumes did not scale well. It is also estimated how the ratio between traffic related to the naming system is in comparison to amount of traffic for the user data sent and received at a host. This is done by measuring lookups and updates done by the clients, also the lookup traffic is split into different categories: Web, telephone and streaming lookups. This makes it possible to calculate what the relation between effective traffic and lookup traffic is.

3.3 Measurement of CPU and Memory Usage

In this thesis measurements of how the CPU and Memory usage scales is performed. This is done by forming an expression for it, like it is explained in section 3.2. By doing this the measurement is independent from the implementation and more general. This is measured by giving how many requests a CPU would have to process during one busy hour, and by looking at the total number of requests a server will process during the 24 hours of a simulation and creating a medium of processing events per second.

The memory needed is calculated at many points in the system. First thing to think about is how much memory a server needs when storing its own records and also the cached records, secondly look on the memory demand to have pending lookups in the system at some nodes like servers or border routers.

3.4 Evaluation Process

There are several key factors that need to be looked at closer during the evaluation process. This section will describe all these factors. All measurements are done under

the same circumstances during 24 hours of network time. The traffic during the evaluation process is based on peak traffic, i.e. busy hour data described in section 3.9. The reason for choosing to simulate 24 hours of network traffic is to eliminate the effects of the so called warm up period. The warm up is the time period before the simulation reaches a stable state. This time period should be as small as possible compared to the total simulation time, so that its effect on the result obtained from the simulation could be disregarded.

Updates

Due to client movements in the system there will be a lot of updates that are sent to the naming system. When a client connects to a new access point (see section 3.5.1) it gets a new address, but keeps its symbolic name. This new information is sent to the naming system for it to update its name-to-address mapping. So, the performance of the naming system during the high load due to updates must be evaluated, and this has to be done at the different levels in the hierarchical methods.

Lookup Requests

Clients will perform lookup requests based on their traffic pattern. To investigate the effect on the system due to the lookups two things will be looked at: The number of lookups in the naming system and their effect on the naming system as a whole. The second issue that needs to be evaluated is: What effect the number of lookups has on different levels in the hierarchical methods.

Memory Demands

There are three aspects to consider when evaluating the memory demands. First of all is the needed memory to save all the information in the naming system. When looking at the DNS solution and the Path-Coupled one; what are the memory demands at the different levels compared to the total memory demand? The second aspect to consider is the amount of memory needed by enabling caching in the system. Last, but not least, mainly related to the Path-Coupled method; how is the memory consumption affected on the routers during the lookup/connection phase?

3.5 Common Design

This section presents design of the simulation environment that include common components and components specific for each of the name-to-address resolution methods to be evaluated, i.e. DNS, the DHT-based method, and the Path-Coupled method. For each of the methods, a design plan is described, simulator components and possible design choices.

3.5.1 Common Components in the Simulator

Clients

Clients will send lookups and updates to the naming system. A client should be able to send a lookup request to the naming system based on its traffic profile. A client should also inform the naming system with an update-request when its name-to-address mapping has been changed due to movements in the network topology. Measurements on clients will be performed in the same manner as described previously in section 3.4.

Access Points

The access point in the naming system simulation environment works as a hybrid between a client and a traditional wireless access point, which acts as the entry point for all communication to the network and the naming system. The reason the access point was developed is the limitations in the simulation software concerning routing tables' connectivity, but it also had positive side effects like saving memory. The clients keep track of which access point they are connected to, and they make their lookups and updates through them. For more information see section 4.1.

Database

The database is a stand alone globally shared repository for storing information critical to the operation of the simulation, for more information se section 4.1.

3.5.2 Common Design choices

The placement of clients in the simulation area is done with a uniform distribution; all clients are evenly distributed across the simulation area.

3.6 DNS

3.6.1 Design Plan

The DNS system needs to look as much like the real one as possible. Therefore a local DNS server was placed in every AS. These DNS were not only used to handle outgoing request but also to provide request processing for registered records in the AS. The higher level of hierarchy is placed higher up in the topology with the root server at the very center of the whole topology, equally ranged from everyone.

The DNS method is implemented using the same method as in the previous thesis, an iterative model (as shown in Figure 2) where the local DNS server does all the request processing and just delivers back the correct result to the access point of the client where it is then deleted. It is sent back to the access point that initiated it even if the client has moved; this does not cause any problems because it does not affect our measurements.

3.6.2 Components in the Simulator

The DNS system uses the three common components as well as one unique; a name server. Clients come in contact with their local DNS system through the access point they are currently under the control off. The name servers then take care of the lookup to make it follow the given method.

Name Server

The name server's functionality will be to answer requests (lookups) for name resolution, and for updating stored name-to-address mappings. In the simulation there will be three different kinds of name servers for DNS. The Root will just have access to information about the level under it, and it will never have to update this information because the system is static and stable throughout the simulations. The second kind of servers represents the middle layer of the tree structure in DNS. They have access to the information concerning the lowest level servers, their addresses and areas of responsibilities. The third kind of servers is the lowest level servers also know as local name servers. They have access to the information about what names should be resolved to an address, if that name is in their domain. The servers also

receive updates and store them in the system. Clients also use these servers as their local name servers, and so these server perform the iterative lookup for the names supplied.

3.6.3 Design Choices

DNS is going to be implemented with the ability to turn on and off caching. If cache is used it is with 100/200/300 seconds caching as recommended in the previous thesis. The numbers refers to the different levels of the DNS system, ranging from the lowest till the highest. So for example, the address to the second layers (.com, .org …) will be stored for 300 seconds.

When an iterative lookup is processed there has to be some state information stored in the system, otherwise the lookup can not be carried out in a correct manner. Since the simulator is a system where it is known that there will not be any lost packets makes it possible to store the state information in the messages themselves.

All the servers use one common shared repository, the database, to store all information about names and addresses. To keep the functionality as it is intended, the servers restrict themselves to not access information they should not have the right to.

3.7 Chord Based Name to Address Resolution

3.7.1 Design Plan

In order for the simulations to be comparable the Chord naming system is separated from the underlying network, which also was done in the previous work.

When a client wants to initiate a connection to a resource or another client in the network topology a lookup query is sent to the naming system. The Chord naming system takes the appropriate action and supplies the client with information about the queried name. When a client moves in the network it contacts its access point in order for the system to realize that a change has occurred and that the naming system needs to be updated.

3.7.2 Components in the Simulator

The implementation has been divided in four different entities; a Chord node, the client, access points, and a system database. The client, access point, and the system database are explained in section 3.5.1. A Chord node is able to be set up as a part of a naming system based on the Chord protocol, and it is able to communicate with other nodes according to the Chord protocol. The client node is able to communicate with the Chord naming system by signaling the nearest access point telling it to issue lookups or updates. The access points works as points of attachment for the clients, and is responsible for that client's can communicate with the naming system. The system database keeps track of a client's coordinates and the ID of its corresponding access point.

Chord Nodes

The Chord node must be able to receive update messages and lookup requests. When receiving a request to make a name-to-address lookup from a client, the Chord node must forward the request to the Chord node that is responsible for that

key according to the Chord protocol. When the node that is responsible for that key is reached the address that is coupled with the key is sent back to the Chord node that received the request from the client. That Chord node is responsible for that the answer is sent back to the client. If a Chord node receives an update messages from a client, it should also be forwarded the node that is responsible for that key. A Chord node that receives an update that it is responsible for should take appropriate action to ensure that the functionality of the naming system is preserved.

Features that should be supported by a Chord node are to set up the ring and initialize its finger table and successor list. A Chord node should also be able to route messages according to the Chord protocol, with both recursive support and iterative support. A Chord node should keep a local database that contains name-to-address mappings for the keys that the node is responsible for, thus not utilizing the global system database for this purpose. With this database a node should be able to resolve name-to-address queries, and through updates keep the database consistent.

3.7.3 Design Choices

The Chord lookup operations are done in a recursive fashion as shown in Figure 5. Caching is implemented according the "send it to the predecessor"-method described last in section 2.2.2. The system is to be considered stable in these evaluations so there is no need to implement the join and leave features, as well as the stabilize and *fix_finger* functions. The predecessor pointer, finger table, and the successor list are initialized at start time. These implementations are the same as the work done in the previous study.

Each Chord node is responsible for a fixed number of keys during the simulations, where each responsible area is equally distanced from each other. The number of keys that each Chord node is responsible for is the ratio of the key space and the number of clients in the system. The length of the successor list and the size of the finger table at each node are equal to the number of bits in the key identifier space.

The number of Chord nodes in the simulations will be equal to a power of 2, i.e. 2^7, which yields ring of size 128 Chord nodes, with the exponent ranging from 5 to 6. The key space handled by the ring is dependent on the number of clients in the simulation. If there are 4000 clients, the key space will be 4096 keys which yields that each node is responsible for 4096/128 = 32 keys and that the size of the finger table will be $\log_2 4096 = 12$ entries.

Caching in the system is implemented in the same way as in the previous thesis (see section 2.3.9 in [23]), i.e. a copy of a record is sent to a nodes predecessor every time changes occur, which is described last in section 2.2.2. This way there is no need to set TTLs on the cached records because a nodes cache will always contain the correct information. The caching in the system can be turned on or off for each simulation.

3.8 Path-Coupled Name to Address Resolution

The Path-Coupled method is very different from the others and therefore has very different demands on the simulation environment and components when implementing.

3.8.1 Design Plan

The Path-Coupled system does not only require the servers to be placed in a structured way; therefore it is very important how name servers are placed in the system. All servers will use the same implementation but with the ability to be parameterized to achieve their variety.

In every network there is a border router which handles requests coming into the network. The ability to route towards a network has to exist. This is very tricky in a simulator environment that automatically assigns the addresses to everything. However, a solution was found and implemented using the dummy nodes; more information can be found in section below.

3.8.2 Components in the Simulator

The common components used for the Path-Coupled implementation is the three previously mentioned namely: client, access point and database. Additionally three other components will be used, the Name Server, Router and the Dummy Node components will be implemented and used, below follows a description of them.

Name Server (RLR/RS)

There are three different kinds of name servers, RLR, RS of level 1 and RS of level 2. All of them share the same basic functions; processing lookups and updates. The lookups are processed by looking up the sought information in the database and sending it back. All servers share the same global repository, the system database, to store this information, but they do not access information they should not have access to. This saves memory however, it causes some problems concerning updates and consistency, for more information about this and other issues concerning the Path-Coupled lookups and update see section 4.4.

Router

In the Path-Coupled method the routers have to issue a lookup to a RS to know where to route the packet, therefore the routers in the simulation is modified. They are modified to perform this lookup to the correct server in charge of their area. After receiving the information it will reroute the packet to where it is specified. At the setup phase of a simulation they contact the dummy nodes responsible for their area to have addresses added to their interface table. More information on this is found in section 4.4.

Dummy Node

The dummy nodes are the component that makes the ordinary network being able to work as a network with area addresses. This was needed because a normal node can not have several network addresses, it also helps the routing table creation of the routers in the network remain automatic. They are contacted by routers and access points at the initiation simulator to configure them. More information about them can be found in section 4.4.

3.8.3 Design Choices

The Path-Coupled implementation will be restricted to work with only three hierarchical levels. There is also some restricts to the namespace, and therefore the structure of the system also gets limited. The valid names ranges from a.0.<any name> to Z.9.<any name>.

3.9 Traffic and Movement models

In this sub section the rate at which different traffic originates will be looked at and calculated. The data found when reading the literature are mean values and has to be converted to rates that can be used in the simulations. The formula (6) is used during the calculations of mean values, also called the expected value, of an exponentially distributed variable.
An example of a conversion, using the equation below (6), is: If a user has two web sessions per hour in mean, the rate of which web sessions occur is one each half an hour.

$$\mathrm{E}[X] = \lambda_{mean} = \frac{1}{\lambda} \Rightarrow \lambda = \frac{1}{\lambda_{mean}} \quad (6)$$

All the traffic in the simulations will be modeled as a Poisson process with the rate λ.

This sub section also contains information and calculations for the amount of traffic generated during the simulations. For all the data rates the overhead of TCP/UDP is added and also the overhead from the IP protocol. The overhead from using UDP is 8 bytes according to RFC 768, TCP gives a overhead of 20 bytes plus and optional header of 12 bytes according to RFC 761. An IPv4 header, with no options, gives a 20 byte overhead as stated in RFC 760. The optional TCP header will not be used in these calculations.

To calculate the amount of traffic seen in the simulations several figures is needed. The number of lookups and updates is given from the simulation results; the sizes of packets sent to and from the naming system are found in section 5.3.2, and the overhead from UDP/TCP and IP is given above. When calculating the user plane traffic the number of lookups is also needed, together with web page sizes (in the case of web traffic) or data rates (streaming traffic and telephone traffic).

The total amount of traffic seen in the network is the sum of control plane traffic and user plane traffic, that is:

(Total amount of traffic) = (Control plane traffic) + (User plane traffic) [bytes]

The relative amount of lookup traffic is the ratio of the total amount of lookup traffic divided with the total amount of traffic, i.e.:

(Relative amount of lookup traffic) =
(Lookup traffic including overhead) / (Total amount of traffic)

The relative amount of update traffic is the ratio of the total amount of update traffic divided with the total amount of traffic as shown below:

(Relative amount of update traffic) =
(Update traffic including overhead) / (Total amount of traffic)

These three expressions are valid for all three different traffic types. The following sections show how the control plane traffic and user plane traffic is calculated for each traffic type.

3.9.1 Web Traffic

In order to get the correct web traffic a modification of the ON/OFF method mentioned earlier is used. The IAT between each user's web sessions is exponentially distributed with a mean arrival rate of 0.4 web sessions per hour. Each web session consists of *N* pages visited; the mean of *N* is 20 web pages per session. In each page there are M embedded references, where the mean of M is 8 according to Table 1, which comes from [21]. The time between each TCP connection, the TCP IAT in Figure 14 and Active OFF time in Figure 13, is seen in the table below and is 1.0 seconds in mean. The Page IAT has a mean of 58 seconds according to Table 1.

Table 1: Web traffic model for (HSDPA)

Number of busy hour web sessions per active web user [1/h]	0.4
Number of pages in a web session	20
Page IAT [s]	58
Number of TCP connections in the same page	8
IAT of TCP connections in the same page [s]	1.0
Downlink TCP connection size (including TCP/IP overhead) [kbyte]	25
Downlink packet size (including TCP/IP overhead) [byte]	750
Uplink packet size (including TCP/IP overhead) [byte]	145

Each client in the simulation starts each simulation with a short process to determine when the first web session should be performed. After a client has performed a web session it schedules a new web session according to the rate below.

The rate used during the simulations is based on the mean value of 0.4 sessions per hour and user. This value together with equation (6) yields a rate λ equal to 2.5.

The last three entries in the table above are used when calculating the total amount of web traffic sent during the simulations. This way the amount of naming system traffic can be compared to a client's total amount of traffic.

To calculate the amount of control plane traffic during a web session the following expression is used:

(Amount of control plane traffic) =
(Number of lookups during web session) × { (Size of lookup request sent to naming system) + (Size of lookup answer sent to client) + (TCP/IP overhead per packet) } [bytes]

So, the total amount (volume in bytes) of control plane traffic due to web sessions in the simulations is gotten from:

(Total amount of control plane traffic during simulation) =
(Total number of web lookups during whole simulation) × { (Size of lookup request sent to naming system) + (Size of lookup answer sent to client) + (TCP/IP overhead per packet) } [bytes]

To calculate the user plane traffic during web sessions the expression below is used. The values for the downlink TCP connection and uplink TCP connection is found in the table above. The 1024 below is the conversion from kilobytes to bytes.

(Amount of user plane traffic during web session) =
(Number of lookups during web session) × { (Downlink TCP connection size (including TCP/IP overhead)) × 1024 + (Uplink packet size (including TCP/IP overhead)) } [bytes]

The total amount (volume in bytes) of user plane traffic is then:

(Total amount of user plane traffic during simulation) =
(Total number of web lookups during whole simulation) × { (Downlink TCP connection size (including TCP/IP overhead)) × 1024 + (Uplink packet size (including TCP/IP overhead)) } [bytes]

3.9.2 Streaming Traffic

The data for the streaming traffic comes from [21] and is summarized in Table 2. In the simulation each user has 0.15 streaming sessions per hour. Each streaming session is modeled like following:
A user visits a web page that is providing a link to a streaming resource. The user clicks on the link and is provided with a URL to the streaming resource. The number of lookups is then a function of the page visited, the number of embedded objects, and the user clicking on the link to the streaming content. The number of embedded objects in a page is eight as seen in Table 1 above. The number of name-to-address lookups is then: 1+8+1=10.

Table 2: Streaming traffic model (HSDPA)

Streaming sessions per active streaming user in busy hour [1/h]	0.15
Maximum concurrent session per user	1
Audio only streaming sessions [%]	3
Holding time of audio only streaming sessions [s]	100
Audio only streaming session media rate [kbps]	20
Video only streaming sessions [%]	23
Holding time of video only streaming sessions [s]	200
Video only streaming session media rate [kbps]	115
Streaming sessions with audio and video channel [%]	74
Holding time of streaming session with audio and video channel [s]	200
Video media rate in streaming session with audio and video [kbps]	92
Audio media rate in streaming session with audio and video [kbps]	20
Streaming session signaling uplink traffic [byte]	5300
Streaming session signaling downlink traffic [byte]	3900

Each client performs a number of streaming sessions during the simulation time. The number of sessions depends on the length of the simulation and the rate of streaming sessions. The rate used in the simulation is $\lambda = 6.67$, which is based on the mean streaming session rate 0.15 and equation (6).

The relative amount of naming system traffic is calculated using the entries in the table above, showing the holding time and the media rate, together with the data obtained from the simulations.

Some simplifications have been done when calculating the load from streaming sessions. First, the streaming traffic is assumed to be sent with a constant bit rate (CBR). Second, the control traffic sent via TCP during a streaming session, see section 2.4.2 on Streaming Traffic, is omitted. The data rates given in the table above is assumed to be including streaming protocol overhead. To be able to add the UDP and IP overhead it is decided that the packet interval will be 20 ms, which gives that

50 packets per second is sent. The data rates then, for streaming traffic, including UDP and IP overhead and when using the values yielding the heaviest load (audio and video streams, as seen in the table above) on the network, is the sum of the audio, video, and overhead data rates.

The overhead from UDP and IP (converted to bits) is: 50× (8× 8+20× 8) =11200 bps (10.9 kbps).

So the total data rate due to streaming traffic and the coupled overhead from UDP and IP is the: 92+20+10.9=122.9 kbps.

The amount of control plane traffic sent during a streaming session, if the signaling traffic from the streaming protocol is omitted, is calculated using the following expression:

(Amount of control plane traffic during streaming session) =
(Number of lookups during streaming session) × { (Size of lookup request sent to naming system) + (Size of lookup answer sent to client) + (UDP/IP overhead per packet) } [bytes]

The total amount of control plane traffic due to streaming traffic in the simulations is calculated from the expression below:

(Total amount of control plane traffic during simulation) =
(Total number of streaming lookups during whole simulation) × { (Size of lookup request sent to naming system) + (Size of lookup answer sent to client) + (UDP/IP overhead per packet) } [bytes]

To calculate the user plane traffic during a streaming session, the following expression is used, the values of media bit rates is found in the table above. The reason for using 1024 in the expression below is to convert the bit rates to bits per second. During a streaming session a number of web lookups is made and is therefore needed in the calculation.

(Amount of user plane traffic during streaming session) =
(Number of lookups during streaming session) × [(Downlink TCP connection size (including TCP/IP overhead)) × 1024 + (Uplink packet size (including TCP/IP overhead))] + (Holding time of streaming session with audio and video channel) × { (Video media rate in streaming session with audio and video) × 1024 + (Audio media rate in streaming session with audio and video) × 1024 + (UDP/IP overhead) } [bytes]

Where the expression that is enclosed by curly brackets could be replaced with the total streaming data rate calculated above (122.9 kbps).

In the following expression the streaming traffic data rate (including UDP and IP overhead) is summed up and replaced by 122.9 kbps, this is done to keep readability. The total amount of user plane traffic during the simulation is then:

(Total amount of user plane traffic during simulation) =
(Number of streaming sessions during simulation) × { (Number of lookups during streaming session) × [(Downlink TCP connection size (including TCP/IP overhead)) × 1024 + (Uplink packet size (including TCP/IP overhead))] + (Holding time of streaming session with audio and video channel) × 122.9 × 1024 } [bytes]

3.9.3 Telephone Traffic

Each client in the network initiates telephone calls during the simulation. The rate of the call origination is based on the amount of speech traffic found in a WCDMA network. In Table 3, which is taken from [21], it is seen that speech traffic per client is 16 mErl during busy hour. For this value to be useful for the simulation it is transformed so that it tells with which rate calls originates for each client.

Table 3: Speech traffic model (WCDMA)

Speech traffic per CS subscriber [mErl]	16
Speech MHT for mobile originating calls [s]	90

$$\frac{\lambda_{mean} T}{60} = \frac{A}{1000} \Rightarrow \lambda_{mean} = \frac{A\,60}{1000\,T} \quad (7)$$

A = Speech traffic per subscriber [mErl]
T = Speech MHT for calls [s]
λ_{mean} = Call arrival rate during busy hour [1/h]

Putting the given value of 16 mErl into the equation above (7), which is taken from [21], will yield a result of λ_{mean} = 0.768 calls per hour. Feeding equation (6) with the mean arrival rate of the calls gives that λ = 1.3.

When comparing the amount of lookup traffic in the network to the amount of telephone traffic, the data rate of telephone conversations has to be defined. The same audio compression protocol, G.723.1 from ITU-T, that was used for audio streams above will also be used here. This codec works with two data rates, namely 6.3 kbps and 5.3 kbps, and it sends speech samples every 30 ms. The amount of IP traffic, from the user plane and control plane, is compared against each other in later sections. [45]

The actual data rate when sending one second of telephone conversation over an IP network using the 5.3 kbps codec is calculated as following:

Number of packets sent each second: 1000ms/30ms = 33.3 packets per second. Amount of overhead sent each second in bits per second (including UDP and IP overhead converted to bits): 33.3×(8×8+20×8) =7466.7 bps, which gives an additional data rate of 7.3 kbps.

So, to send one second of telephone traffic requires a bandwidth of 5.3+7.3=12.6 kbps.

The amount of control plane traffic that is sent during telephone call, if all other signaling traffic is omitted, is calculated using the following expression:

(Amount of control plane traffic during telephone call) =
(Size of one lookup request to initiate call) + (Size of answer to lookup sent to client) + (UDP/IP overhead per packet) [bytes]

The total amount of control plane traffic during a streaming session in the simulations is calculated from the expression below:

(Total amount of control plane traffic during simulation) =
(Total number of telephone calls during simulation) × { (Size of one lookup request to initiate call) + (Size of answer to lookup sent to client) + (UDP/IP overhead per packet) } [bytes]

To calculate the user plane traffic for a telephone call the data rate, including UDP and IP overhead, calculated above is used. Following is the expression used:

(Amount of user plane traffic during telephone call) =
(Speech MHT for mobile originating calls) × (Telephone call data rate (including UDP/IP overhead)) [bytes]

The total amount of user plane traffic during the simulation is then the total number of call times the user plane traffic per call:

(Total amount of user plane traffic during simulation) =
(Total number of telephone calls during simulation) × { (Speech MHT for mobile originating calls) × (Telephone call data rate (including UDP/IP overhead)) } [bytes]

3.9.4 Mobility Movements

Each client in the network moves according to the Random Walk Mobility Model. Each client will bounce back into the simulation area when reaching the border of the simulation area, the new direction of the client is based the well known law in physics that tells us: "*the angle of incidence equals the angle of reflection*". The speed of the client is not affected by the bounce. The parameters that can be changed in the mobility model are: the size of the playground, minimum speed, maximum speed, and step length. The step length is explained in section 4.5.3.

3.9.5 Topology

The network topology is constructed in a top-down fashion. First the core network will be constructed. The core network try to mimic a carrier network found in the real Internet. On to that network several ISP networks is connected, and some ISP networks is also connected to each other, so called peering. To each ISP network there is a number of customer networks connected. These customer LANs vary in size from a couple of hosts to a larger network consisting of hundreds of hosts.

The reasons for not using a topology generation software are several. First is because topology generators work poorly for the OMNeT++ simulator. It would have been an extensive process creating a topology and then converting it to a format readable by OMNeT++, and the conversion process could result in loss of topology accuracy. Second, creating a topology at random that has no connection to the real world at all, versus, building a topology by hand that somewhat relate to some real world topology, where weighted against each other. In this case the second alternative where considered more attractive.

To create the topology a rather simple structure is used that corresponds to the real world but is highly simplified so that the number of hosts and routers is decreased as much as possible. Having this structure makes it easier to expand further to cover the needs that may arise, yet keeping track of the real world model so that they still match.

3.10 The Distribution Aspect of the RLR

This is a rather important issue because the RLR has to perform as good as possible. Questions regarding the RLR are amongst many: Could the RLR be represented by a

network of distributed servers, like the DNS solution? Would the RLR perform better if it were based on a DHT-system? The distribution aspect of the RLR will be evaluated theoretically, based on the results gained from the simulations.

The RLR could be represented by letting it be implemented as the DNS system or by using a DHT-based approach. So, things to consider are; regarding the DNS case, how is the information stored in the DNS. Data that needs to be stored is the *name*, part of the *address*, and the *RTC*. This could be realized by creating a new resource record that contains this information. How will this be achieved if the RLR is represented and distributed using a DHT-based method?

There will be rather small changes in the RLR. So the DNS or the DHT-based implementation will be rather stable. It is when a user changes operator or switches country that changes will occur at the top of the hierarchy.

How will TTL and cache affect the name-resolution hit ratio when using a DNS solution in the RLR? The meaning of name-resolution hit ratio is; when making a query to the naming system an answer is returned, if this answer is correct or not can not be decided until a connection to the sought resource is tried. If the answer was correct and a connection can be established, it is a hit in the naming system, if the answer to the query contained the wrong information, thus resulting in that a connection can not be established, a miss in the naming system has occurred. Information about name-resolution hit ratio is found in the previous thesis [23].

4 Implementation

This section of the document describes how the different models, described in Section 3, have been implemented to suite the OMNeT++ simulator. Further the design choices made during the implementation processes is explained.

4.1 General Information

Why Access Points Are Extended

The access points where created to solve three problems; mobility, routing and the relatively high memory consumption of each client.

To understand what an access point does, it is easiest to think of it as an access point to a network but with agent software acting on commands from clients. These agents create messages and put them onto the network as if the clients would have sent them. The client shares the IP address with the access point and all other clients connected to this access point. This does not cause any problems since all packets traversing back to a client should be deleted at arrival, and there are no measuring points after that point.

The problems of mobility and routing are really interconnected because when a node moved it first had the problem of keeping its IP address or, getting a new address and getting it to work. Since if the clients get a new address all the routing tables in the network have to be rebuilt. This is a too processor consuming operation to be able to do for each movement a client.

The memory was also another matter, a normal node in OMNeT++ would have had several extra modules, like modules for TCP/IP, routing tables, and so on, but now with thinner clients all those are moved to the access points. Since there is a limited amount of access points it gives a huge save of memory.

A client communicates with an access point by direct calls to the access point object. A client has the ability to call any access point at any time, but it calls the object that it finds responsible for its current position in the simulation world.

Having access points working as they do creates a big problem for distributed simulations because all access points and clients have to be placed in the same CPU process. And by having this constraint the only thing that could be distributed would be the rest of the topology, but since the clients are the heaviest ones in terms of processing there would not be any gain from doing a distributed simulation. This can be circumvented with the use of clones; an access point could exist in two different CPU processes having the same settings and acting like one common object and thereby making it possible to have clients on different processors. However, this solution has not been implemented in this thesis simply because it was not needed.

What System Databases Really Do?

System databases are primarily used to deliver global knowledge to clients and servers. The knowledge provided is: What access point is responsible for is what area and what name has which address.

Having one global repository makes it a lot easier to configure the network; since all components just have to register there, and everyone has access to this information. Otherwise the knowledge would have to be propagated to each and every node in the network, which also demand that every node stored it in their memory and that would mean an increased use of memory. However, this global repository forces all the nodes using it to be on the same process, and therefore limits the benefits from distributed simulations, since most nodes would actually have to be on that very process. This can however be circumvented with having clones that would communicate update information between each other, or making the global repository work with message passing instead of direct method calls, as it does now.

The Simulation World

The world where the clients move around inside an area is also referred to as a playground. This playground is divided into squares, and each square is managed by an access point. The clients' moves around in the playground specified by the mobility model and by doing so they change square inducing a movement to the system.

4.2 DNS

Name servers will take on several different kinds of roles; root server, domain server and local DNS server. They will run the same code but be placed in different parts of the system and are configured differently.

There will be one local name server for each AS in the topology, these name servers will have two different tasks, one is to answer lookups from the data it has, and another is to do the lookups for a client that is connected to the AS it was responsible for.

Lookups

Lookups are implemented only with the iterative method. The clients will initiate its lookup by signaling its access point to send a lookup with a name given as a parameter. The access point will then send the lookup to its local name server. The local name server will first do a cache lookup; the cache lookup will rewrite the field's *resolved_host_name* and *host_address* to the closest hit in the cache. If then the server notices that *resolved_host_name* is the same as *host_name*, the request is done and it is sent back to *original_sender*, however, if it is not then it is sent to the DNS server specified in the *host_address*. At that server the request will be answered in the same way, and then sent back to the requesting server where the request will be examined to see if the request is complete. If the request is complete, it is sent back to the original sender; otherwise it will continue the lookup process until it is done. In the implemented system it can never happen that a hostname is not resolved.

Updates

Updates are very straight forward, the clients remember to which server they were registered to at the beginning of the simulation. They will stay registered to this server throughout the whole simulation so updates are just sent there to change the IP address to their new one.

4.2.1 Packet Headers

Table 4: DNS lookup message

isRequest	Used as a toggle to determine if the message is sent to or from the local name server
original_sender	Holds the address of which client that initiated the lookup
host_name	Host name that is going to be looked up
host_address	Address of the host or the next hop to take in resolution
resolved_host_name	A string containing what has been resolved so far
clientTimeStamp	Not used but carries the timestamp of when a packet was created.

4.2.2 Message Types

Table 5: DNS update message

name	Host name to be modified
action	What actions to do, only "update" used
ip	The new IP address of the host

4.3 Chord Based Name to Address Resolution

During the implementation of a Chord based naming system the focus where on the Chord nodes in the ring. They should mimic the real behavior of a Chord system, which is they should communicate and forward/store data according to the specifications. Each node in the naming system should initialize itself with regards to the finger table, successor list, and a predecessor pointer. Information about the size of the ring should be given to the naming system during the start up of the simulation.

Clients in the network create updates and lookups that are sent towards the naming system via an access point. Clients do not communicate with each other in these simulations.

4.3.1 Packet Headers

A new packet type was defined for the Chord protocol. This packet is then sent inside an UDP packet in the simulation. This packet is used when clients makes updates or lookups in the naming system. The newly defined packet is also used for communication between nodes in the Chord naming system.

This packet definition is mainly the same as the one used in the previous thesis. The packet holds information about keys that should be updated, queries about certain keys, how the packet should be routed, and administrative information. One field in the packet tells the naming system whether to perform a recursive lookup or an iterative one. All the fields in the Chord packet is listed and described in Table 6.

Table 6: Chord message

packetName	What is this packets message type? Response, update, request (see section 4.3.2).
packetSource	The address of the client that contacted the Chord Naming System
chordSource	The address of the originating Chord node to send the response back to
symbolicName	The symbolic name that the query or lookup regards
nextHop	Where the packet should be routed next when in the Chord Naming System
hopCount	The number of hops in the Chord Naming System
lookupAlgorithm	Tells the naming system whether to use recursive lookup or iterative lookup
messageType	Defines what type of information that is carried in the packet
resolvedAddress	The address connected to the symbolic name that is registered in the system.
sendTime	The time when the packet was originated

4.3.2 Message Types

In the simulation there are different kinds of messages sent. For the clients and Chord nodes to be able to distinguish between them the *packetName* needs to be set. In Table 7 the possible values for *packetName* is listed and described.

Table 7: Possible values for packetName

10	CHORD_CLIENT_REQUEST	Request message from client with key
20	CHORD_CLIENT_RESPONSE	Response message to client with data
30	CHORD_REQUEST	Request message from chord node to chord node
40	CHORD_RESPONSE	Response message from chord node to chord node, only when iterative lookup
50	CHORD_NODE_FOUND	Response message when responsible node found

When the type of the packet is set, the system needs to know which information that the packet contains. This information is set so that the receiving entity knows what to do with the packet. This information is stored in the *messageType* field, and the different values are described in Table 8.

Table 8: Possible values for messageType

100	CHORD_UPDATE	This message contains update information
101	CHORD_LOOKUP	This message contains information regarding a lookup
102	CHORD_UPDATE_OK	Marks a message telling that an update was successful
103	CHORD_LOOKUP_OK	Marks a message that a lookup was successful
104	CHORD_LOOKUP_ERROR	Error message sent when no information was found

In the Chord message packet it is possible to tell the naming system which lookup fashion to use. Setting the field *lookupAlgorithm* to one of the values in Table 9 forces the naming system to use either recursive lookup or iterative lookup.

Table 9: Possible values for lookupAlgorithm

0	CHORD_RECURSIVE	Recursive chord lookups
1	CHORD_ITERATIVE	Iterative chord lookups

4.3.3 Chord Nodes

This implementation of the Chord nodes is a modification of the previous work, which where done for ns-2 simulator. The old implementation where written in C and contained a lot of ns-2 specific code. But the functions for building the finger table, successor list, and the predecessor pointer where easily modified to suit the OMNeT++ simulator.

Each Chord node is given an IP address, a node ID, and the size of the key space at startup. From this information each node can obtain full information about the ring, but each node is only concerned about the nodes that need to be reachable for its own needs, that is, the nodes in the finger table, the successor list, and the predecessor node. During the simulation the number of nodes will vary from 32 nodes to 64 nodes. The key space is dependent on the number of clients in the simulation.

A local database is kept by each Chord node in the system. In this database the name-to-address mappings is stored. This database is not the same database as the System database.

Each node has a finger table of size *m*, where *m* is the number of bits in the key identifier space. See section 2.2.2 for information about finger tables. Each node can easily convert a node ID to the corresponding IP address thus easily populating the successor list and finger table.

The key space consists of the numeric values from 0 to 2^x, where x is between 10 and 14. For example, if the key space is equal to 2^{10} (1024) keys, a Chord node with ID 0 and a ring size of 64 nodes are responsible for the keys 4033 to 0. The node with ID 1 is responsible for the keys 1 to 64, and so on.

Packet Processing

The protocol consists of two parts; one send message function, and a receive message function. A Chord node receives a packet from an access point, on behalf of a client, or another Chord node in the naming system. The node examines the packet, processes it and depending on the packet type it is sent to the next hop or back to the access point who issued the request.

If a Chord node receives a packet with a request from an access point (on behalf of a client), i.e. a message with the *packetName*-field set to CHORD_CLIENT_REQUEST, the first action taken by the node is to see whether it is the responsible node for the given key. Two cases now occur: The first is that the Chord node is responsible for the given key. It then checks the message type to see if the packet is an update packet (CHORD_LOOKUP) or a lookup packet (CHORD_UPDATE). If it is an update packet the node stores the information, and no response is sent back. If it is a lookup request the node looks in its database to see if there exists some information about the given key. The Chord node sends a response back towards the querying client, through the access point, with the address coupled with the given key and sets the packet type to CHORD_CLIENT_RESPONSE and the message type to CHORD_LOOKUP_OK if information was found in the database. If it does not find any information about the given key the node responds with setting the message type to CHORD_LOOKUP_ERROR.

The second case that could occur is that the Chord node is not responsible for the key. It then checks to see if the packet is a lookup packet, if so it checks the local cache for information about the node. If it gets a hit in the cache it sends an answer towards the querying client, via the access point, as described above. If the Chord node is not responsible and has no cache information it tries to find a node in the ring that is responsible for the key. The node does this by first checking to see if its predecessor is responsible for the given key. If the predecessor is not found to be responsible it checks to see if a candidate exists in the successor list, and then last it checks the finger table. When a responsible node or a better candidate is found the packet type is changed to CHORD_REQUEST, the node puts its own address in the field *chordSource* and the receiving nodes address in the *nextHop*-field, and forwards the message that node in the ring.

When a Chord node receives a packet of type CHORD_REQUEST it knows that the packet came from another node in the naming system. It performs one of the two cases described above with slight modifications. If it is a lookup and information is found at this node, the answer is sent to the Chord node that has its address set in the field *chordSource*. If it is a lookup but no information is found and the lookup method is iterative the node puts the address of the next Chord node (responsible or better candidate) in the *nextHop*-field, but sends the packet back to the node whose address is found in the *chordSource*-field. Update requests is handled the same way. A message that contains an answer that is sent between the Chord nodes has a packet type of CHORD_RESPONSE.

If a Chord node receives an update request that it is responsible for, and caching is enabled in the system, it constructs a packet of type CHORD_REQUEST with message type CHORD_CACHE_MSG and sends it to its predecessor. When a node

in the naming system receives a packet that looks like just described it puts the information in its local cache and discards the message.

4.3.4 Access Points

Each client that is connected to an access point triggers a method in this object that causes it to make a lookup request for a random name, or if the client is newly arrived, to make an update in the naming system. All messages that are returned from the naming system are discarded by the access point. The access point constructs a message according to the chord message specifications, and sends it to a random destination in the naming system using UDP.

An access point will only generate packets that are requests sent towards the Chord naming system, which is with packet type CHORD_CLIENT_REQUEST. The message type of this packet is either an update packet (CHORD_UPDATE) or a lookup packet (CHORD_LOOKUP). All packets that are received by the access point are silently discarded.

4.3.5 Clients

Each client makes lookups that is triggered by their traffic pattern behavior. When a lookup is to be made the client triggers a method in the access point that is responsible for the client at that time. The client passes the name to be looked up as a parameter to the access point method.

Each client is also responsible for its own movements in the network. When a client has made a movement it checks to see whether it has moved into another access points area, thus it should make an update in the naming system, of if it still resides in the care of the old access point, thus the client should not make an update.

The clients does not receive messages from the network, it only handles messages that is so called self messages.

4.3.6 System Database

The database used in the Chord simulations is different from the local database kept at each Chord node. The system database is not used to the same extent as in case of DNS and Path-Coupled as it does not store the name-to-address mappings globally. The system database keep track of each client's location, i.e. coordinates, in the simulation area, and each access point's area of responsibility.

4.4 Path-Coupled Name to Address Resolution

Lookups

A lookup is initiated by a client in the access point responsible for its area. The lookup request is then sent to the RLR to find out to which area to route it. The requests are then sent back to the access point that will send the request further to the next area. At the border of all areas there is a border router which will pick up the message, send a request to its responsible RS to resolve where the packet should be routed next.

Detailed

A message is sent from the border router towards the RLR carrying the name to resolve and the area variable set to 0, the rest of the message variables are kept blank at the moment. At the RLR the fields *heading*, *headingAddress* and *next_hop* will be updated, more information about them can be found below in Table 10. The *heading* and *headingAddress* fields store an address of where the host is currently located; this is in order to keep consistency. The RLR send the request back to the initiating border router that will just add one to the *area* field and then send the packet further towards where it is specified in the *next_hop* field. This address will be the address of a network or the highest level, example "A". The border router that receives the request now knows, due to the *area* variable, that it is supposed to ask the highest level for directions. It adds one to a counter of pending requests and then sends it towards the RS of the same level. The RS then updates the fields *heading* and *headingAddress* corresponding to the method to keep consistency as well as updating the field *next_hop* to give the area of where to be routed next. Then send the request back to the edge router. The edge router sets the *area* variable to 2 and sends it towards the *next_hop* address. The edge router with this destination address (might be the same router) receives it and knows from the *area* variable that it should ask the RS of level 2 for the final address of the host. The packet is then routed to that access point that deletes it. Nothing is sent back in this implementation because it can be done in two ways, and this thesis is meant to be as general as possible.

In the explanation of the method it is said that routers will keep a copy of the request and then ask the RS. However, it was not implemented this way, but to measure how much memory that is needed to store all pending requests a simple variable was used. This variable was increased or decreased depending on the message, for example when a question went out towards the RS it was increased, and when a reply came back it was decreased. And the nodes were set to remember the highest value of pending request, and reported them at the end of a simulation.

Updates

Updates are a bit more complicated then in the DNS and Chord name resolution methods. Since information higher up in the architecture sometimes has to be updated. Updates start with a client realized it has changed access point; it will then initiate a lookup in the new access point. This update message will be sent to the local RS of the access point; from there the message will be sent upwards in the hierarchy until it reached the level specified in the *updateAt*-field. Then the database will updated with the new value. Then the message will just traverse downwards toward the old domain of the client, until it arrives at the old domain's RS, where it will be deleted.

How Consistency Is Maintained

When a lookup is made the packet records a *heading*, the *heading* points to where the sought host is at the same moment the lookup is initiated, then at every step of the way when its processed the *heading* information is compared and checked if the packet is currently located below or above the point of the tree (of RSs) where the information should be visible. If it is above then this packet should not ever have seen this information so it disregards it and instead continues towards its destination specified by its heading value. And if it is below, the *heading* information will be updated to the new destination address. To clarify consider the following example: The packet is currently located at the RS of *A* and is heading towards *A.4* and the

database was just changed so that the sought host was relocated to *B.2*. This would mean that the RS of *A* should have no idea about this change because the path was changed higher in the hierarchy than it is currently located, so the packet should continue towards *A.4*. This will also yield a miss in the lookup if accuracy would be measured.

Dummy nodes

Dummy nodes were constructed to assign nodes in an area a common IP address as well as help the automatic procedure of constructing routing tables for these IP addresses. Everywhere in the system the shortest path towards an autonomous system should be chosen and therefore the dummy node has to be placed at the very center of the autonomous system, or being connected to all of them. The dummy node is assigned an IP address at simulation startup as well as gets added in all routing tables of the whole network. The dummy node then registers itself in the database on which area it is responsible for. Then the nodes parameterized to belong into an AS contacts the dummy node to tell it to add its address to its own stack of IP addresses and thereby making it pick up any packets heading towards this network. A node in the simulation environment can have multiple IP addresses mapped to one network card and therefore it can have all the addresses for all the AS's it belongs to, for example the IP address of itself, the IP address of A and of A.4, three addresses in total.

Table 10: Path-Coupled lookup message

host	Name to lookup
next_hop	Where the packet should be routed next
area	Is an integer that shows where the packet is in the routing, for example 0 represents the RLR level. For more information see section 4.4
heading	This is used to say where the packet was heading, its used to keep consistency, more in section 4.4
headingAddress	Address of where the host used to be, se section 4.4

Table 11: Path-Coupled update message

name	Host name to be updated
oldDomain	Name of the old domain
newDomain	Name of the new domain
address	New IP address
updateAt	Where in the hierarchy it has to be updated at to keep consistence, it has to do with the hack described in section 4.4
up	Boolean to show if the message is going upwards or downwards in the hierarchy, it makes programming a lot easier.

4.5 Simulation Model

4.5.1 Topology

In the end it was chosen to implement the topology using a map of a real network. A real world map of a topology was adopted to fit our simulation, and then some extra

core network was added. But since the statistics that are being collected is independent of topology a really simple core network was chosen.

The topology consists of 36 AS, varying in size and shape. These 36 AS can easily be organized into four bigger AS's to be used for hierarchical reasons. The sub areas of these bigger ones are connected to each other through a router each then the four bigger AS's is connected to one core router. This topology enables everyone to reach each other, and it is correct in the sense of how a packet needs to travel to get to each other. The link delays will be set to represent larger distances when they are supposed to be so.

The hosts are moving independently between the different AS's.

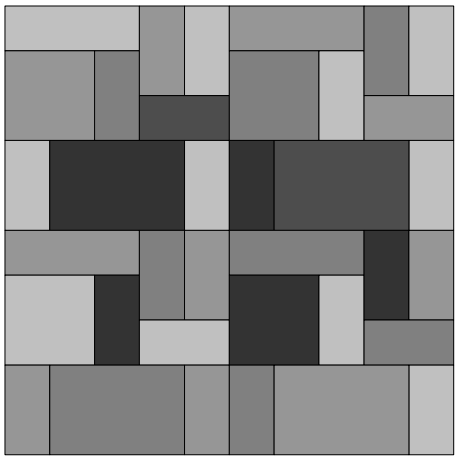

Figure 29: Map of the autonomous systems

4.5.2 Traffic Pattern

Since this work is only interested in the traffic that concerns the naming system the user traffic of the clients is never generated. Each client just pretends to make for example a web session, whereas only naming system traffic is generated. So, for example, when a client decides to start a web session it does not send any real web traffic, it only uses the pattern of a web session to know when it should make lookups in the naming system.

The traffic patterns in the simulations done is, as mentioned earlier, divided in to three different types of traffic: Web traffic, Streaming traffic, and Telephone traffic. This causes web sessions to have the heaviest load on the naming system.

Each client schedules a message in the system telling it when to start a session of specific kind. All traffic patterns types are independent of each other.

Web Traffic

The inter arrival times of a clients web sessions during the simulation is a Poisson process with exponentially distributed inter arrival times. The rate at which web sessions originate is 2.5 according to calculations in section 3.9.1. This value is feed to the exponential random number distribution function in the simulator.

Instead of scheduling all traffic messages that is coupled to a web session at the initialization of a session, one message at a time is scheduled that contains information about the rest of the session. The reason for this is to minimize the load

on the system, and it suits the goals of trying to restrict the memory usage during simulation time.

The web session behavior is explained below with pseudo code:

```
n ← number of pages in this session
while n > 0 do
   perform lookup in naming system
   m ← number of embedded references in this page
   while m > 0 do
         perform lookup in naming system
         wait so that TCPIAT is ~1.0 seconds
         m ← m-1
   wait so that pageIAT is ~58 seconds
   n ← n-1
schedule a new web session in the future
```

Streaming Traffic

Streaming traffic could be seen as a special case of web traffic, with $n = 1$. That is, during simulation a client visits one web page that contains m embedded references and a link to the streaming object. The client follows the link to the object, thus making an additional lookup for the streaming resource. The rate for streaming traffic is 6.67 according to the calculations in section 3.9.2, and this is the value that is feed into the simulator.

Telephone Traffic

The load from telephone traffic in the system is rather small. When a call is initiated a request to the naming system for the address of the receiving client is made. During the simulation of telephone traffic no efforts is made to see if the clients has moved or updated their naming system information during the phone conversation. The reason for not doing this is the time limits of this project. The rate parameter used when simulating telephone calls is 1.3 according to the calculations in section 3.9.3.

4.5.3 Mobility Model Implementation

The clients in the simulation use the Random Walk mobility model when simulating their movements. The behavior of the model can be controlled by the parameters in Table 12, and is sent to the simulation via *omnetpp.ini*. A client's movement is controlled by a timer that is set by the client during the initialization process. The timer expires at interval set by the *moveInterval* parameter. When a client's timer expires it checks to see if it should continue to move in the current direction, or if it should choose a new *direction*. If a new direction should be chosen, it is chosen from a uniform distribution (0, 2π). The client also chooses a *speed* to use when making its movement; the speed is chosen from a uniform distribution (*minSpeed*, *maxSpeed*). The client then calculates the *step length*, which is a function of *moveInterval* and the *speed*, and finally decides the number of *steps* to make (*distance*/*step length*) when traveling in the chosen *direction*.

Table 12: Parameters that control the mobility model

moveInterval	Time interval to trigger movement (in seconds)
moveKind	Bounce at the border or torus like movement
minSpeed	Minimum speed for a client
maxSpeed	Maximum speed for a client
distance	The distance to cover when choosing new direction
maxX	Boundary of simulation area in x-direction
maxY	Boundary of simulation area in y-direction

4.6 Simulation Environment

In this section follows a description of how the simulation program was setup and configured to run accordingly.

The simulation software is highly parameterized. However, most parameters were chosen to be kept constant to limit the amount of flexible parameters.

In the simulation environment there is an area called the playground. This is a square of the coordinates (0, 0) to (999, 999). In this area the clients can move freely. This square is in its turn divided into 100 squares. One or several squares can be used as an AS, for our simulations the same layout is used by all simulations, the layout can be found under section 3.9.5.

All the links in the network has been configured with different delays, all to make the comparisons between the systems as correct as possible. The links has also been customized in such a way that they can handle extremely many concurrent transmissions and also the data rate is vastly exaggerated. Also, there were no queues because queues would sometimes change parameters and thereby not letting the simulator run as it should. These settings were made because there is no interest in measuring congestions in the network since the methods should be investigated independently of the network topology as stated in 3.9.5. Below, in Figure 30, the network with all the delays is depicted. Below the figure is Table 13, which shows the latencies used for the different name resolution methods.

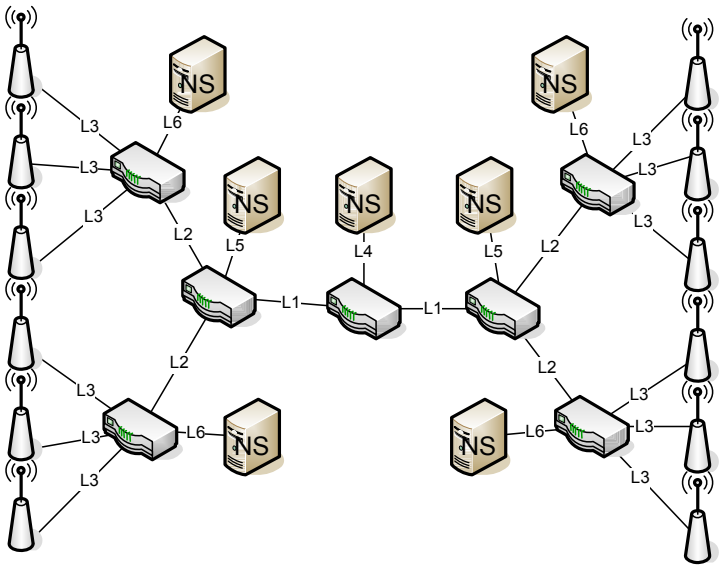

Figure 30: Network layout

Table 13: Network latencies

	DNS	Path-Coupled	Chord
L1	25	25	25
L2	35	35	35
L3	5	5	5
L4	10	10	10
L5	10	10	–
L6	5	5	–

The communication in the network takes place over UDP, UDP is traditionally unreliable but since the simulator is a controlled environment and therefore wont drop any packets unless set to do so, it becomes reliable and usable for the simulation purposes, hence UDP is used.

5 Analysis

This Section presents the results the analysis from our simulations, as well as analysis of each protocol. In the end the three name-to-address resolutions protocols are compared to see how they behave and perform compared to each other.

5.1 General

When looking at the number of requests for the different simulations it should be about the same because of random numbers property of being evenly distributed. These diagrams are here to show how many request that is initiated in the system. It is easily spotted that all the different systems are generating approximately the same amount of requests.

Figure 54 below shows that all the four different simulations transmitted about the same amount of lookups. The difference between them is too small to be relevant. For more details see the appendix.

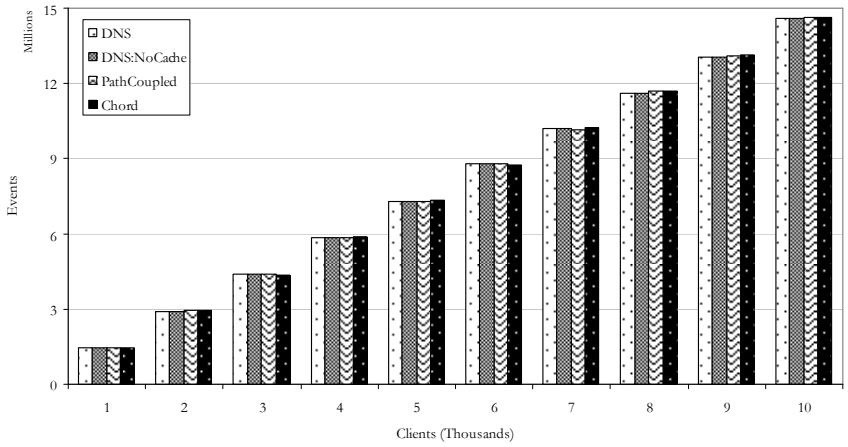

Figure 31: Initiated lookups

The figure below, Figure 55, shows that it is about the same amount of updates in all of the four different simulations. The difference is too small to be relevant. The variation of random numbers made chord have slightly more updates, while the difference between the others were far less. For more information see appendix.

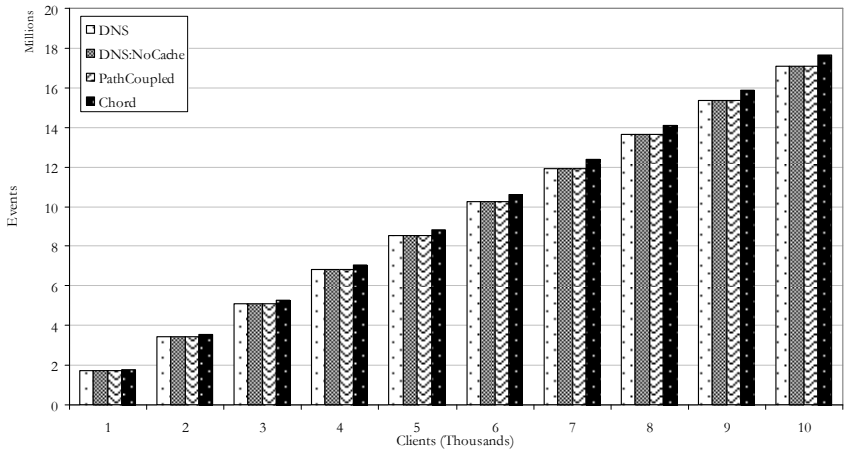

Figure 32: Initiated updates

5.2 DNS

This sub section contains analysis of the simulation results obtained in the DNS-simulation. It is expected that the DNS simulation will scale ok with cache and very poorly while not using cache, due to the nature of the DNS structure.

DNS was analyzed using two different settings; with cache and without cache. The first section will take a look at the cached version. It will then be followed by the un-cached version. Both will be analyzed in terms of traffic to levels of name servers and traffic in each server in medium on different levels. With medium it is meant the total traffic volume to that level divided on all the servers of that level.

DNS

In the figure below, Figure 33, it can be observed that for every one request there will be three times as many events on the level 2 parts of the system. Theoretically it should be four times, but since it benefits from caching it is less. The value becomes less because the all levels are not contacted for every initiated request. This is because replies from different parts of the system are being cached, the effects from this caching can be seen in the diagrams presented later in this section, it will also be seen in them that some levels are more affected then others. However, what this shows as a bigger picture is that most of the traffic takes place on the level 2 part of the system and this is good for scalability reasons since the level 2 consists of many more servers then the higher layers.

As mentioned before it can be seen in the figure that a request creates more than just one processing event. This is because a request will visit many nodes before it is resolved. It can also be seen that in comparison the higher levels of the structure is omitted from a lot of the traffic. This makes it possible to scale it even further with more AS's and the traffic levels would still be at a ok volume, however, it will hit a roof at a later stage that it can not get past.

It might be more convenient to show the medium amount of traffic to the different nodes to gain understanding how well it would scale. Figure 33 below depicts this.

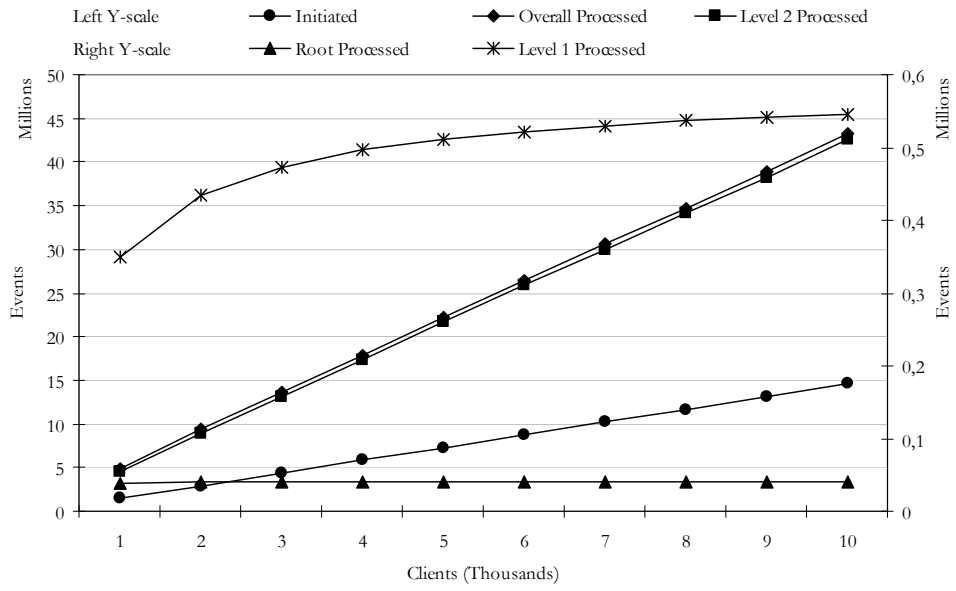

Figure 33: DNS lookups

In Figure 34 a surprising fact is seen in the comparison, the root scales better then the level 1. This has to do with caching and the number of level 1 server's. If there would be more servers of level 1 there would be more server's sharing the load and therefore the medium traffic would go down. It can also be seen that the traffic for the levels 2 has a linear increase, which is good enough for scalability since a level 2 domain can always be configured in size and the problem can therefore be planned away.

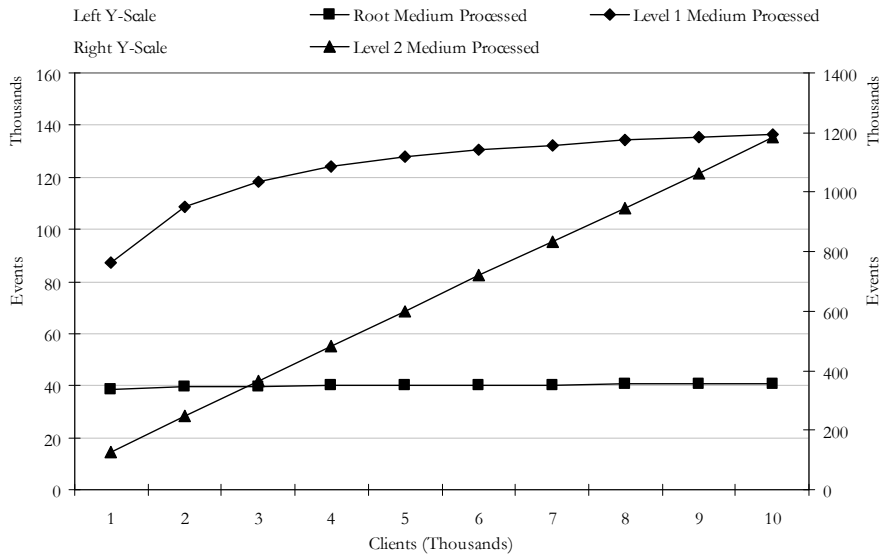

Figure 34: DNS lookups medium per server

DNS Update

As already known, in the tested DNS system the servers where not changing address and therefore there is only updates done at level 2. This is what can be observed in the figure below, Figure 35, the curves of initiated in total ("Initiated") and received at level 2 ("Level 2 Processed") follow each other perfectly. While the other two remains at zero because they do not receive any traffic, and they should not because of how the DNS system is constructed.

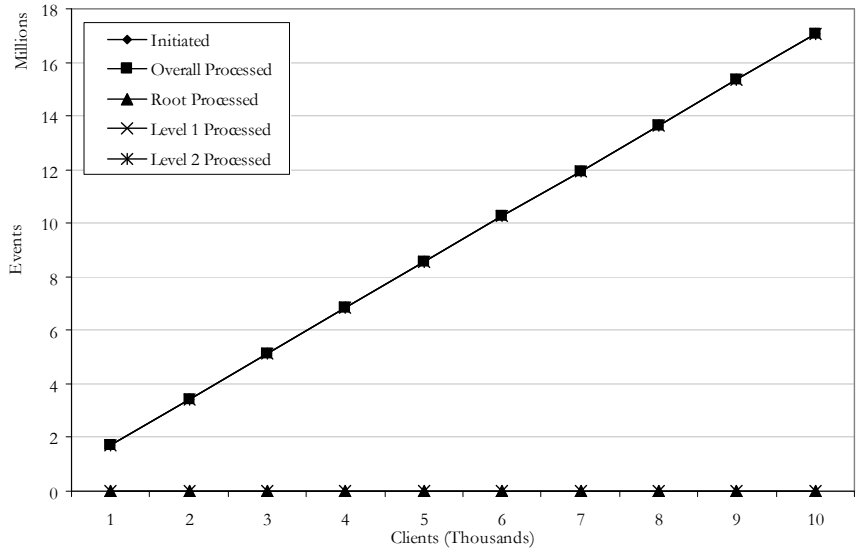

Figure 35: DNS updates

Even though it is obvious how the mean value will be, it is presented Figure 36 for comparison purposes. The only interesting thing to see is that the increase is linear.

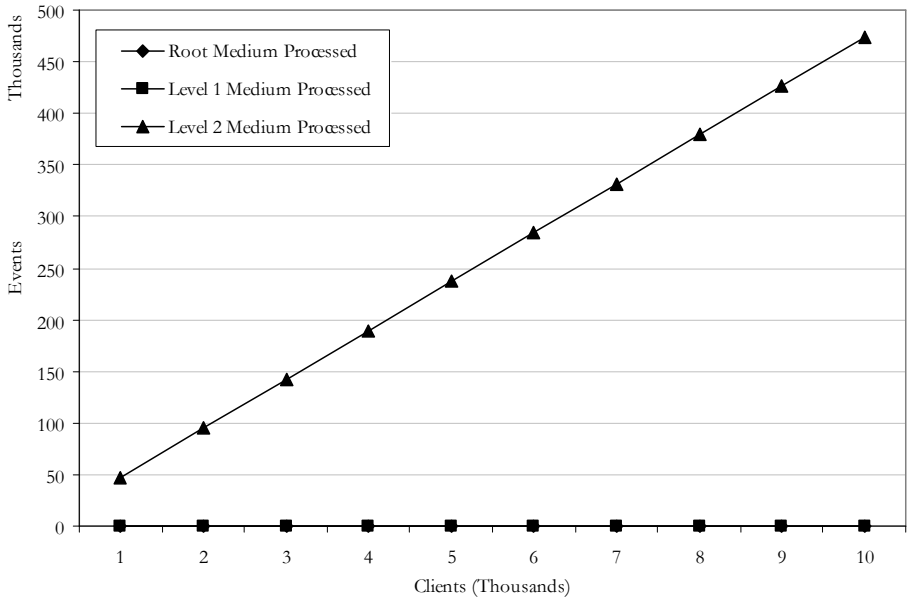

Figure 36: DNS updates medium per server

Overview

Figure 37 shows the total number of processed events on level 2, and it is almost the same as ”Processed Requests”, that is a measurement of how many processing events in total that has happened It is only a very small portion of the events that takes place elsewhere in the system. This is a good factor since, as mentioned earlier; there are many different servers for the level 2 to share the load. Other than that it can be noted that the root is not affected at all by the scaling of clients; this is due to the caching as mentioned earlier.

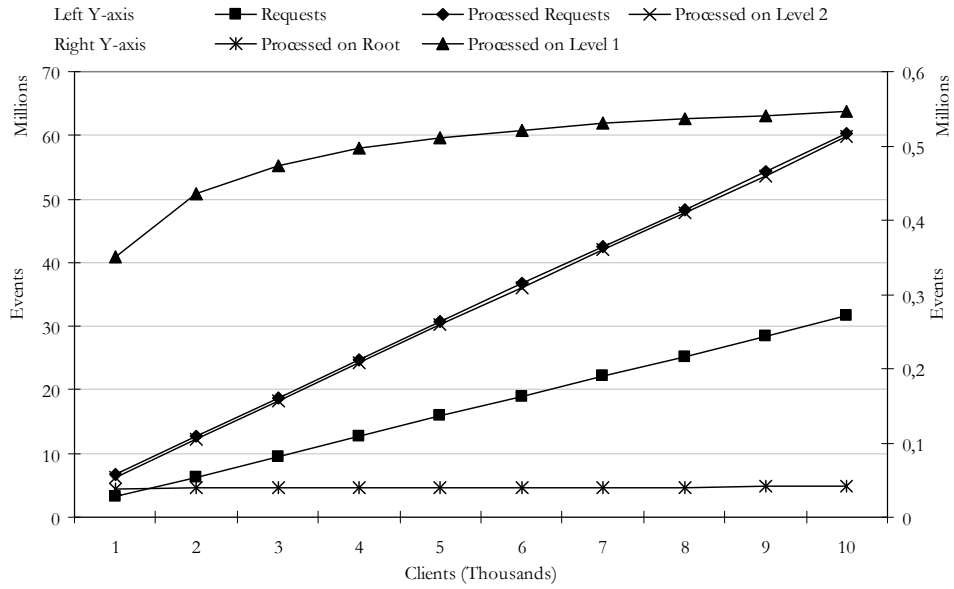

Figure 37: DNS lookups and updates

Figure 38, below, shows that it scales well on root and on the level 1. As well as linear on level 2 and this is not necessary bad.

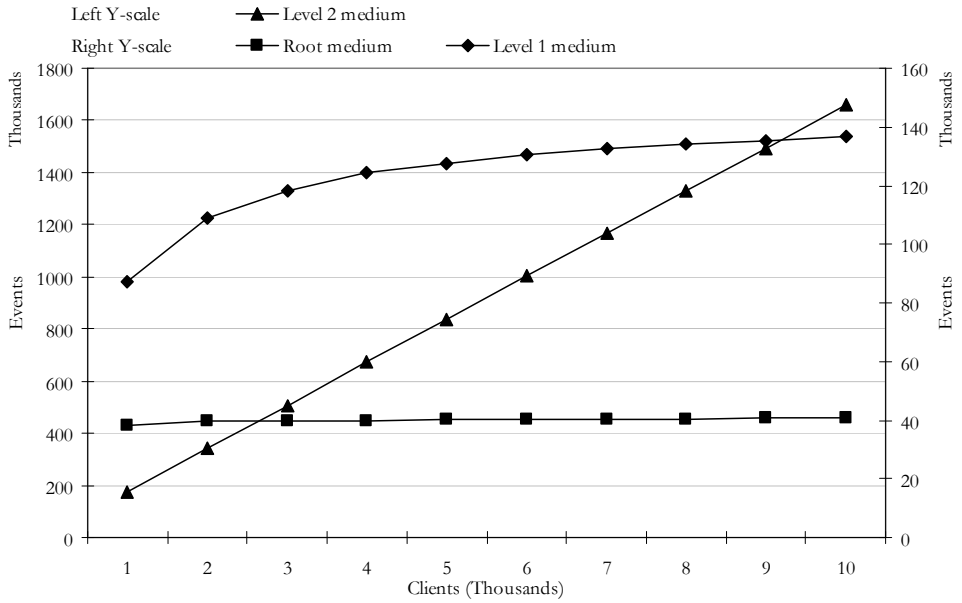

Figure 38: DNS lookups and updates medium per server

DNS Un-Cache Lookups

The curve for initiated lookups ("Initiated", Figure 39) is identical to the curve of processed lookups at the root level ("Root processed", triangle data points in Figure 39); this is because a request will always visit the root once and only once. The same applies to the level 1 server; all requests will visit this level as well. However, it is to be kept in mind that the level 1 consists of more servers then the root level, and the load can therefore be distributed. Even if this might seem like a solution there is a problem, most names might be located on the same.

It can be seen in Figure 39 that the processed on level 2 is approximately five times higher then initiated lookups; this is because a request will go to multiple locations in the network before it is completely resolved.

Note that the three curves (in Figure 39) concerning lookups "initiated", "root processed" and "Level 1 processed" is all aligned, this is because all requests will visit those nodes once and only once. But what is more interesting is the fact that the level 2 gets a lot more traffic to it. It does 80 % of the systems total processing. This is a good factor since there are a lot more nodes on level 2 sharing this traffic.

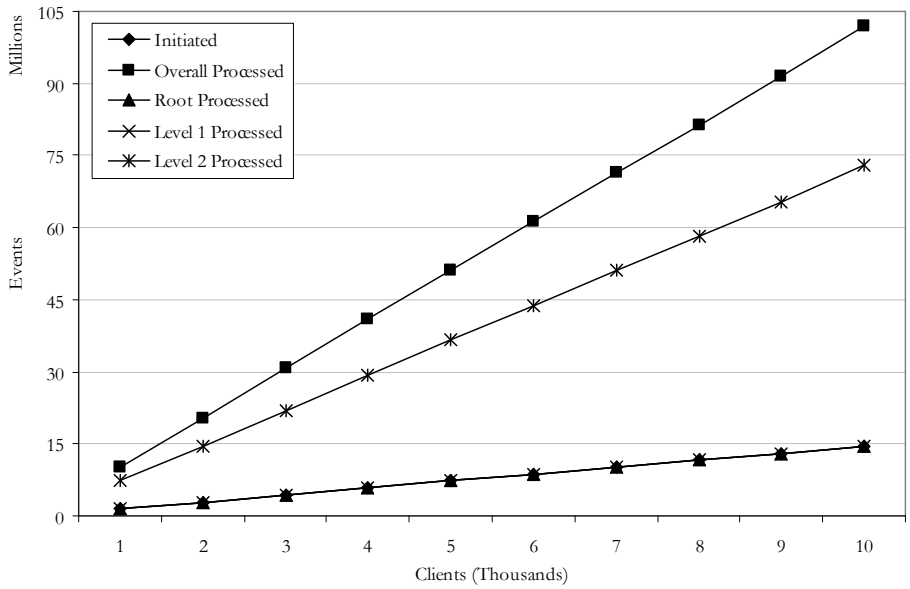

Figure 39: DNS un-cached lookups

Medium

What is interesting to see, in Figure 40, is that the medium traffic on the root level grows a lot faster then at level 1 and level 2; this is absolutely horrible for scalability because the root server will be choked long before the other levels are.

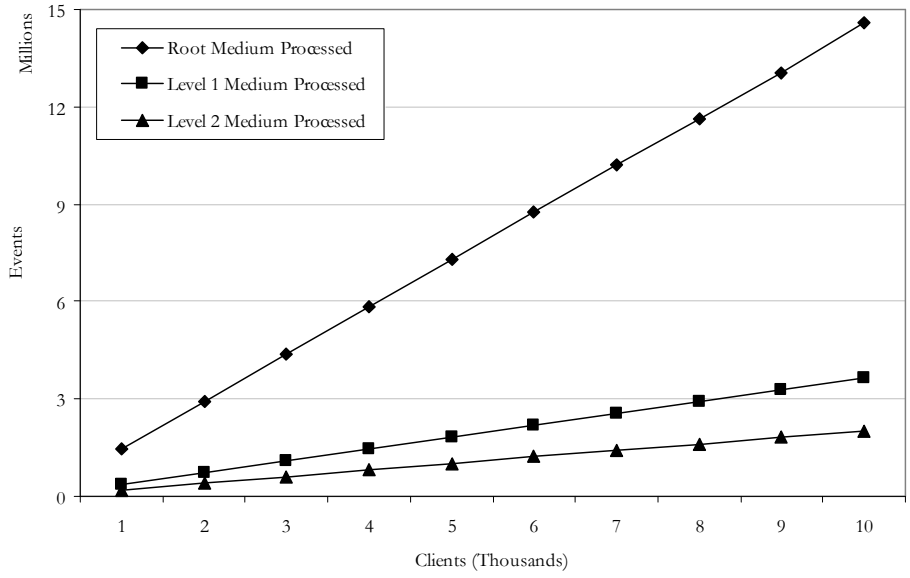

Figure 40: DNS un-cached lookups medium per server

DNS Un-Cached Updates

Updates in the un-cached DNS system, shown in Figure 41, are all done at the level 2 and only there, it grows linear, which is to be expected. Only one processing event per update is done, since an update is sent to one location where it is received and processed, and then it is done.

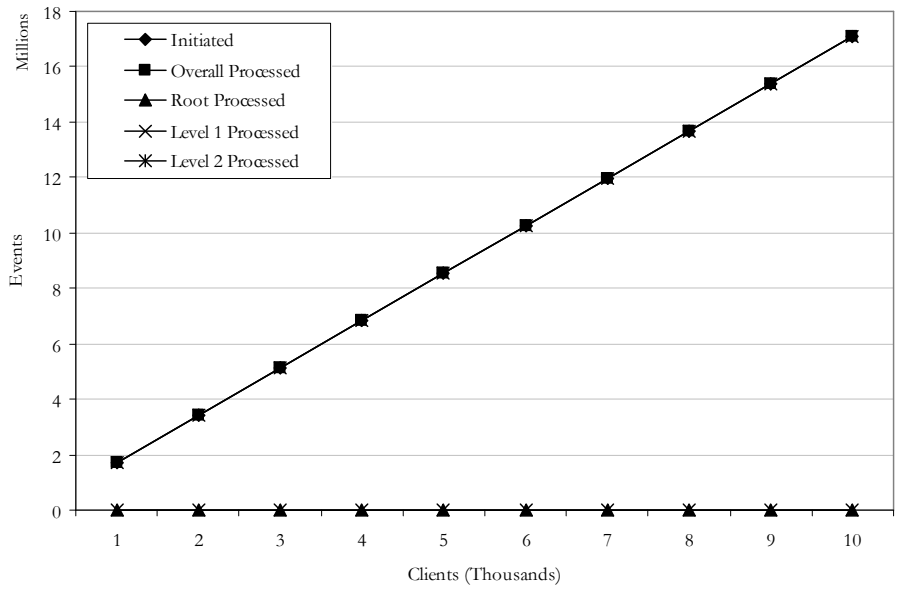

Figure 41: DNS un-cached updates

As the number of users grows in the system, the number of updates at level 2 will grow. Figure 42 shows the medium growth for one server node.

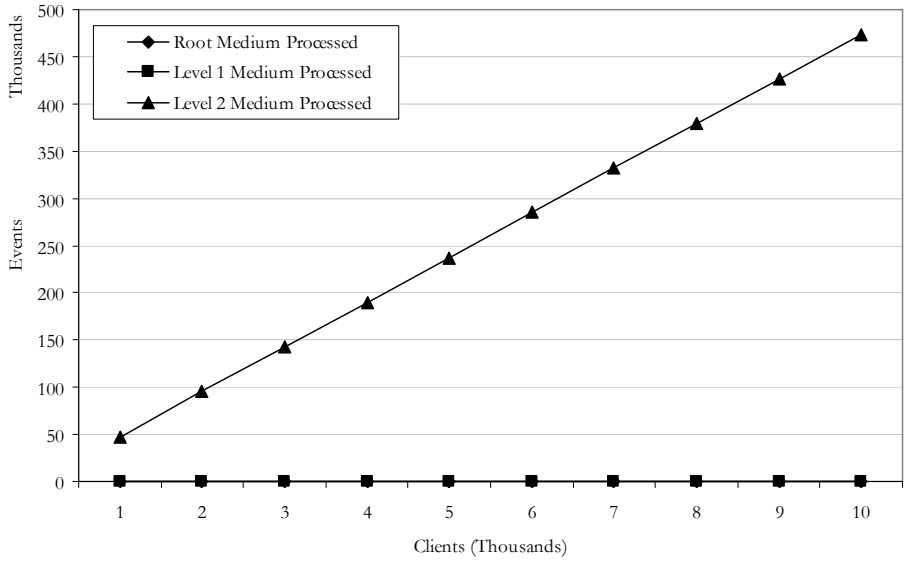

Figure 42: DNS un-cached updates medium per server

Lookups and Updates

Figure 43 shows that when there is not any caching in the DNS system is works a bit differently, there is first a lot more traffic going on, about 50 % more, but still the absolute majority is processed at the level 2 servers.

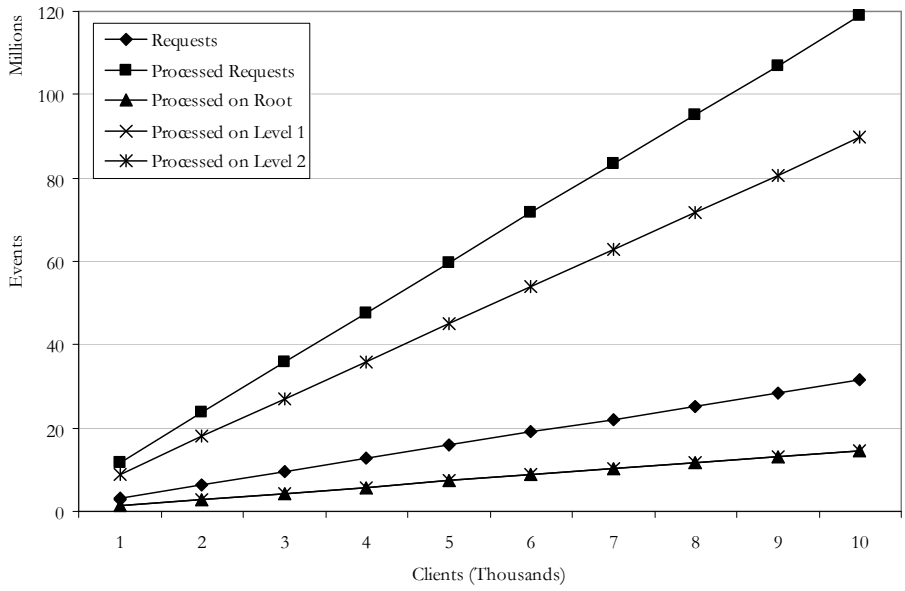

Figure 43: DNS un-cached lookups and updates

The medium per server per level, depicted in Figure 44, shows that the load scales about the same on level 1 and level 2. While the load on the root increases much faster. This is not a good factor since it shows that the root will a lot sooner be overloaded than other servers in the system.

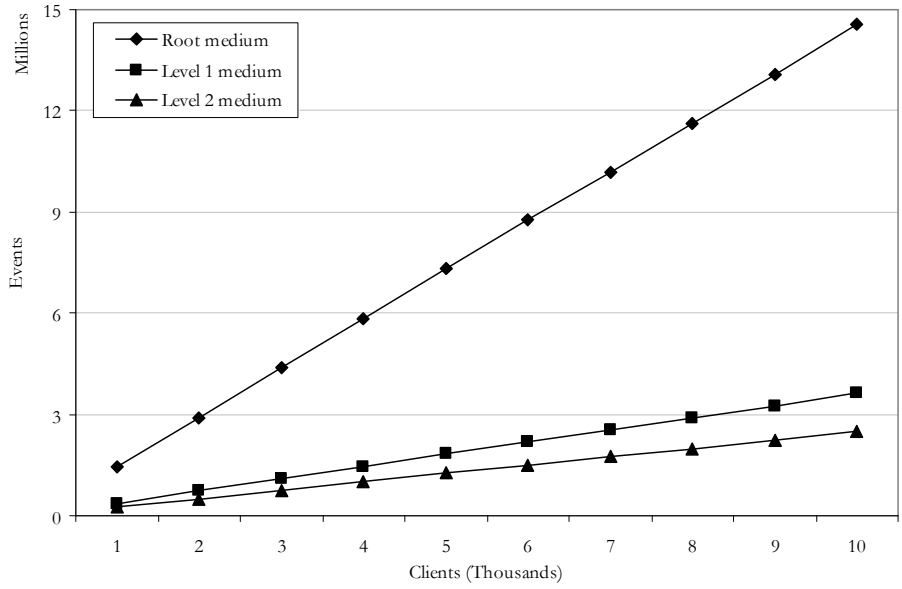

Figure 44: DNS un-Cached lookups and updates medium per server

DNS Comparison

It can clearly be seen that the DNS system has severe scalability problems at the root server. However, this problem is currently countered with the use of cache but it will at some point not be enough any more.

5.3 Chord Based Name to Address Resolution

This sub section provides the analysis of the results obtained when running the Chord simulations. During the simulations of the Chord system no cache was

enabled, since the effects of caching in these Chord simulations only affect the path length in rare cases. This could occur when routing requests reaches a node that is immediate successor of the responsible node, i.e. the responsible nodes predecessor which a copy of a cache record is sent to.

5.3.1 Validation of the Simulation Model

To verify that the implementation of the Chord naming system worked correctly some test simulations where performed. These results should be similar to the results obtained during validation in the previous thesis work [23].

In Figure 45 the average path length is shown. These results show the average number of hops in the Chord ring during a lookup. It is seen that the average path length (y-axis) is a function of the Chord ring size. The size of the ring ranges from 32 nodes to 2048 nodes (x-axis). These results agree with the ones found in the previous work.

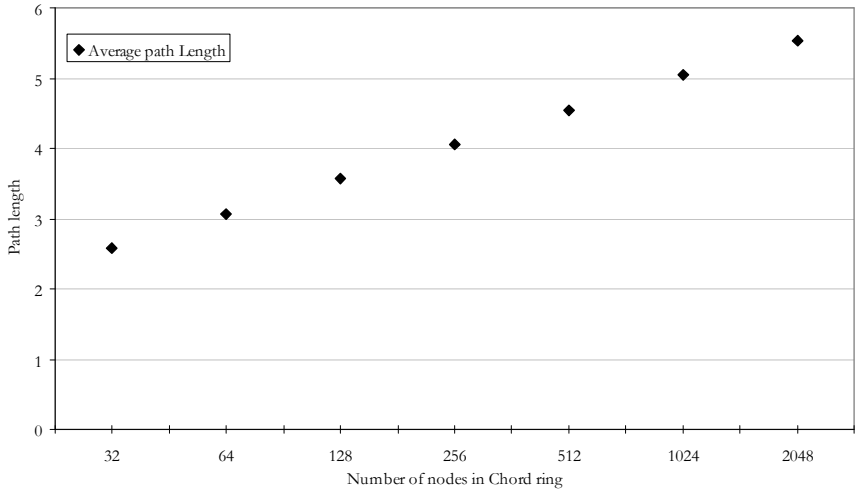

Figure 45: Average path length during different Chord ring sizes

Figure 46 shows a histogram over the path lengths that are observed when having a ring size of 4096 nodes. This result is similar to the Probability Density Function produced in the previous thesis [23], which shows the frequency of different path lengths.

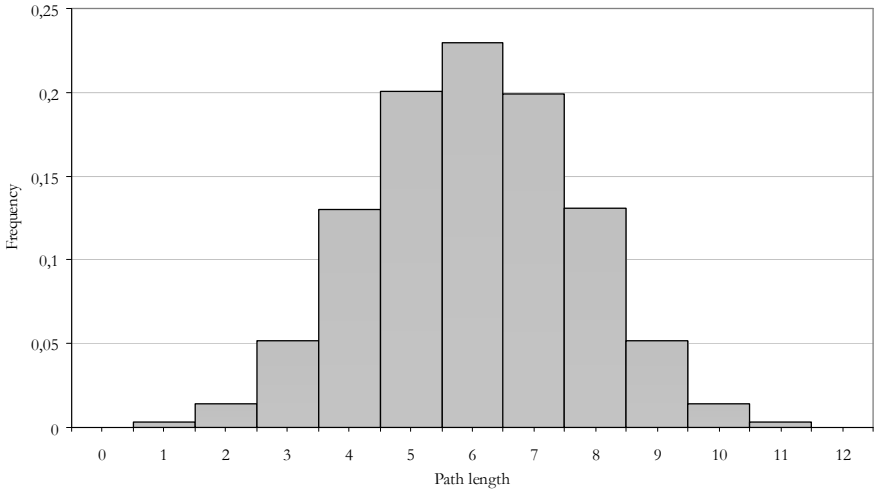

Figure 46: The frequency of different path lengths in a ring with 4096 nodes

5.3.2 Simulation Results

CPU Demands

In the figures below, Figure 47 and Figure 48, the number of packets sent from clients towards the Chord naming system is compared to the number of packets processed in the naming system. The processing of packets regards the operations made on a packet in the naming system. A packet containing a request from a client could trigger three different actions in the Chord naming system. The first one that could occur is that the receiving node is not responsible for the lookup- or the update-request, thus routing it towards the responsible node. Second action is when a responsible node receives an update request and stores the information in the packet. The last action to occur is when a responsible Chord node receives a lookup request which the nodes replies to by sending a packet back to the Chord node that first received the request. Each nodes action is regarded as a processing of a packet, thus the curve with squared data points ("Measured no of packets processed by Chord Ring") shows the number of messages handled by the naming system.
The last curve with triangular data points ("Ideal no of packets processed by Chord ring") shows the number of processing's in an ideal case, i.e. when the number of hops in the Chord ring is equal to the average number of hops, see Figure 45.

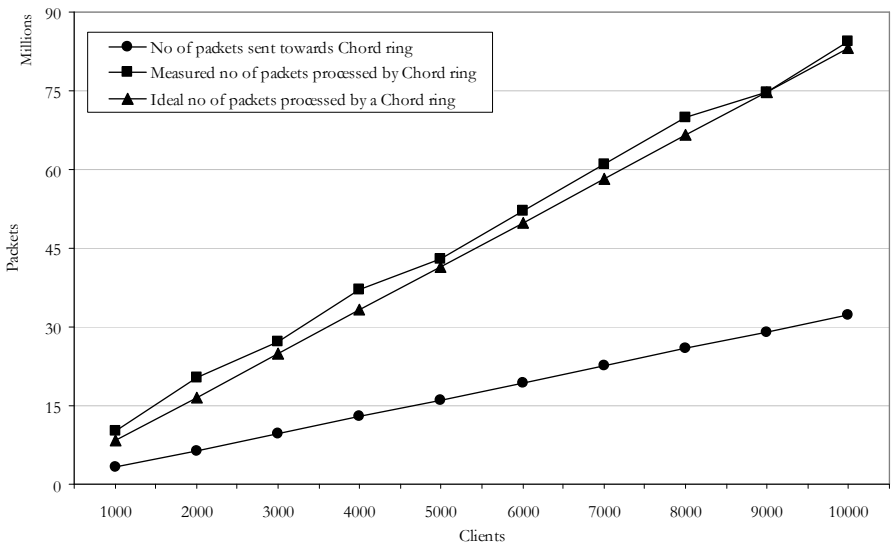

Figure 47: The total number of packets sent compared to the total number of packets processed in the naming system with 32 nodes

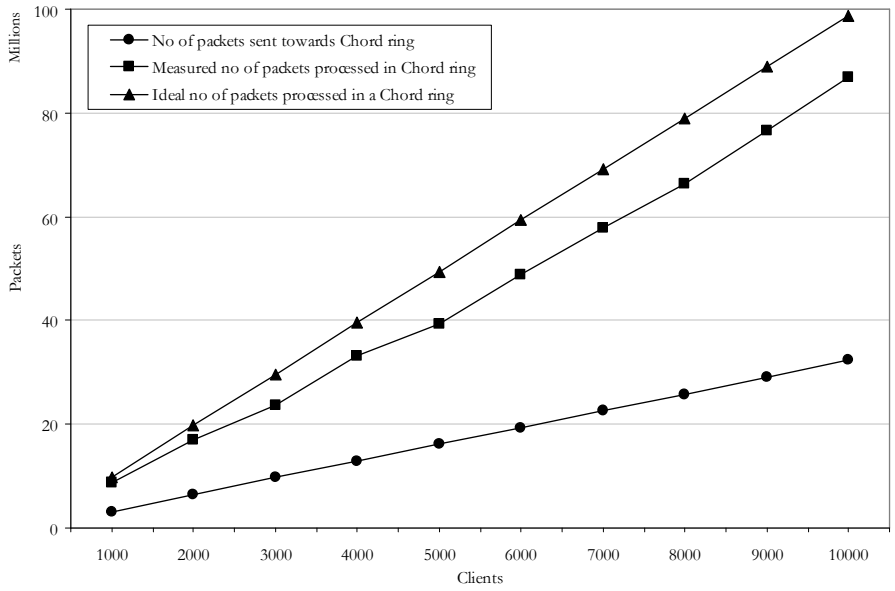

Figure 48: The total number of packets sent compared to the total number of packets processed in the naming system with 64 nodes

The thing that is interesting, but not that surprising, to notice in the two figures above is that the load on the naming system is roughly a function of the total number of packets sent towards the naming system times the average hop count in the system. In Figure 48 the measured number of packets that is processed by the Chord naming system ("Measured no of packets processed in Chord ring", squared data points) is slightly smaller than the ideal case ("Ideal no of packets processed in Chord ring"), which is due to a large finger table in our implementation. Remember that the size of the finger table is the number of bits in the key space, so when the number of clients increases the larger key space will make the finger table grow, but the size of the ring is constant, that is the size of the ring is 32 nodes or 64 nodes. So, when the key space increases each node in the ring is able to route messages further into the ring. The long term effect of this is that the total number of packets processed in the ring is slightly decreased.

Below, in Figure 49, it is seen how the load on one node in a Chord ring increases as the number of clients increase in the network. The diagram shows that each node has a higher load if the ring size is smaller, even tough the total amount of processed packets is higher in the case with a larger ring due to the extra number of hops. This means that an increased ring size reduces the load on each chord node, but it increases the average number of hops in the ring thus increasing the lookup latency.

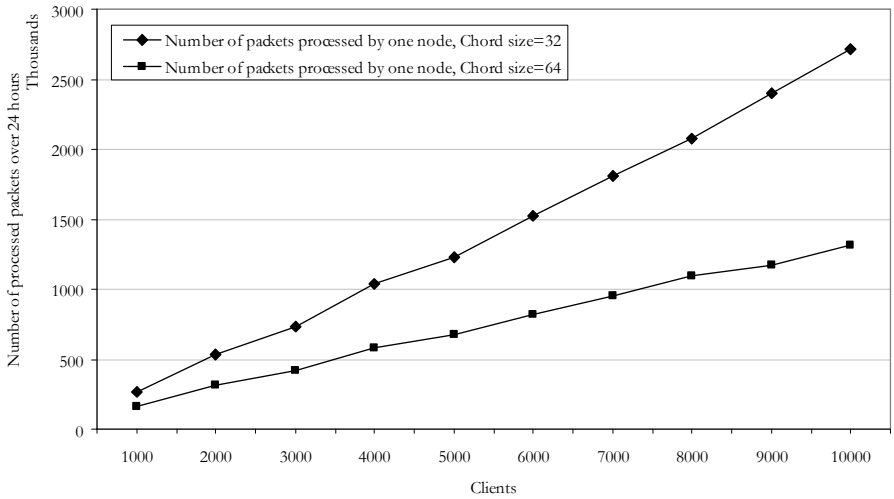

Figure 49: Number of packets processed by one node in ring

Updates

The number of updates that is sent to the Chord naming system is shown in the figures below, Figure 50 and Figure 51, with the curve "No of updates sent towards Chord ring". The number of updates that is processed by the Chord ring, as explained above, is shown with the curve "Measured no of packets processed by Chord ring". Compared to the measured number of processed packets is ideal case labeled "Ideal no of packets processed by Chord ring". This curve shows the amount of work in an ideal case where the number of hops in the ring is equal to the average path length as depicted in Figure 45.

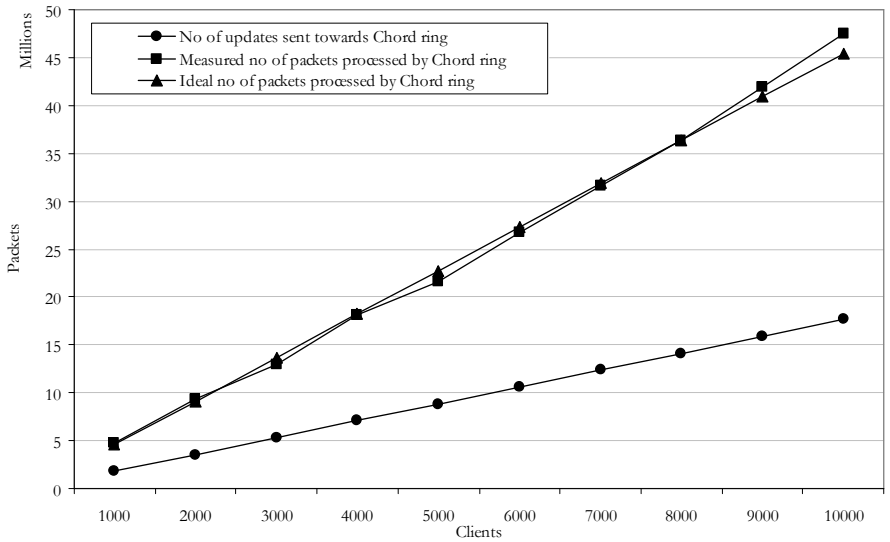

Figure 50: The total number of update packets sent compared to the total number of updates processed in the naming system with 32 nodes

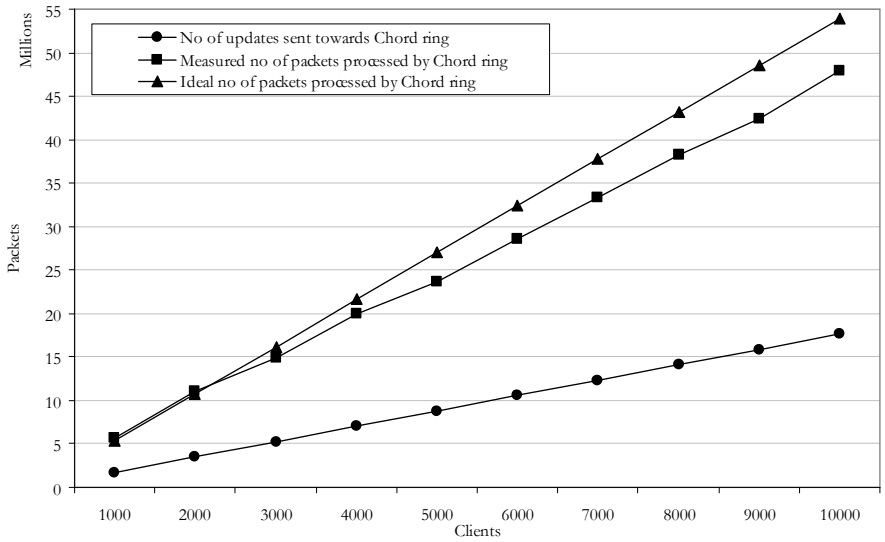

Figure 51: The total number of update packets sent compared to the total number of updates processed in the naming system with 64 nodes

Below, in Figure 52, it is seen how the load due to updates is on the Chord naming system. The diagram shows that each node has a higher load if the ring size is smaller ("Number of updates processed by one node, Chord size=32"), even tough the total amount of processed packets is higher in the case with a larger ring due to the extra number of hops. To conclude this it means that an increased ring size reduces the load on each ring, but a larger ring gives a higher total amount of processed packets and larger update latency.

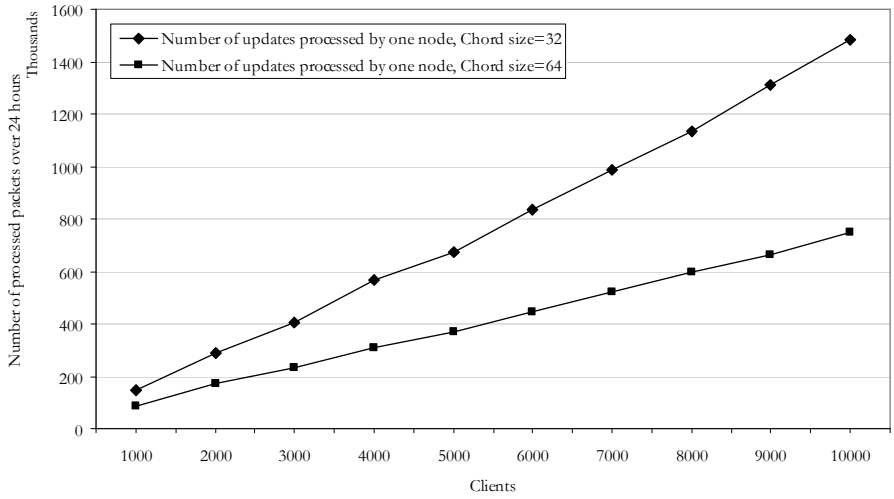

Figure 52: Number of updates processed by one node in ring

Network Load

To be able to calculate the amount of traffic sent in the network due to lookups and updates, the sizes of these packets needs to be derived. The structure of a Chord message packet is seen in Table 6. In the packet three different data types is used: integers, double, and strings. The integer takes up 2 bytes, a double uses 8 bytes, and a string in the OMNeT++ simulator is a combination of a character pointer and a number of characters where a char pointer takes 4 bytes and each char takes 1 byte. Each packet is sent inside an UDP packet and which in turn is encapsulated in an IP

packet. The UPD overhead is 8 bytes and the IP overhead is 20 bytes, as seen in section 3.9.

Table 14: Data types used in Chord packet

C++ Data Type	Chord Packet Field
integer – 2 bytes	packetName, hopCount, lookupAlgorithm, messageType.
double – 8 bytes	sendTime.
string – 4 + n1 bytes	packetSource, chordSource, symbolicName, nextHop, resolvedAddress.

The packet fields that uses strings as data types contains IP addresses on the form 10.0.x.y, except for the field *symbolicName* that contains the symbolic name coupled with the IP address as explained in section 4.3.1. It has been decided, for the sake of the calculations, that a typical IP address will have two digits for x and three digits for y. This gives that a typical IP address consists of 10 characters. Together with the char pointer this gives that a typical string is 14 bytes. The reason for having IP addresses stored in strings is an implementation issue, as they could have been stored as integers or in other special purpose data structures.

When sending a lookup packet from a client towards the Chord ring, the client does not fill all the fields in the packet. The fields filled in by the client are the *symbolicName*-field that contains the name of the searched resource, and *packetSource*-field that contains the clients IP address. But the empty fields still contains the char pointer. So, the packet size of a lookup packet leaving a client is: $4\times2+1\times8+5\times4+2\times10=56$ bytes, adding the UDP and IP header give a packet size of $56+8+20=84$ bytes.

The answer for a lookup request has the three additional fields filled with data. This gives a packet size of: $84+3\times10=114$ bytes, including UDP and IP header.

An update request from a client to the naming system contains almost the same fields as a lookup packet from a client. The only difference is that the field *symbolicName* contains the client's name. The size of an update packet is then $84+10=94$ bytes, including UDP and IP overhead.

The Table 15 contains a summary of the different packet sizes seen in the Chord naming system during simulations.

Table 15: Size of different packet types including UDP and IP overhead

Packet Type	Packet Size
Lookup request	84 bytes
Lookup answer	114 bytes
Update request	94 bytes

Web sessions

During a typical web session 20×8 lookups are made according to Table 1, and the number of answers, sent back to the access points, is equal to the number of lookup requests, i.e. the number of packets on the link between the naming system and the client is equal to twice the amount of initiated lookups by the client. The total amount of control plane traffic for a typical web session is then: $20\times8\times84$ +

20×8×114 = 31680 bytes (≈30.9 kB). Each web session consist of downlink traffic and uplink traffic, using the numbers in Table 1 the total amount of data sent during a typical web session is equal to 20×8×25×1024+20×8×145=4119200 bytes (≈3.9 MB). This figure has both UDP and IP overhead included.

Figure 53 shows how the amount of web traffic on the user plane increases as the number of clients increases; this is compared to the growth of the control plane traffic that is generated during a web session. The figure has two axis's where the amount of web traffic ("Total amount of web traffic during web sessions", triangle data points) should be read against the left y-axis, and the amount of lookup traffic ("Lookup traffic due to web sessions", squared data points) has its y-axis to the right in the diagram. The values from the traffic include both UDP and IP overhead.

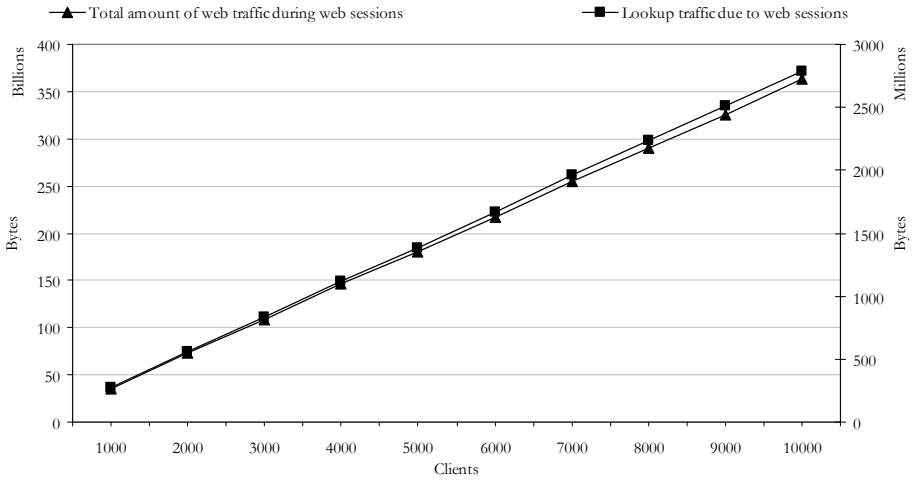

Figure 53: Traffic in the network due to web sessions and the amount of traffic from lookups during web sessions

Figure 54 shows the ratio between the amount of lookup traffic and the total amount of network traffic in a web sessions. It is seen that the ratio is the same over all the simulations, from 1000 clients to 10000 clients. This shows that even though the amount of clients in the network increases thus increasing the traffic in the network, the relative amount of lookup traffic is unchanged.

The total amount of network traffic is the sum of lookup traffic and the corresponding answers to these lookups, and also the user plane traffic. All calculations for these values are found above. The get the ratio for the figure the amount of lookup traffic is divided with the total amount of network traffic.

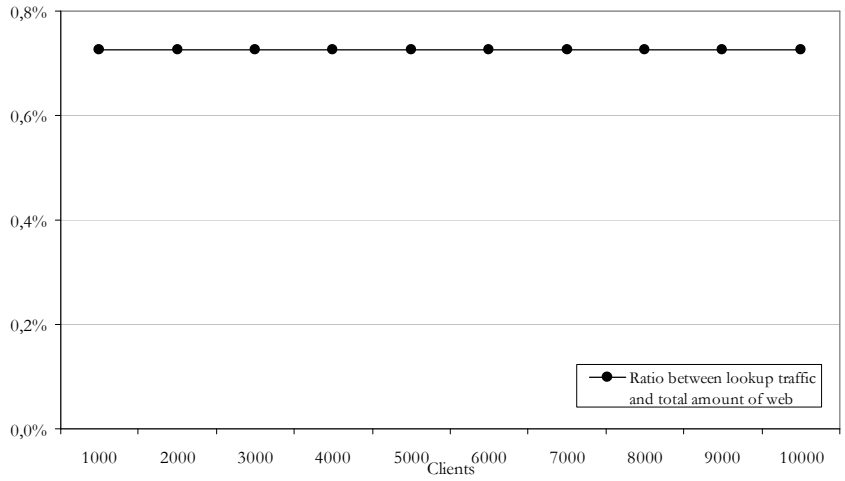

Figure 54: Ratio between lookup traffic and total amount of web traffic

Streaming traffic

During a streaming session typically 10 lookups is made, which is seen in Table 2, which yields a total control plane traffic amount of 10×84+10×114=1980 bytes (≈1.9 kB), which includes both UDP and IP overhead, per session. In Table 2 there exist three different types of streaming sessions. For this diagram the biggest one is used, which is the audio and video streaming sessions (74 %). The amount of user plane traffic during a streaming session, which consists of both audio and video, is calculated in section 3.9.2

The following figure, Figure 55, illustrates the relation between the total amount of user plane traffic and the total amount of control plane traffic and how they follow each other when the number of clients in the network increases. The amount of streaming traffic ("Total amount of streaming traffic during streaming sessions", triangle data points) should be read against the left y-axis, and the amount of lookup traffic ("Lookup traffic due to streaming sessions", squared data points) is read against the right y-axis.

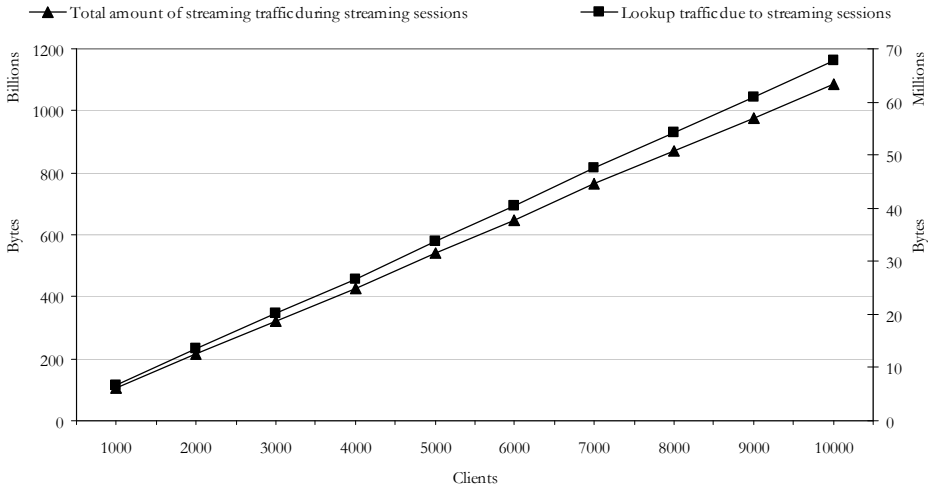

Figure 55: Amount of traffic in the network due to streaming sessions and the amount of lookup traffic

The following figure, Figure 56, shows the ratio between lookup traffic and the total amount of traffic due to streaming sessions. It is seen that the relative amount of lookup traffic does not change when the number of clients in the network is increased.

The total amount of network traffic is the sum of control plane traffic and user plane traffic. All calculations for these values are found above.

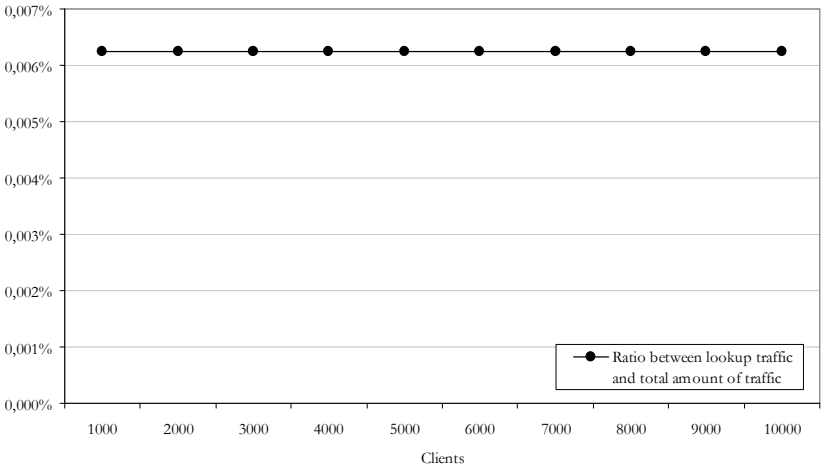

Figure 56: Ratio between lookup traffic and total amount of streaming traffic

Telephone traffic

When a client initiate a call only one lookup is made which gives a control plane traffic amount (including both UDP and IP overhead) of 84+114=198 bytes (≈0.2 kB), using the values in Table 15. This value is the amount of control plane traffic per session. Figure 57 shows the relationship between the total amount of traffic in the network during telephone conversations and the amount of traffic from the lookups made when the calls are initiated. The amount of telephone traffic ("Total amount of telephone traffic in the network", triangle data points) should be read against the left y-axis, while the amount of lookup traffic ("Lookup traffic due to telephone calls", squared data points) should be read against the right one. The curve showing the total amount of traffic is based on the number of lookups generated from telephone traffic and the data rate calculated in section 3.9.3.

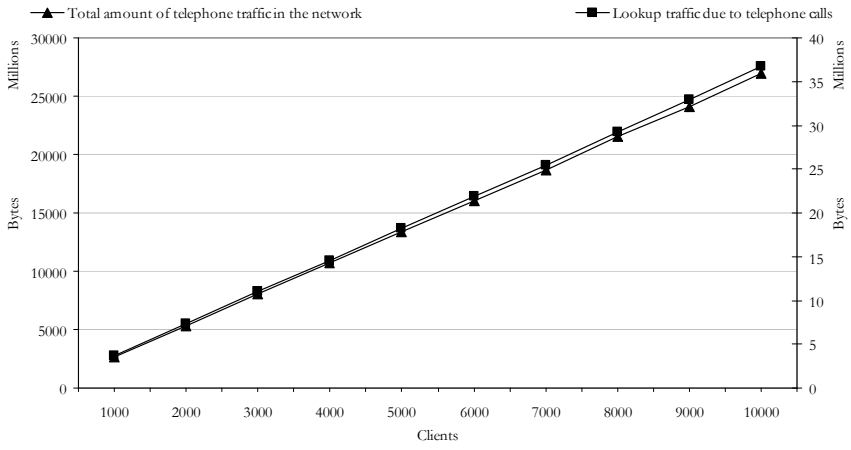

Figure 57: Amount of telephone traffic in the network compared to the amount of traffic generated by lookups

Below, it is seen that the relative amount of lookup traffic due to telephone sessions in the network is constant. In Figure 58 the ratio is unaffected due to the fact that the number of client's increases, leading to that the total amount of traffic in the network is increased.

The total amount of network traffic used for this figure is based on the values above. The sum of control plane traffic and amount of telephone traffic yields the total amount of network traffic.

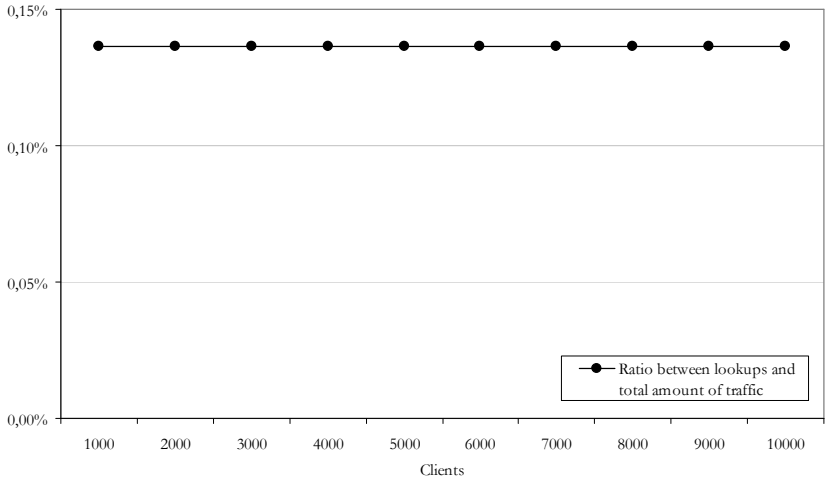

Figure 58: Ratio between lookup traffic and total amount of telephone traffic

5.4 Path-Coupled Name to Address Resolution

Here the results for the Path-Coupled methods simulations are presented; this section also shows how it scales and its problems and perks. The method will be analyzed in terms of traffic to the whole layer and traffic in medium to each server on different levels. With medium it is meant the total traffic volume of that layer divided on all the servers of that layer.

Lookups

It can be seen in Figure 59 below that the ratio between created requests and processed requests are 1 to 1 comparing on all levels. This means that a request visits all levels once and only once when it is being resolved. This also explains why the total amount of processed requests is three times higher then created. Factors to consider here is that the higher the level the fewer the machines are, and therefore the real load is higher. This will be clearer in the next paragraph where the medium load of a machine is presented.

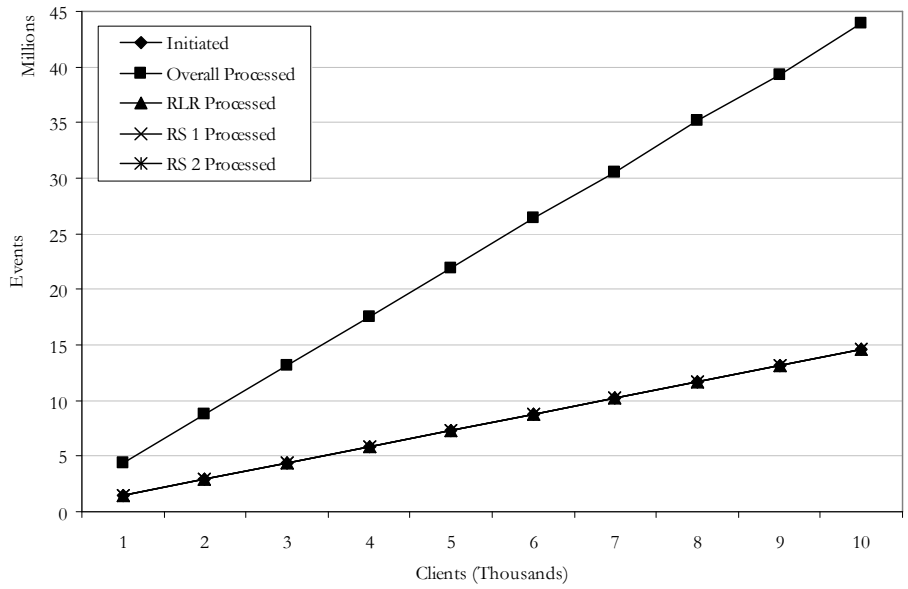

Figure 59: Path-Coupled lookups

Figure 60 below shows the medium load on a single server of that layer. It can be seen that the lower levels scales far better then the RLR since the medium values are growing at a much slower rate.

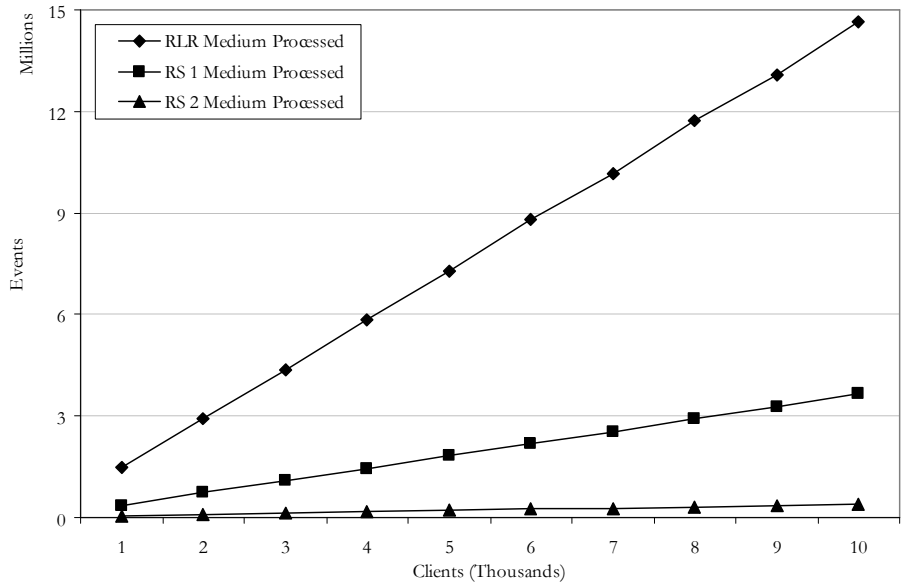

Figure 60: Path-Coupled lookups medium per server

Updates

Below, Figure 61 shows that the total updates coming to the different levels of the system. It is to be noted that most updates takes places on level 2, the second most on level 1 and the least often on the RLR. This is to be expected since most of the movements usually take place within an AS or between two AS's.

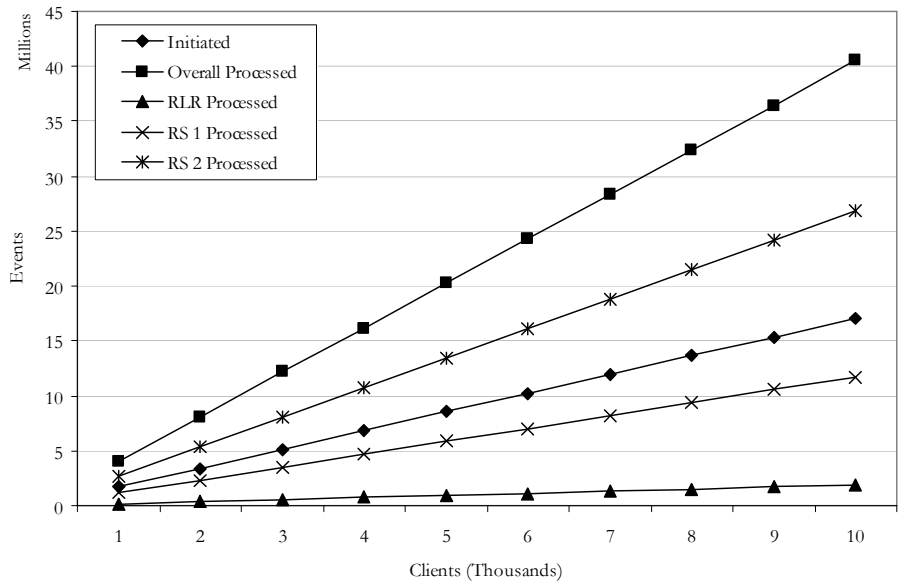

Figure 61: Path-Coupled updates

The medium however, visualized in Figure 62, shows that the RS1 servers are the most heavily loaded. This is due to movements since one RS1 have many AS's under its control, it will be in charge of all the movements in between them. This can be observed in the diagram below.

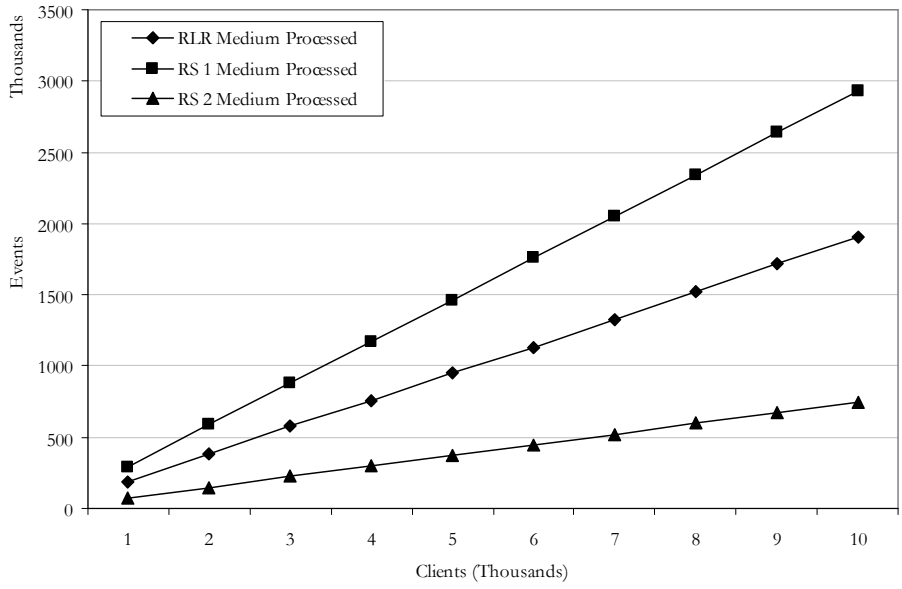

Figure 62: Path-Coupled updates medium per server

Overview

Both updates and lookups in Figure 63 below and it can be observed that most of the traffic goes to the lower parts, which is a really nice factor for scalability since there are more servers at level 2 then the higher ones to share the load.

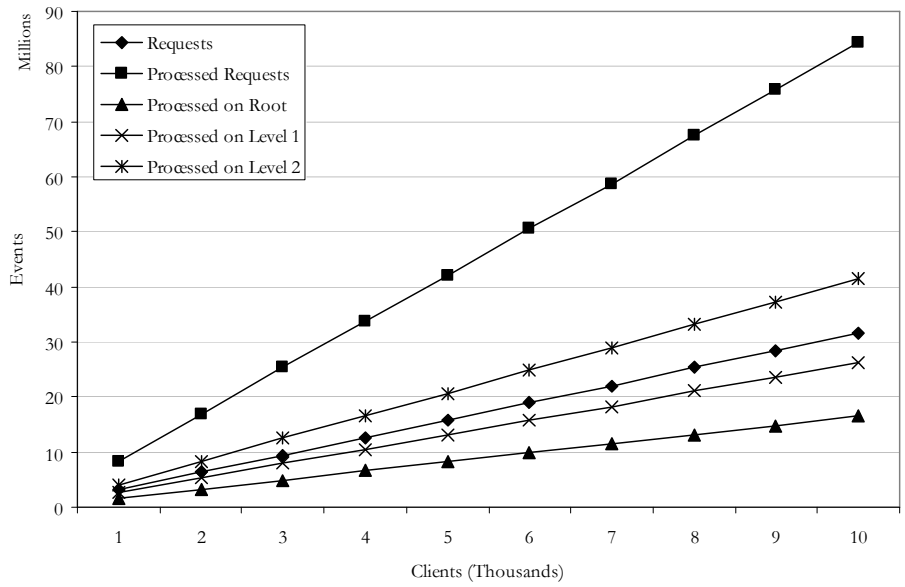

Figure 63: Path-Coupled lookups and updates

The medium value, Figure 64 below, presents what was discussed a bit in the previous paragraph, the load on the RLR is the highest and that means that the RLR might be a point that suffers from poor scalability, the RS of 1 has the same problem but a lesser version of it. If it is a problem will be known when it is compared to the other methods in a later section.

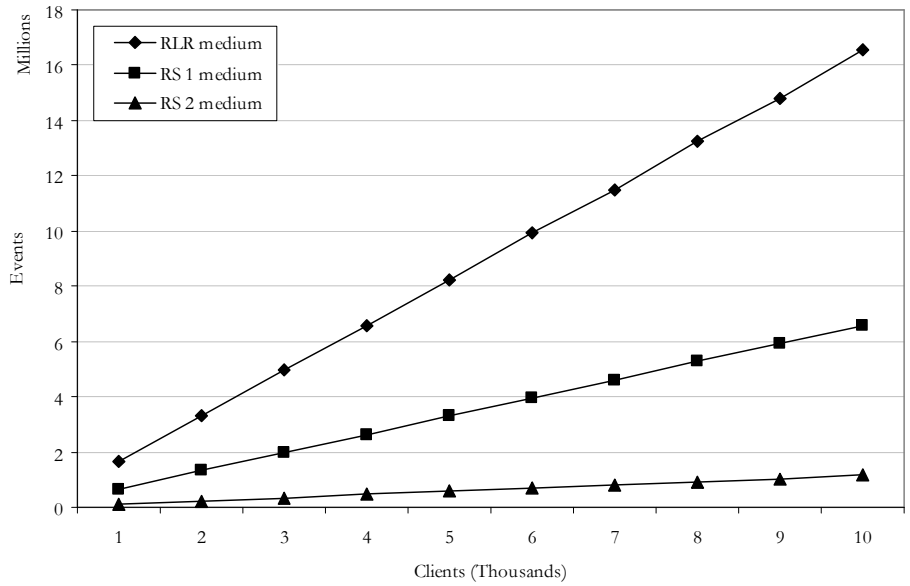

Figure 64: Path-Coupled lookups and updates medium per server

5.5 Memory Usage

The different methods uses very different ways to spread out the data in the network, and this section provides an analysis of what effects that has concerning scalability.

The DNS system stores all references to the final hosts in its lowest part, and in the higher levels only references are stored.

A calculation for the simulated system in this thesis is that DNS would store four entries at the root level, 36 entries in the level 1 and the client entries at the lowest level. This means effectively that it is great concerning scalability of memory, since there are a lot more servers at the lowest level sharing the memory. However, caching can put demands on the lowest level, thus making it to store un-proportional amounts of data. This can be countered in implementations by losing up on the demands of keeping cache data stored. The only problem is that it gives an overall increased load on the system. Understanding the effects of caching is quite demanding and will therefore not be addressed more thoroughly in the thesis. Below follows an illustration of the memory distribution as well as some memory calculations behind it, see Figure 65. It illustrates that the root on the top needs to store very little information in comparison to the lower layers.

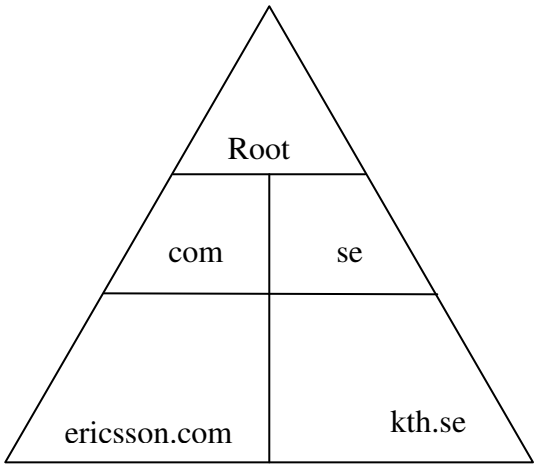

Figure 65: Visualization of storage demand in DNS

If n is the number of clients in the naming system, m is the number of level 2 servers and k is the number of level 1 servers. Then the root will have k entries, a level 2 server will have m/k entries in medium, however, it might vary from 0 to m entries. A level 2 server will have n/m entries; also here it might vary from 0 to n entries.

In the Chord system each node will in theory handle an approximately equal amount of names. This amount is the total number of hosts divided by the amount of nodes; however, if some node is not present that nodes host names will become responsibility of another node, causing this node to have a double amount of names. If more nodes would be missing and they would all fall under the responsibility of the same node the memory demand on it would increase dramatically. This is absolutely a danger and a factor that impairs scalability. The following figure below, Figure 66, illustrates this problem; also some calculations are also given below.

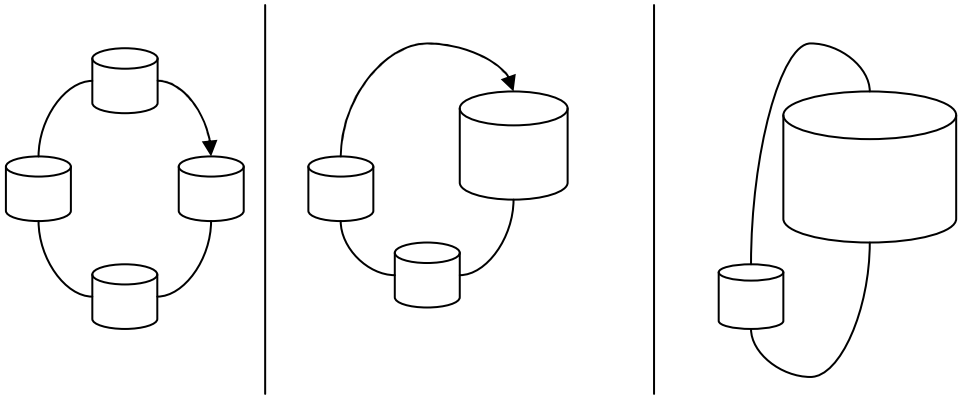

Figure 66: Visualization of chords storage problem

If n is the number of clients in the naming system and m is the number of nodes the ideal case would give n/m entries at every node, however because of mechanism that makes a node take over responsibility for missing nodes this value can vary between n/m to n entries.

The Path-Coupled method stores as many entries that are below the server in the structure. This means that the RLR stores all entries in the whole system, while a local RS only stores the nodes in its area. This would mean that the memory demand would look like an up side down triangle, with the RLR. This is not good scalability wise. Below follows an illustration, Figure 67, of the storage demand as well as memory calculations.

<table>
<tr><td colspan="4">RLR</td></tr>
<tr><td colspan="2">RS of A</td><td colspan="2">RS of B</td></tr>
<tr><td>RS of A.V</td><td>RS of A.X</td><td>RS of A.Y</td><td>RS of A.Z</td></tr>
</table>

Figure 67: Visualization of storage demand in the Path-Coupled method

If n is the total amount of names in the system, m is the total number of servers at level 1 and k is the number of servers at level 2. Then the RLR will store n entries. A level 1 server will store in medium n/m entries, however, it might vary between 0 and m entries. And at a level 2 server there will in medium be n/k entries, however, this might also vary between 0 and n entries.

All three methods place a need on a server to be able to store all n entries. This is not very good because there might be some problems if all servers should be able to handle all n entries, e.g. problems in form of demands on hardware, it might not be economically viable. However, in the DNS case it is possible to plan away this demand since the structure and name distribution remain static. In Chord and the Path-Coupled method it is different, there it can not be planned because in the Chord it depends on how many nodes that are present and in the Path-Coupled method it depends on where the clients are located and they can potentially be located at the same place. All this concludes that DNS has the only fully

deterministic sustainable way of distributing the need for memory, while the others must rely on statistical properties such as reliability/availability and mobility patterns respectively.

There is also another aspect of the Path-Coupled method; in its solution it stores requests in routers along the way while it requests routing information from the RLR/RS. It was investigated during the simulations how many requests that was stores in maximum at each and every router. The figure below, Figure 68, presents the maximum ("Maximum", squared data points) of these values and the sum ("Sum", triangle data points) all routers maximum need of memory. This is to show the trend of needed memory as the amount of users grows.

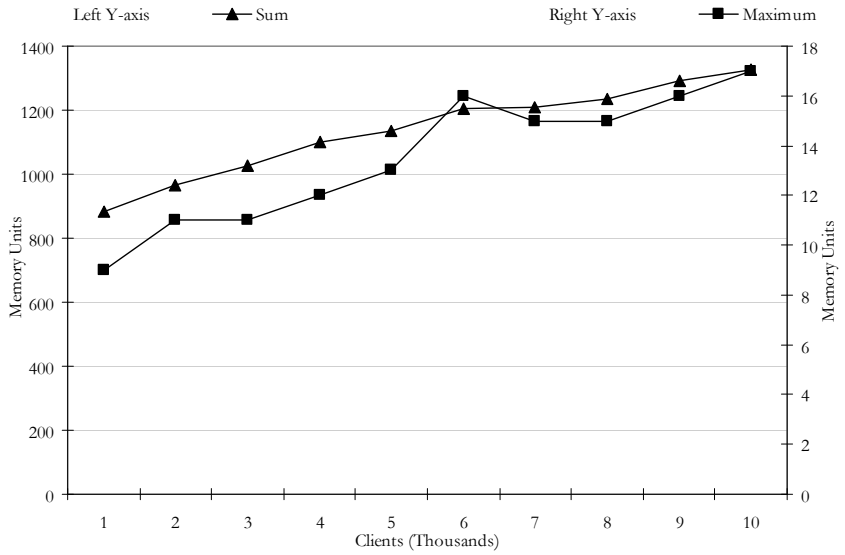

Figure 68: Path-Coupled router memory usage

The increased need for memory grows quite slowly which is a nice factor. It gives a finger waving that the memory does not have to be that big to make the method work.

5.6 CPU Usage

In the following section the CPU usage during busy hours is presented. The Chord system was compared only to the lowest layer of the other two methods that is hierarchical because this was the only way to make it as fair as possible.

The CPU usage is highly dependent on the implementation and therefore these results will be based upon processed events per second so the diagrams in this section give the events processed per second on the whole on the different level of the system. The medium values are not used in this section because they would because it is not needed to gain an understanding of how the CPU usage will scale.

The CPU usage on the RLR and un-cached DNS root scales linearly as can be seen in the diagram below (Figure 69), while it remains constant in the cached DNS solution with cache.

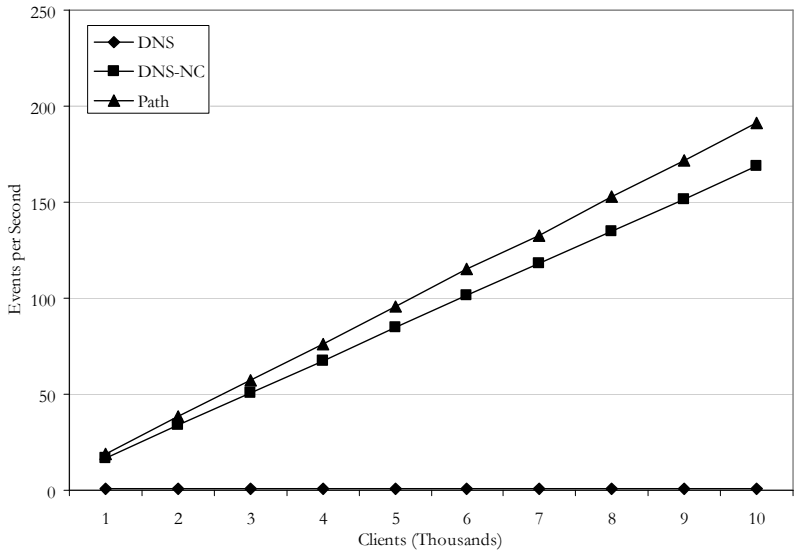

Figure 69: Requests per second at root

It is the same for the middle level as well, shown in Figure 70; the increase is linear in the un-cached DNS and Path-Coupled methods, while remaining constant in DNS. However, in this case the Path-Coupled methods need for CPU increases faster than the un-cached DNS method.

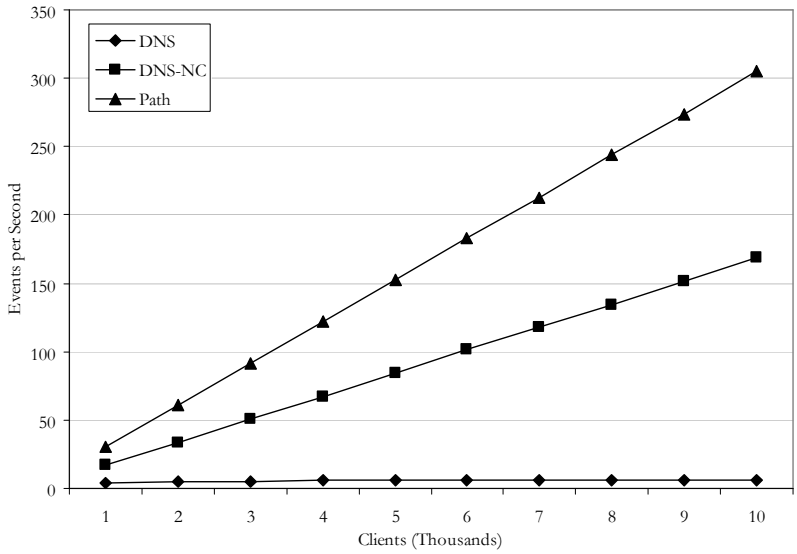

Figure 70: Requests per second at Level 1

It is interesting to see below in Figure 71 that the Path-Coupled servers are far less processing intensive then the others, for level 2. It is also interesting to see that the Chord nodes are as processing intensive as the un-cached DNS servers.

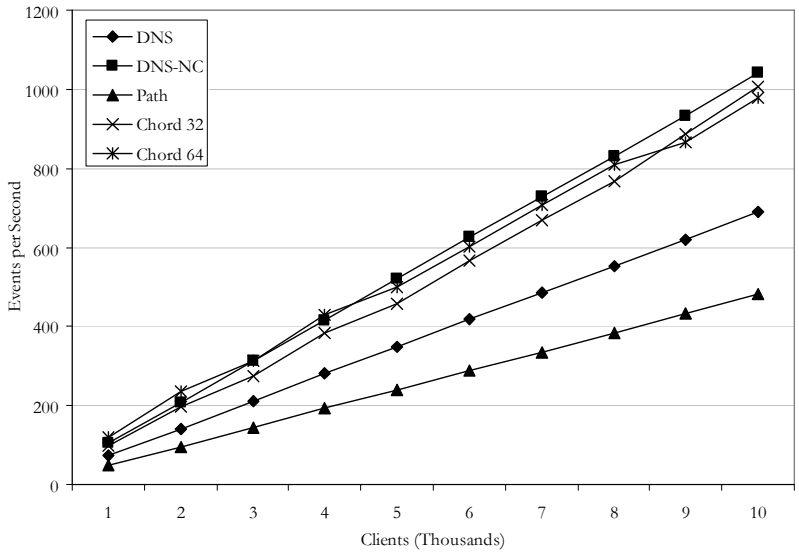

Figure 71: Requests per second at Level 2

5.7 Comparison of the Results

In this section there is an extensive analysis of the overall results, and a comparison of the three methods examined. This section shows how the three different methods compare to each other in terms of scalability when having a high number of mobile entities.

In the figure below, Figure 72, the traffic volumes for the whole system is shown. And it can be seen that most of the traffic takes place towards the lower parts of the systems; it is either more or equal to the traffic in the higher layers. This is a good property since often the higher layers consists of fewer servers than the lower ones does.

It can be noted that the RLR, of the Path-Coupled method, has about the same traffic volume as the un-cached DNS root, while the cached DNS root has significantly less traffic to it. The middle layer looks about the same, with the difference that the Path-Coupled method performs a bit worse than the un-cached DNS, this due to the fact that the Path-Coupled deals with updates which the DNS system does not. In the lowest layer the Path-Coupled method performs the absolutely best while the different Chord versions and the un-cached DNS perform the worst. However, even though they might have bigger traffic volumes it does not necessary mean they scale poorly, as will be seen in the next paragraph when the medium traffic volume per server is investigated.

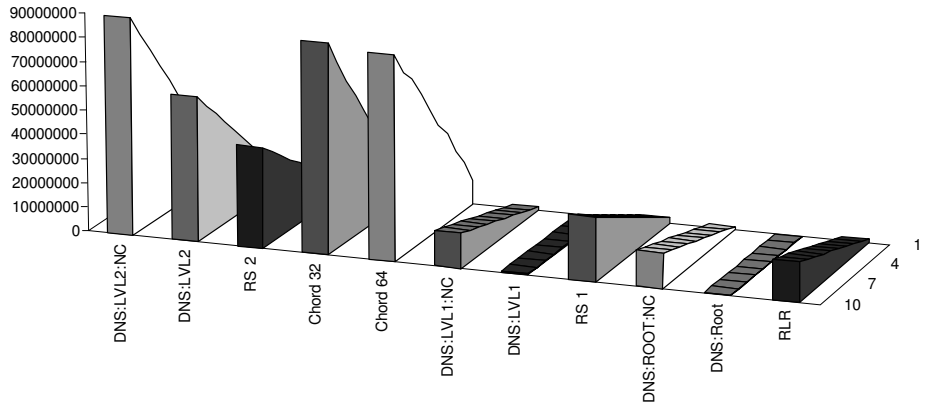

Figure 72: Traffic volume to different parts of the systems

Figure 73 shows the medium load of a single server at the different parts of the systems. This diagram visualizes that RLR and un-cached DNS root servers are heavily loaded and are therefore a chokepoint, while the (cached) DNS root performs exceptionally well. In the middle layer it is shown that the DNS solutions are better then the Path-Coupled, this hugely has to do with update traffic, since update traffic never comes to this level in the DNS solutions. At the lowest layer the Path-Coupled performs the best tightly followed by Chord with 64 nodes, and it shows a trend that the traffic in Chord becomes less if there is more nodes, however, more nodes would in a real world scenario most definitely mean more AS's and thereby more users. It is also to be noted that the DNS servers is slightly better then the Chord solution of 32 nodes.

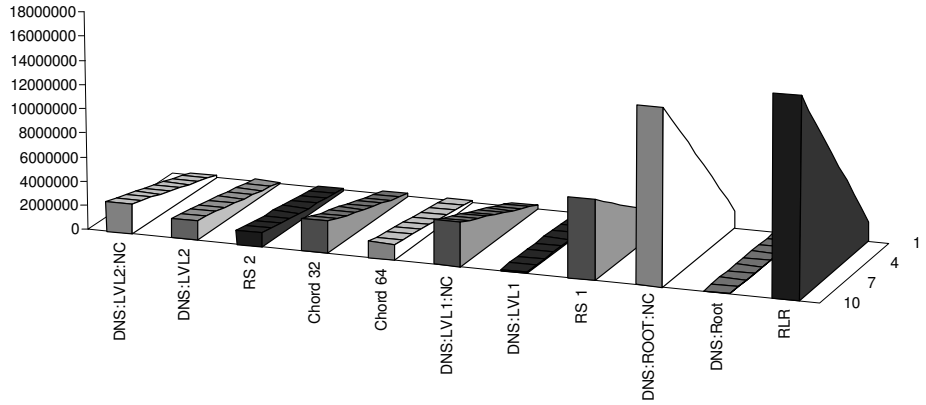

Figure 73: Medium traffic volume per server

Overall it shows that the Path-Coupled method is almost equivalent with the un-cached DNS system, and that it competes well against the Chord system on its lowest levels, while the DNS system with cache scales well.

5.8 Analysis of RLR Distribution

As mentioned in sections 2.5 and 3.10 the aspects regarding the distribution of the RLR is to be investigated. During the implementation the RLR has been represented by a single entity, but as the load increases the distribution of it needs to be considered. When the Path-Coupled method is deployed in a real world global scenario the number of connections against RLR will be in the magnitude of millions per hour.

In this work two different possibilities regarding the distribution of the RLR have been investigated: to let the RLR be built upon the DNS method, or distribute the RLR on a DHT-based method and in this case the Chord protocol.

5.8.1 RLR Distributed Using a DNS Solution

The first distribution aspect to be examined is the one based on the DNS method. As seen in the diagrams above, Figure 59, the number of lookup requests that reaches the RLR is numerous. Each time a client wants to connect to a resource in the network a name-to-address query has to be sent to the RLR. This implies that the load on the RLR is significant from a lookup perspective. When looking on the number of updates in the naming system compared the number of updates in the RLR, see Figure 61, it is seen that updates that change information in the RLR is not that many.

So, how could the RLR be distributed based on a DNS solution? A modified solution of the DNS with maybe a lesser number of hierarchical levels than what is the case today, and a cache TTL that is set so that the system has as good performance as possible when considering the traffic related to updates. This setup could be seen in Figure 74, which shows the clients' nearest router connected to the RLR (i.e. the lowest level in the DNS hierarchy). In the figure the RSs of different levels is omitted.

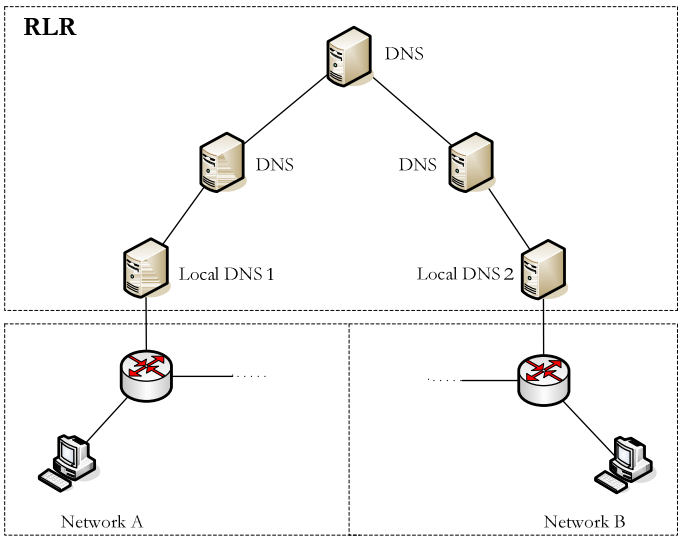

Figure 74: Possible layout of distributed RLR using the DNS method

The advantages gained from distributing the RLR in the following manner is that the load, due to lookups, on the RLR is distributed, thus increasing performance. To

distribute it the following way also gives a better performance of the system regarding the memory needs and the CPU demands.

The downside of this solution is that when this report is written, it is hard to determine the effects of caching in the RLR/DNS-solution. The value of the cache TTL should be set so that it is in the same range as the update frequency towards the RLR. Another drawback is that the increased lookup time due to the extra number of hops in the RLR itself that could be introduced. The extra number of hops could be minimized when utilizing caching with properly chosen TTLs.

Regarding the caching in the DNS naming system the previous thesis looked at the correlation between cache TTLs, how often movements in the network occurred, and the effect on the hit ratio. They found that to achieve a 95 % hit ratio in the DNS naming system with a specific TTL on different levels in the hierarchy the clients could not move too often. In the table below, Table 16, these findings is shown [23]. The table shows the TTL values used at the different levels in the DNS hierarchy. The values is on the form x/y/z where x represent the TTL on the lowest level in the hierarchy, the y value corresponds to the TTL value on the TLD level, and the z value corresponds to the cache TTL on the highest level i.e. at the root-servers.

Table 16: Time between movements to reach 95 % hit ratio with connections every second

TTL (seconds)	Time between movements until reached 95 % (seconds)
0/0/0	7-8
1/2/3	11
0/600/900	7
1/600/900	13
10/20/30	94
100/200/300	980
3600/5040/7200	34000

The following figure, Figure 75 which is taken from [23], also shows how the hit ratio in a DNS system is affected by the TTL and the frequency of movements. It is seen in the figure that a 95 % hit ratio is achieved only when having a cache TTL of 300/420/600 and that movements, i.e. updates that affect the RLR, occur not more often than every 40-50 minute (2500-3000 second). This means that updates that reaches all the way up to the RLR can not occur more often than this if one wants a system with a reliable performance against the users. The figure below is an extreme case when name resolution requests pertaining to connection setups against the DNS system, i.e. in this case when the RLR is represented by a DNS system, occur every second.

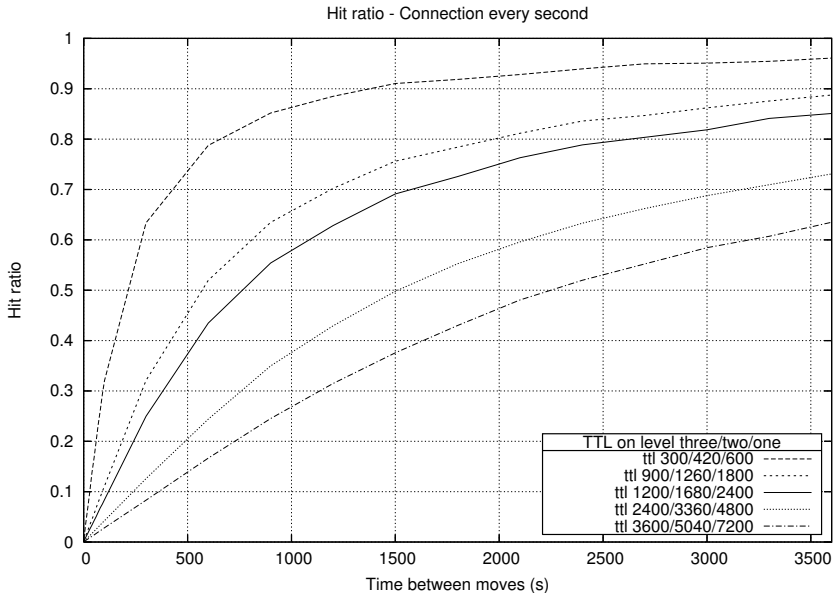

Figure 75: Hit ratio in a DNS system with moving clients, different TTLs, and connections every second

When looking at the other extreme, when a connection occur more sparsely, as depicted in Figure 76 below, it is seen that the same TTL value ("ttl 300/420/600") gives a 95 % hit ratio when updates ("Time between moves (s)" on the x-axis) occur more frequently. With this setup updates could occur every 500 second (roughly every 8 minute), and a high hit ratio is still achieved.

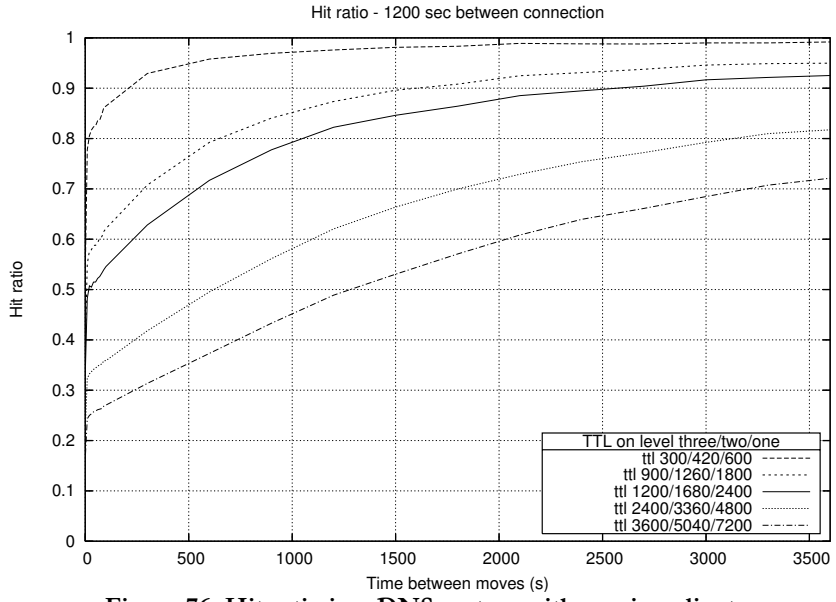

Figure 76: Hit ratio in a DNS system with moving clients, different TTLs and connections every 1200 second

The problem with the figures above (Figure 75 and Figure 76) is that they seem to converge as the time between movements increases. It would be interesting to know how the curves looked if longer tests had been performed, i.e. if the time between movements where higher. This way it would be possible to see at what value the curves converged.

Another problem with these figures is that in the real world a 95 % hit ratio is not acceptable. Many times one or two failed connections, due to resolution errors, during one day are acceptable for a user. The only case seen in the figures above that could deliver this is to use a ttl setting of 300/420/600, connections every 1200 second (20 minute), and movements/updates that affect the RLR not more often that every 3600 second (1 hour). This is seen in Figure 76 with the curve "ttl 300/420/600", that shows the hit ratio (y-axis) converging to 1 as the time between movements increases (x-axis).

The problem with the DNS naming system is that it cannot maintain a high hit ratio when the system has lots of updates. Even though when the number of updates is rather small the cache timers will introduce problems. To determine the values of the cache timeouts the frequency of updates that occur in the RLR has to be evaluated. This could be done by using the mobility model used during this work together with a simulation topology that is more detailed and more close to the real world than the one used here. This way the amount of full hierarchal movements could be examined and from that a conclusion of cache timeout values could be drawn.

5.8.2 RLR Distributed Using a DHT-based Method

The other solution is to have the RLR distributed with the aid from a DHT-based solution, which is the Chord ring for this work. So, the information that is stored in the RLR should be evenly distributed over the Chord ring. This is depicted in Figure 77, where the rest of the Path-Coupled structure is omitted.

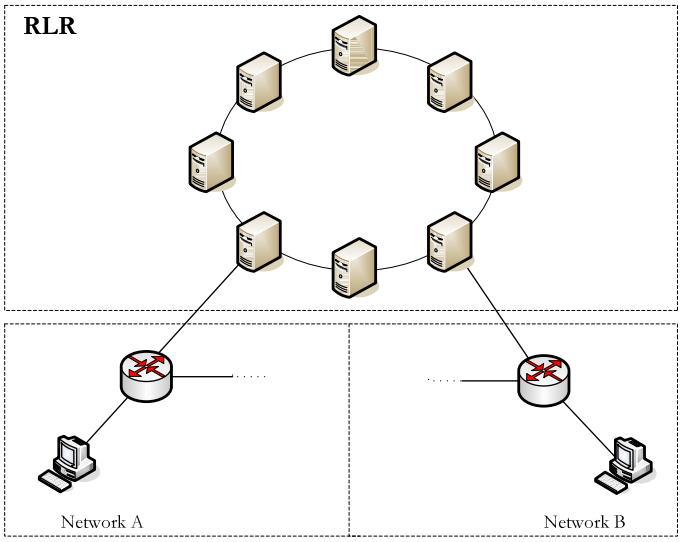

Figure 77: RLR distributed using a Chord ring

The case here, which is the same as in previous sub section, is that the RLR must be able to handle the number of lookups and updates as seen in Figure 59 and Figure 61. The number of lookups that reaches the RLR is numerous, whereas the number of updates that affects the information kept in the RLR is rater infrequent.

The advantage of using Chord, or another DHT-based solution, as a solution for a distributed RLR is that it scales well when the number of lookups to the system increases, which can be seen in Figure 47, Figure 53, and in Figure 54.

Issues that can be found when using a Chord solution is that depending on the ring size the lookup times will be affected. When the size of the ring increases the lookup time will also increase. But, on the other hand, the load on each node in the ring will decrease when the number of nodes in the ring is increased.

When looking on the hit ratio and the frequency of movements, the work done in the previous thesis shows that to reach a 95 % hit ratio the time between movements, i.e. updates in the system, should not be more often than every 16 seconds to 20 seconds. These findings are summarized in Table 17 below, and should be compared to the ones in Table 16 above. [23]

Table 17: Time between moves to achieve a 95 % hit ratio in a Chord system

Number of Chord nodes	Time between moves (seconds)
512	16
1024	17
2048	18
4096	20

The data in the table above is also represented by a diagram, shown in Figure 78 below. It shows that a 95 % hit ratio is reached rather fast, and that a naming system based on Chord is able to handle rather frequent movements. In the simulations performed by this work a ring size of 32 and 64 nodes has been used. How the effect from movements on the hit ratio looks like has not been measured. But from the trend seen in the table, one can assume that a smaller ring opens up for more frequent updates. But the design of the Path-Coupled method is such that the number of updates that reaches the RLR is kept at a minimum, since full hierarchal movements are rare.

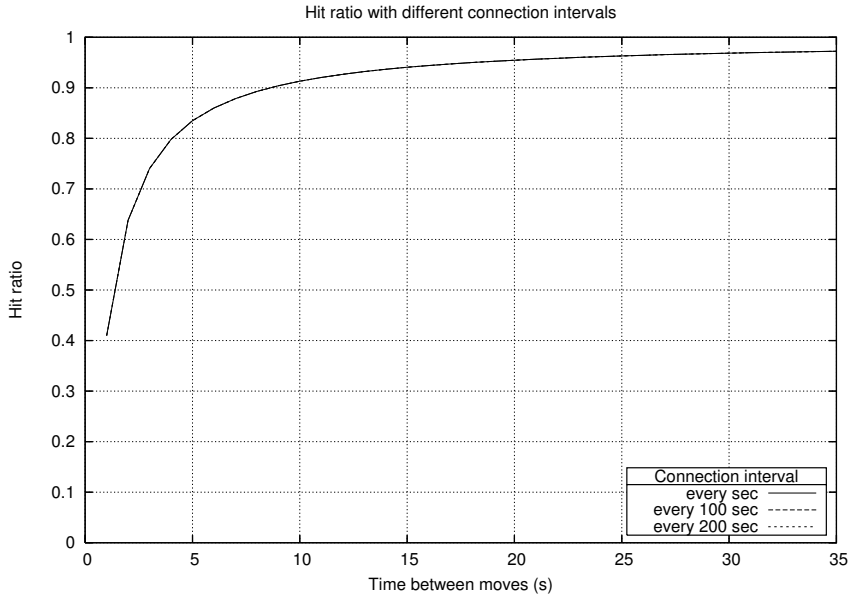

Figure 78: Hit ratio in Chord based naming system over different movement rates and using different connection intervals

In the figure above the curves ("every sec", "every 100 sec", and "every 200 sec") seem to converge rather quickly. This should be compared to Figure 75 and Figure 76 in the previous section, where it is seen that the time between moves that is

needed to make the curves converge is much bigger, i.e. it is two order of magnitude in the DNS case.

In the figure below, Figure 79, the hit ratio is a function of time between movements. The measurements is performed on different Chord ring sizes, and shows that the hit ratio is not affected by the ring size, the small difference that exists is to small to have any significance. But, what is interesting to observe is that the curves seem to converge to a value between 0.95 and 1. This means that the Chord naming system never gives a 100 % hit ratio, meaning that a lot of requests to the ring will never be answered or given an incorrect answer. It would be interesting to see what would happen with the curves as the time between movements increase.

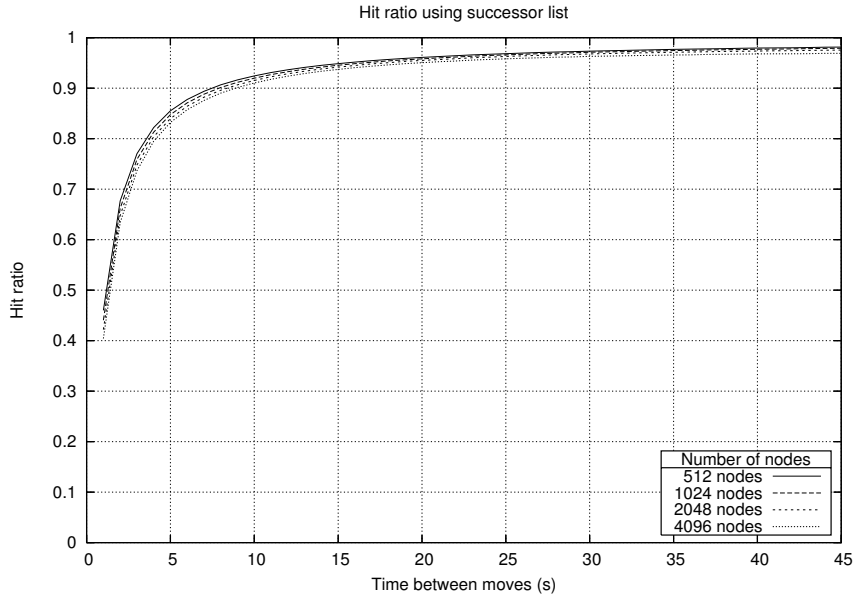

Figure 79: Hit ratio in Chord ring using a successor list and varying the ring sizes

The effects imposed by a Chord ring, i.e. the fact that a 100 % hit ratio perhaps never is achieved is disregarded for the moment, are that the lookup time is increased. The factors that affect the lookup time are the latency between the nodes and the size of the ring. But, this increased lookup time can be minimized with for example large finger tables, a clever caching scheme, or a DHT-based solution that works better than the Chord system. The information found in [41] shows that, using another DHT protocol than Chord, it is possible to achieve a lookup time that is O(1). The idea is to replicate the most popular objects at the moment to reduce the latency. But, more replication gives higher storage demands and bandwidth costs. The authors of the CoDoNS application [41] have been able to utilize the Beehive [42] DHT protocol and a replication algorithm to optimize the balance between latency and replication. They also address the issue of fast updates in the system, attacking the problem of slow propagation of updates, due to cache TTLs, in the DNS.

5.8.3 Summary Regarding the RLR Distribution

To summarize what has been written above, it has been shown that a DNS solution for the RLR distribution could work fine, if the cache TTLs is set properly. The DNS solution is also able to achieve a high hit ratio if the updates is not to frequent, and the lookup latency is acceptable (depending on the efficiency of caching). A

DHT-based solution provides a high hit ratio when the number of updates is roughly 15 to 20 seconds apart, and it has a good lookup latency O(log n).

In Figure 80 the hit ratio from a Chord ring of 4096 nodes is compared to a DNS naming system. The Chord solution clearly outperforms the DNS solution. The only DNS alternative that performs slightly better than Chord is when having a cache TTL of 0 seconds, which is not desirable.

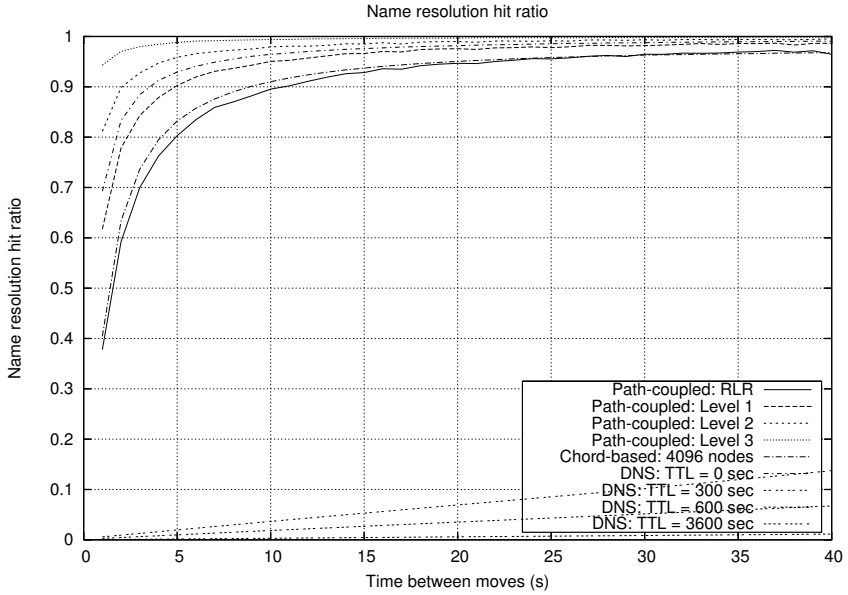

Figure 80: Comparison of hit ratio between the different systems

But, the problem with Chord-based solution is that the solution does not achieve a 100 % hit ratio, in contrast to the DNS system which could achieve 100 % hit ratio if the TTL issue is solved. A Path-Coupled system must provide a high accuracy to a user when performing name-to-address resolutions. Not having the last 5 %, as seen in Figure 79, in the hit ratio could make the method to go from killer app to total failure.

The conclusion is that a DHT-based solution is not a good alternative for the distribution of the RLR. Maybe there exist other DHT-based protocols that could provide a better hit ratio, but that issue has not been investigated.
A DNS-based solution is a more attractive solution for the distribution of the RLR. Even though the cache timeouts causes problems as showed in the previous sections, the systems lack of support for more frequent movements in the system, it has some advantages. One of the advantages is the systems capability to achieve a high hit ratio when set up properly. Another advantage is that the Internet as of today already employees the DNS solution, thus giving the opportunity for easier transition and deployment of the new Path-Coupled method.

But, keep in mind that this section is only an initial survey of the distribution aspects regarding the RLR. This aspect needs further investigation before a decision can be made on how to distribute the RLR.

6 Conclusions and Future Work

In this master thesis, which is a continuation of a previous work concerning the validation of the Path-Coupled method, three different methods for name-to-address resolution have been evaluated regarding scalability. The mentioned new novel dynamic name-to-address resolution method Path-Coupled have been compared against the method that controls the Internet today, namely the DNS method [40], and a name-to-address method based on the DHT implementation Chord [46].

In the previous work the evaluations where performed using the ns-2 simulator; since that simulator was not capable to run the simulations for this work the OMNeT++ simulator was chosen.

The key measurement in this evaluation is the scalability aspect of the three methods in a highly mobile environment. To achieve this each client in the simulation playground moves according to a mathematic function, also called mobility model, and generate three different types of traffic patterns. The movements in the network are to generate updates to the naming system, to see how it behaves during mobile scenarios. The traffic from each client mimics the behavior of web traffic, streaming traffic, and telephone traffic. They do not actually send or receive any of this traffic; the traffic pattern is there to trigger lookups to be sent to the naming system in a realistic fashion.

The second aspect of this work was to investigate the distribution aspect of the RLR in the Path-Coupled name-to-address method needed to cope with the scalability issues of logically centralized RLR. A background of the problems concerning the RLR has been given. There is also a discussion about different solutions that is based on the results gained from the simulations performed and the results obtained in the work done before this master thesis.

The result from this evaluation shows that DNS without cache suffer from poor scalability. Apart from that the method show good properties regarding the scaling aspect, but the DNS method with cache shows poor properties regarding the mobility aspect. The Chord-based method scales well and is able to handle high mobility, but it is seen that it suffers from memory problems and has problems with not achieving a good hit ratio. In the Path-coupled method the RLR is showed to have problems regarding scalability, which is investigated in the sections on distribution aspects of the RLR. The Path-Coupled is designed to handle high mobility in the network, and it is seen that, apart from the RLR, it has good properties to handle this issue.

The investigation of the RLR distribution shows that the number of updates that affects the RLR is rather infrequent. It is also shown that to achieve an acceptable hit ratio in the RLR using a DNS distribution, the timeouts of the cache is a critical issue. It is seen that a DHT-based distribution of the RLR is a good alternative due to its scalability features, and that it performs well. But the Chord solution is not able to ensure a 100 % hit ratio, which is necessary for the performance of a naming system. The DNS solution shows that it is more reliable in the hit ratio aspect. But a DNS-based solution suffers from the problem of cache TTL values, which is not in the scoop of this report to investigate further.

Future Work

This study has focused on the scalability aspect, during high mobility, of the three different name-to-address methods. But there still exists some work to be done before a live version of the protocol could be deployed.

One aspect that needs further investigation is the distribution aspect of the RLR. A more thorough evaluation of different distribution methods is needed. Maybe alternatives to the Chord-based solution should be investigated regarding the RLR distribution. Other DHT-based protocols could have better performance regarding lookup latency and hit ratio than the Chord solution. The frequency of updates that reaches and affects the RLR also needs further investigation, and how the TTL values in a DNS-based distribution should be set.

The new Path-Coupled method is according to the specification more secure that the existing solutions. This feature should be evaluated on several aspects, namely: resistance against DOS-attacks and the ability to prevent hi jacking and man in the middle attacks, to see if it is possible to incorporate a scheme that ensures authentication of data and data integrity. Other interesting properties that are important to study are the robustness and resilience of the system.

7 References

[1] F. Bai and A. Helmy, 2004. *A Survey of Mobility Modeling and Analysis in Wireless Adhoc Networks*. Book Chapter in submission to Kluwer Academic Publishers.

[2] T. Camp, J. Boleng, and V. Davies, 2002. *A Survey of Mobility Models for Ad Hoc Network Research*. Wireless Communication & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications, vol. 2, no. 5, pp. 483-502, 2002.

[3] C. Bettstetter, H. Hartenstein, and X. Perez-Costa, 2004. *Stochastic Properties of the Random Waypoint Mobility Model*. ACM/Kluwer Wireless Networks, Special Issue on Modeling and Analysis of Mobile Networks, vol. 10, no. 5, Sept 2004.

[4] V. Borrel, M. Dias de Amorim, and S. Fdida, 2005. *On natural mobility models*. LNCS, International Workshop on Autonomic Communication (WAC), Athens, Greece - October, 2005.

[5] *IMPORTANT: An evaluation framework to study the "Impact of Mobility Patterns On RouTing in Ad-hoc NeTworks"*. F. Bai. Internet; accessed 5 October 2006, <http://nile.usc.edu/important/>.

[6] F. Bai and A. Helmy, 2004. *The IMPORTANT Framework for Analyzing and Modeling the Impact of Mobility in Wireless Adhoc Networks*, Book Chapter in submission to Kluwer Academic Publishers.

[7] J. Yoon, M. Liu, and B. Noble, 2003. *Sound Mobility Models*. Proc. ACM MobiCom, September 2003, San Diego, CA.

[8] J. Yoon, M. Liu, and B. Noble, 2003. *Random Waypoint Considered Harmful*. Proc. IEEE INFOCOM, vol 2, pp 1312-1321, April 2003, San Francisco, CA.

[9] C. Chiang, 1998. *Wireless networking multicast*. PhD Thesis, University of California, Los Angeles.

[10] *Wikipedia: Self-similarity*. Internet; accessed 20 September 2006, <http://en.wikipedia.org/wiki/Self-similar>

[11] M. Crovella and A. Bestavros, 1997. *Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes*. IEEE/ACM Transactions on Networking, 5(6):835--846. Revised and substantially corrected version of [Crovella and Bestavros, 1996].

[12] W. E. Leland, M. Taqqu, W. Willinger, and D. V. Wilson, 1993. *On the Self-Similar Nature of Ethernet Traffic*. Proc. SIGCOM93, 1993, San Francisco, California, pp. 183-193.

[13] V. Paxson and S. Floyd, 1995. *Wide-Area Traffic: The Failure of Poisson Modeling*, IEEE/ACM Transactions on Networking, pp.226-244, June 1995.

[14] K. Calvert, M. Doar, E. W. Zegura, 1997. *Modeling Internet Topology*. IEEE Communications Magazine 35:160--163.

[15] J. Winick and S. Jamin, 2002 *Inet-3.0: Internet Topology Generator*. University of Michigan Technical Report CSE-TR-456-02.

[16] A. Medina, A. Lakhina, I. Matta, and J. Byers, 2001. *BRITE: An Approach to Universal Topology Generation*. Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems-MASCOTS'01, Cincinnati, Ohio, August 2001.

[17] PSGen. Internet; accessed 11 October 2006, <http://www.mindfab.net>

[18] R. Parkinson, *Traffic Engineering Techniques in Telecommunications*, Infotel Systems Corp. Internet; accessed 10 October 2006, <http://www.tarrani.net/mike/docs/TrafficEngineering.pdf>

[19] K. Lan and J. Heidemann, 2001. *Structural Modeling of RealAudio Traffic*. ISI-TR-544.

[20] P. Barford and M. Crovella, 1998. *Generating Representative Web Workloads for Network and Server Performance Evaluation*, Proceedings of the ACM SIGMETRICS Conference, Madison, WI, July 1998.

[21] B.P. Gerő, T. Varga, and S. Győri, 2006. *Reference Traffic Model for User Traffic in WCDMA and GSM*. 2/155 59-HSD 101 13/6 Uen. Ericsson Internal

[22] M. Särelä, 2004. *Measuring the Effects of Mobility on reactive Ad Hoc Routing Protocols*. Master of Science Thesis, Helsinki. HUT-TCS-A91.

[23] M. Johansson, K. Olsson, and P Åkerlund, 2006. *Dynamic Name-to-Address Resolution. Evaluation of a novel name-to-address resolution method.* MSc Thesis, ICT/ECS-2006-89, School of Information and Communication Technology, Royal Inst of Technology (KTH), Stockholm, 2006.

[24] B. Hechenleitner and K. Entacher, 2003. *Pitfalls when using parallel streams in OMNeT++ simulations*. International Workshop on Inter-domain Performance and Simulation (IPS) Salzburg, Austria, 20-21 February, 2003.

[25] *Wikipedia: Computer Simulation*. Internet; accessed 11 February 2007, <http://en.wikipedia.org/wiki/Computer_Simulation>.

[26] Y. A. Sekercioglu, A. Varga, and G. K. Egan, 2003. *Parallel Simulation made Easy with OMNeT++*. 15TH EUROPEAN SIMULATION SYMPOSIUM AND EXHIBITION, Simulation in Industry, Simulation Services for the Future, October 26-29 2003, Delft, The Netherlands.

[27] *OMNeT++, Discrete Event Simulation System*. Internet; accessed 11 February 2007, <http://www.omnetpp.org>

[28] *The PRIME Research, Parallel Real-time Immersive network Modeling Environment*. Internet; accessed 11 February 2007, <http://prime.mines.edu/index.html>

[29] *ModelNet, Systems and Networking, UCSD Computer Science*. Internet; accessed 11 February 2007, <http://modelnet.ucsd.edu/>

[30] *JiST / SWANS; Java in Simulation Time / Scalable Wireless Ad hoc Network Simulator*. Internet; accessed 11 February 2007, <http://jist.ece.cornell.edu/>

[31] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker, 2002. *Scalability and accuracy in a large-scale network emulator*. ACM SIGOPS Operating Systems Review archive, Volume 36, Issue SI (Winter 2002), OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation, SPECIAL ISSUE: Peer-to-peer infrastructure, Pages: 271 – 284, 2002.

[32] J. Perez, 2005. *MQTT Performance Analysis with OMNeT++*. MSc Thesis, School of engineering and Research Center in Communication Systems, École Nationale Supérieure des Télécommunications, Sophia Antipolis, 2005.

[33] *OMNeT++ User Manual*. Internet; accessed 11 February 2007, <http://www.omnetpp.org/doc/manual/usman.html#toc_12>

[34] A. Varga. *Re: [omnetpp] experiences with large networks? – revisited* (4 October 2005). omnetpp-l at omnetpp.org, available at <http://www.omnetpp.org/listarchive/msg05768.php>. Internet; accessed 26 January 2007.

[35] Roland Bless. *Re: [omnetpp] experiences with large networks? – revisited* (6 October 2005). omnetpp-l at omnetpp.org, available at <http://www.omnetpp.org/listarchive/msg05772.php>. Internet; accessed 11 February 2007.

[36] M. Liljenstam, J. Liu, D. Nicol, Y. Yuan, G. Yan, and C. Grier, 2005. RINSE: the real-time immersive network simulation environment for network security exercises. Simulation: Transactions of the Society for Modeling and Simulation International, 82(1):43-59, January 2006.

[37] E. Schoch, 2006. *VANET Simulations with JiST/SWANS*. Presentation made at Secure Vehicle Communication Workshop SEVECOM 2006.
[38] B.P. Gerő, T. Varga, and S. Győri, 2006. *Reference Traffic Model for Signalling Traffic in GSM R12, WCDMA R5 Systems*. 1/15559-HSD 10113/5, Ericsson Internal.
[39] V. Pappas, D. Massey, A. Terzis, and L. Zhang, 2006. *A Comparative Study of Current DNS with DHT-Based Alternatives*. In the Proceedings of IEEE INFOCOM'06, Apr. 2006.
[40] *Wikipedia: Domain Name System*. Internet; accessed 20 January 2007, <http://en.wikipedia.org/wiki/Domain_name_system>.
[41] *CoDoNS, Cooperative Domain Name System*. Internet; accessed 29 January 2007, <http://www.cs.cornell.edu/people/egs/beehive/codons.php>.
[42] *Beehive*. Internet; accessed 29 January 2007, <http://www.cs.cornell.edu/people/egs/beehive/index.php>.
[43] *ITU: ENUM*. Internet; accessed 29 January 2007, <http://www.itu.int/osg/spu/enum/>.
[44] *Ambient Networks*. Internet; accessed 2 February 2007, <http://www.ambient-networks.org>.
[45] *Telecom Traffic Online*. Internet, accessed 19 January 2007, <http://www.erlang.com/bandwidth.html>.
[46] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F Kaashoek, F. Dabek, and H. Balakrishnan, 2001. *Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications*. IEEE/ACM Transactions on Networking.

Appendix

A.1 List of Abbreviations

3G – Third Generation Technology
AS – Autonomous System
CPU – Central Processing Unit
CS – Circuit Switched
DHT – Distributed Hash Table
DNS – Domain Name System
GM – Gauss Markov Mobility Model
HSDPA – High-Speed Downlink Packet Access
HTTP – Hyper Text Transfer Protocol
IAT – Inter Arrival Time
IP – Internet Protocol
ISP – Internet Service Provider
ITU (ITU-T) – International Telecommunication Unit (ITU Telecom Standardization)
LAN – Local Area Network
MHT – Mean Holding Time
NED – NEtwork Description
PC – Personal Computer
PS – Packet Switched
RDM – Random Direction Mobility Model
RFC – Request For Comments
RLR – Resource Location Register
RPGM – Reference Point Group Model
RS – Resolution Server
RTC – Registration Transaction Code
RTP – Real-time Transport Protocol
RTSP – Real-time Streaming Protocol
RWK – Random Walk Mobility Model
RWP – Random Waypoint Mobility Model
SR – Smooth Random Mobility Model
TCP – Transmission Control Protocol
TTL – Time To Live
UDP – User Datagram Protocol
URL – Uniform Resource Locator
WCDMA – Wideband Code Division Multiple Access
WWW – World Wide Web

A.2 Modifications of the Simulation API

In the OMNeT++ framework some modifications where done to the predefined NED-files. These changes are as follows:

Router.ned
Changed lines 60 and 62 from

```
ppp:    PPPInterface[sizeof(out)];
  display: "p=90,257,row,110;q=l2queue;i=block/ifcard";
eth:    EthernetInterface[sizeof(ethOut)];
  display: "p=145,257,row,110;q=l2queue;i=block/ifcard";
```

to

```
ppp:    PPPInterfaceNoQueue[sizeof(out)];
  display: "p=90,257,row,110;q=l2queue;i=block/ifcard";
eth:    EthernetInterfaceNoQueue[sizeof(ethOut)];
  display: "p=145,257,row,110;q=l2queue;i=block/ifcard";
```

Standardhost.ned
Changed line 86 from

```
ppp:    PPPInterface[sizeof(out)];
  display: "p=205,350,row,90;q=txQueue;i=block/ifcard";
```

to

```
ppp:    PPPInterfaceNoQueue[sizeof(out)];
  display: "p=205,350,row,90;q=txQueue;i=block/ifcard";
```

A.3 Strange phenomena

When trying to set the processing delay of the IP module it has to be done twice, or else it will not apply to all routers.