Modeling and Simulation Platform for Large-Scale Network Structures

Group 880

3rd June 2003

Aalborg University

Institute of Control Engineering

Frederik Bajers Vej 7 DK-9220 Aalborg Ø Telephone +45 96 35 87 00

TITLE: Modeling and Simulation Platform for Large-Scale Network Structures

THEME: Network Management

PERIOD: 3rd of February - 3rd of June, 2003

PROJECT GROUP:

Group 03gr880

GROUP MEMBERS:

Frode Fjermestad
Peder Helle
Christian Lodberg
Michael Pedersen
Muhammad Tahir Riaz

SUPERVISOR:

Ole Brun Madsen

PUBLICATIONS: 8

MAIN REPORT: 102 pages

APPENDIX: 24 pages

ENDED: 3rd of June 2003

Synopsis:

This report documents the development of a simulation platform for large-scale network structures. The platform is developed by using the object oriented language C++ with a command prompt user interface.

The platform can generate networks of different topologies. These are: simple ring, double ring, wheel, fully connected, N2Rpq and random topology. It is also possible to create a network manually. After generation the networks can be manipulated. The platform is able to simulate the maximum flow, number of hops and traffic load for networks up to 1500 nodes. The maximum simulation time for each of the algorithms is below 15 minutes.

A Java application presents the simulation results graphically in order to make it easy to compare network performance.

Preface

This report is written by five students from the eigth semester of the specialization Network Planning and Management at Aalborg University. The project has been documented during the period from 3rd of February to 3rd of June 2003.

Frode Fjermestad

Christian Lodberg

Peder Helle

Michael Pedersen

Muhammad Tahir Riaz

About this Report

Disposition of the Report

The report is divided into four major parts, which are:

1. Analysis and Requirements.
2. Development of the Simulation Platform.
3. Simulation and Analysis of Graphs.
4. Large-Scale use of the Simulation Platform.

The contents of each part will be summarized briefly in the following paragraphs. Also, the report contains a number of appendixes, which are used to document details, which are not relevant for the main report, e.g. use cases used in the object oriented analysis and design.

Analysis and Requirements

The Analysis and Requirements part begins with a short description of different network performance parameters. Furthermore, a perfect simulation platform is described. Then the requirements specification is situated in order to introduce the specific requirements and limitations to the simulation platform.

Development of the Simulation Platform

The development of the simulation platform begins with two chapters, that provides an object oriented analysis and object oriented design respectively. Hereafter, the simulation platform is implemented. Furthermore, this part ends with a performance test versus acceptance test.

Simulation and Analysis of Graphs

This part first describes different large-scale network structures. Afterwards, the network structures will be simulated by using the simulation platform.

Large-Scale use of the Simulation Platform

The final part describes a simulation of different network configurations containing 1500 nodes using the platform. The part ends with a chapter in which the simulator platform's limitations are described. Furthermore, the uses of Linux clustering for large-scale network simulations are discussed.

Typographic Aids

The report is written in LaTeX and the figures are made in XFIG.

Bibliographic references are given in square brackets with the name of the author(s) and the year of publication. An example of a bibliographic reference: [1]. If no explicit author can be identified, the company or organisation name is being used instead. Some references may include a specific page number in the reference, but this principle is not adhered to in general.

References within the report come in two types. If the reference is within the current chapter, only the section number is being used for reference (e.g. section 5.2). Otherwise, if the reference is to another chapter of the report, the reference is given with both section number and page number (e.g. section 5.2 on page 25).

Notes in the report are given in footnotes[1]. Figures and tables are identified by the chapter number and an incrementing number within each chapter. The same numbering scheme is used for equations and formulas, but only important and/or referenced formulas are numbered. An example:

$$E = m \cdot c^2 \tag{1.1}$$

Contents of the Enclosed CD-ROM

The CD-ROM enclosed with this report contains all the software and source-code used in the project.

When the CD is put into a computer with a Windows operating system, a user interface will appear. From this the contents are found.

[1] This is an example of a footnote

Contents

Chapter 1

Introduction

According to the national statistical information about percentage of people accessing the Internet in Denmark[1], 72% of the Danish population had access to the Internet 2 years ago. Today more than 77% of the Danish population has access to the Internet either from work or at home, and the number is increasing. A plot of the trend in Internet access is sketched in figure 1.1. The amount of data sent through the Internet is also increasing rapidly.

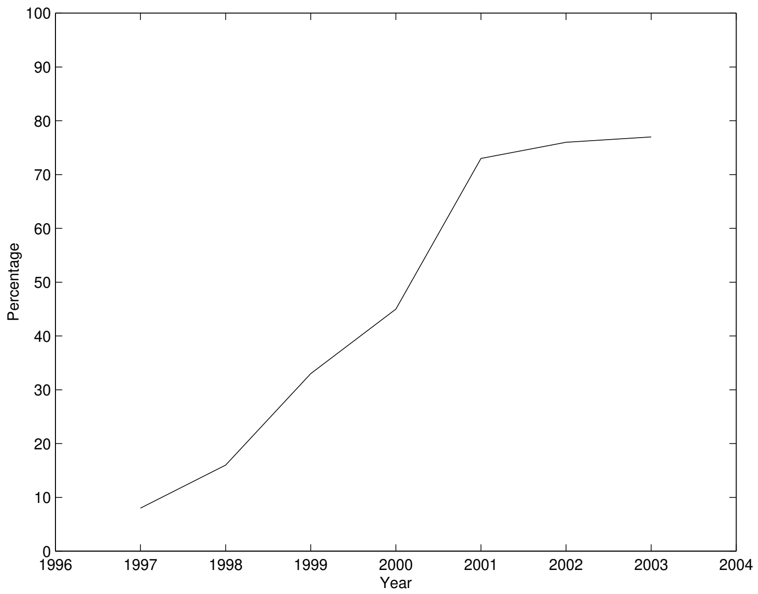


Figure 1.1: Amount of the Danish population with access to the Internet.

If this continues in a few years, the traffic at the Danish network will exceed its capacity. Therefore, it may be appropriate to expand the network or make a completely new up-to-date and future proof network with fiber optics in the ground.

When designing such a network, a strategy of how many switching stations and how they must be interconnected is needed. This is what this report deals

[1] www.dst.dk

with.

The purpose of this project is to develop a simulation platform for large-scale network structures. The platform must simulate some of the most interesting parameters, which will be explained in the report.

In order to make a useful simulation platform, different functions or tools must be provided by the system user. Therefore, first an analysis to such a platform will appear in this project. An analysis of the various performance parameters which can be applied in order to enhance or solve different problems in simulating large-scale network structures will also be given. Furthermore, a platform which must fulfill all requirements given by the user will be described and therefore in this context called a perfect simulation platform.

In the object oriented analysis and design, the actual simulation platform will be developed. A use case analysis will first be performed, in order to find out exactly how the product will be used. Relations between the classes are described before the actual design starts. In the design phase, the static model will describe the responsibilities, attributes, and relationships between each class. Then implementations will be carried out, a status bar and a menu system will give the user of the simulator the actual interface where the interaction and information will be provided. Finally, the product will be tested before it will be used for simulating different network configurations in order to give the results. Different simulations will be carried out and described. Furthermore, a continuous use of the words; platform and tool will be used throughout the report.

The project uses the terms used in graph theory. There is a short description of the graph theory definitions in appendix A on page 105.

Part I

Analysis and Requirements

Chapter 2

Network Performance Parameters

This chapter contains a consideration of different network performance parameters. There are several network performance parameters and in network design it is wanted to fulfill most of the demands from these parameters. But an optimization concerning one parameter is often degrading one or more other parameters. The idea behind this chapter is to find the network performance parameters in which the simulation platform should be able to simulate. This chapter is mostly based on [1].

2.1 Reliability

A high value on this parameter secures that every bit is delivered correctly. A network can be reliable in form of error control and error correction. But in network design, the reliability parameter is defined as a percentage of uptime compared to down time. The reliability parameter is important in the ability to trust the network. Low reliability on a network will cause a lot of harm for a user. This parameter will not be taken into consideration when designing the platform.

2.2 Jitter

For applications such as audio and video streaming, it does not matter if the packets uses 20 ms or 30 ms to be delivered, as long as the transmission time is constant [1]. The variation in the packet arrival time is called jitter. High jitter will give an uneven quality to an audio or a video sequence, therefore the requirements to jitter is high with audio and video on demand. This parameter is not taken into consideration in the design of the simulation platform, since it mostly will depend on the application a network is going to be used for.

2.3 Bandwidth

Bandwidth has a general meaning of how much information can be carried in a given time period (usually a second) over a communication link. In a more technically point of view bandwidth is the range of frequencies which an electronic signal occupies on a given transmission medium. Both analog and digital signals have a bandwidth. The bandwidth is defined as the frequency difference of the lowest and highest signal component in analog systems. In digital systems bandwidth is expressed as data speed in term of bits per second (bps).

A large-scale network will typically consist of a lot of links, each with its own bandwidth. If the bandwidth of one of these links is much smaller than the others it is said to be a network bottleneck in which is wanted to be avoided. In the simulation tool, nodes and links will be given a capacity which can be considered as bandwidth.

2.4 Latency/Delay

In a computer network, latency is similar to delay. It is an expression of how much time one packet of data use to travel from one point to another. Some of the most used latency measures for networks are end-to-end trip time, round-trip time and keystroke response time.

End-to-end trip time is the time it takes for a data unit to travel from source to destination. Round-trip time can be found by sending a packet that is returned to sender measuring the time used. Keystroke response time is the delay from when a button is pressed until something happens on the screen. This can be experienced in web-browsing. Even if one is connected with a broadband connection, there sometimes will go some time from a link is clicked, until there are changes on the screen.

Propagation time, and the operation time of the network components as routers and switches will influence the latency. Also this parameter is partly related to the number of hops parameter, but will not be included in the simulation platform design.

2.5 Cost

When analysing different topologies each element in the network costs money. The elements to consider are the nodes and links which makes the connection between the nodes in the network. The cost parameter is directly related to the quality of a network, for more money more links can be added and better nodes can be used.

Minimizing the cost when designing a network is an important goal. The links between the nodes have to be placed in a clever structure to achieve good performance for an acceptable cost. In another point of view, it can be cheaper to invest more money when building a network. If a network is build to meet predicted future requirements, such as e.g. a larger number of fiber cables, is the costs of course increasing. But the price of some extra fibers or fiber tubes is relative small according to digging the ditch. Therefore money will be saved in the future when improvements has to be added as the requirements to the network performance is increasing.

The price to pay when using different networks is not going to be considered in the simulation platform design. But the number of links according to the number of nodes and link and node capacities can give an overview of the cost parameter.

2.6 Scalability

The use of this parameter displays the network as being able to be expanded or connected to other networks. The Internet, e.g., is a good example of a large-scale network, where nodes are added all the time.

This parameter in the network topologies are an important step toward an accurate mathematical model for both analysing the performance of existing topologies and designing new and more scalable networks. This is because it is often a problem that there is too little knowledge about the existing network topologies when developing new network topologies for large-scaled networks. The simulation platform will be designed in such a way that a network is able to be expanded, but the platform will not have any knowledge concerning the scalability parameter.

2.7 Redundancy

Designing redundant routes has two purposes: minimizing down time and load balancing. Minimized down time is described first.

Minimized down time leads to increased uptime. By using redundant or meshed network designs, the effect of link failures is minimized. A fully meshed network provides complete redundancy; every node has a link to every other node. The number of links in a fully meshed network can be calculated by the equation, which is given below.

$$\text{Number of links} = n \cdot \frac{n-1}{2} \tag{2.1}$$

In equation 2.1, n is the number of nodes. Each node is connected to every other node. Divide the result by 2 to avoid counting node X to node Y and node Y to node X as two different links. This provides good performance, because there is only one single hop delay between any two sites. A fully meshed network is on the other hand very expensive and will need more bandwidth and CPU resources to process broadcasts because of the high number of routes.

Load balancing is a concept that allows a node to take advantage of multiple best paths to a given destination. The paths are derived either statically or with dynamic protocols, such as RIP, EIGRP, OSPF, and IGRP (see appendix F for these acronyms). Load balancing can occur with certain protocols, when there are several equal cost routes to a single destination. This may also occur with unequal cost paths with EIGRP and IGR [2]. Pre destination load balancing means that the node distributes the packets based on the destination address. Given two paths to the same network, all packets to destination 1 in that network will go through the first path, all packets for destination 2 in that network will go through the second path, and so on. Pre packet load balancing means that the node sends one packet to destination 1 through the first path, the second packet to (the same) destination 1 through the second path, and so on. Load balancing improves network performance.

The redundancy parameter will at this time not be considered in the simulation platform design.

2.8 Network Flow

A stream of packets from a source to a destination is called a network flow. In a connection oriented network, all the packets belonging to a flow follow the same route, but in a connection less network, they may follow different routes. If the flow is good compared to bandwidth, the utilization of the network is good. A maximum flow simulation only concerning the link capacities will be designed in the simulation platform. To improve the maximum flow simulation in further work with the platform, the node capacities can also be added into th algorithm to find the real maximum flow.

2.9 Robustness in Networks

Robustness in networks means that a network has an ability to recover in certain ranges of exceptions and situations. A robust network can take account of the hazards of failures and their automatic recovery. Another definition states that a robust network is a network that behaves in a controlled and expected manner when expected variations arise in its dominant parameters, but also in the face of unexpected variations. A simulation which calculates a degree of robustness will not be available in the simulation platform.

2.10 Throughput

Throughput is the actual level of raw data across a network. In other words, a measurement of how much real data can be sent from one point, and received at another point of a network.

The network path may involve multiple links and channels, through multiple intermediate devices as routers and switches. Throughput can never be greater than the bandwidth.

Reasons why the throughput is lower may for instance be that errored packets have to be retransmitted. The throughput parameter depends on the number of hops parameter, but no throughput parameter simulation will be designed in the platform.

2.11 Number of Hops

Between two nodes the number of hops is the number of links in the path chosen by the routing algorithm. This parameter is important when designing a large-scale network according to the throughput, bandwidth, latency/delay, propagation time, and other parameters. A number of hops simulation will be available in the simulation platform.

2.12 Summary

This chapter has defined some basic network performance parameters. In order to design a simulation tool within the proposed time schedule, only a few of these parameters is decided to be implemented in the simulation platform. A conclusion of this chapter is that the simulation platform, which is intended to be designed should be able to simulate the maximum flow(network flow), traffic load, and number of hops in a network. The idea behind the traffic load simulation is that one node is sending one packet to another node, or sending one packet to every other node in the network, and then calculate some traffic parameters. The choice of these parameters are a combination of the importantness of these parameters and of making the task of implementing the tool realizable in the proposed time schedule.

Chapter 3

A "Perfect" Simulation Tool

This chapter describes what will be appropriate elements in a good simulation tool. It is done in such a matter that it gives an insight in how complex a simulation platform should be in order to satisfy the development of networks. A tool fulfilling all these elements in this report is referred to as a "Perfect simulation platform". When developing large-scale networks a lot of parameters must be taken into consideration in the modeling and design phase, using a simulation platform. In this chapter some queuing theory will be described, i.e. more in general the Poisson process.

Due to the complexity in designing modern large-scale networks, there is set higher demands for simulating network configurations during development of the networks. Therefore, a description of such a perfect simulation platform will now take place in relation towards the goals for the simulation platform which is designed later. The following is listing subjects related to a simulator, which will be discussed in this chapter.

- Queuing System in Networks.
- Traffic Monitor.
- Human Computer Interaction.
- Random Traffic Generator.
- Different Protocols.

3.1 Queuing Systems in Networks

In networks, data packets through a node must be handled by the node itself and proceed on if for example the node is a router somewhere in a network. When data packets comes into a node, the packets meet a buffer which will handle the packets when the node is ready to perform this service. If a new packet arrives in the same buffer and the old packet is still in the buffer, a queue will now start building up and become larger for each packet coming in.

Filling and emptying a buffer is the theory of queuing systems. This will be concerning the amount of buffer content, that is the queue length and the amount of time spent in each of the elements in the buffer while passing through the queue.

The simplest representative of queuing networks is the tandem queuing network depicted in figure 3.1.

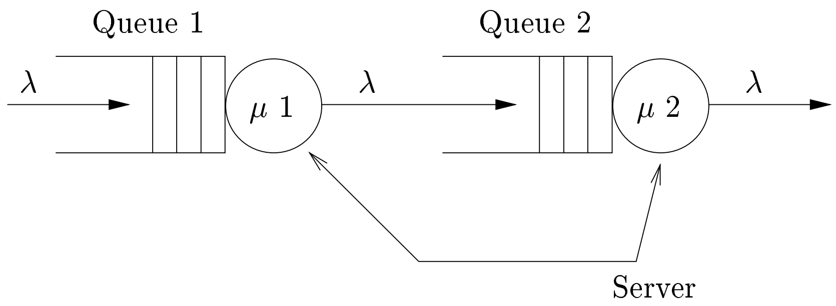


Figure 3.1: The Tandem Queuing Network

Figure 3.1 illustrates a Poisson arrival process with rate λ and it is called a single queuing system with one arrival process. The figure is a simple network involving only two nodes, such a system includes the M/M/m/K/L queues, which is the Kendall notation.
The meaning of the notations is explained in table 3.1.

Identifier	Description
M	M is exponential and it means memoryless, because if the inter arrival time is exponentially distributed with mean 1/L, the expected times to the next arrival is always 1/L regardless of the time since the last arrival. Therefore, it leads to be called as the memoryless distribution [3].
m	is the number of servers
K	is the population size
L	is the service discipline

Table 3.1: The Kendall notation.

3.1.1 The Poisson Process

The Poisson process[1] of k stream with the mean rate λ_i results in a Poisson stream with the mean rate λ given by:

$$\lambda = \sum_{i=1}^{k} \lambda_i \tag{3.1}$$

Equation 3.1 counts the number of arrival processes. The parameter λ is the intensity of the process and it is the only quantity that may distinguish two processes. Therefore, the higher the intensity is, the more arrivals per time unit, i.e. the more intense an arrival process. The value of a Poisson counting process at earlier times $t > t_1 > t_2 >, ..., > t_n$ is equal to the value $t + \delta \cdot t$ when the value is given at the time t, this means that the Poisson process is memoryless [4].

It can be concluded that designing a simulator with the use of queuing theory gives a simulation platform a much more realistic usefulness for developing or analysing network structures.

3.2 Traffic Monitor

Here is a short description of a traffic monitor[2] and the use of traffic analysis discussed. In general a monitor is attached to a network which gives the user information about specific events and reports statistics such as average number of frames per second and the average frame size. In other words a monitor is a device which listens to all traffic in a network, e.g., LAN. Such a traffic monitor also must be able to sniff packets which are encrypted. The software is an analysing program. In the Traffic Monitor program the user can configure different parameters which are used when the program is running to analyse packets in a given network. The monitor program will not interrupt the packets following in the network, it will only take a copy of the packets detected by the program. The idea behind the monitor program is that it is possible for the program to analyse the packets header and payload frame to identify e.g. the packets source and destination [5].

The network monitor is a very useful platform in the case of detecting faults in a network, e.g. if a node is down and unable to send or receive packets. Thus a network monitor could be a good platform for determining the networks uptime and finding faults in a given network configuration. A network monitor must also be able to provide the user or designer information by:

[1] The Poisson process is also called a *point process*.
[2] Also called a network analyzer or sometimes a network sniffer.

- Protocols and port usage.
- File types transferred.
- User IP addresses.

3.2.1 Remote Monitoring (RMON)

Remote Monitoring also called RMON is a very useful platform in traffic monitoring. RMON gives the network administrator information from the RMON Management Information Base (MIB). The RMON became a standard in 1992 as RFC (for Ethernet) [6]. This platform is able to give informations such as graphing, alarm, logging, and reporting capabilities or health. The network monitor which must perform the sniffing must consist of the following two components:

- A physical component which measures the packets flowing pass, this is connected to the physical medium.
- The processor which analyses the gathered data.

Analysed data is transmitted through a network to a remote network management station. Therefore, monitoring a network in such a case is called Remote Network Monitoring.

Now the question is how it is possible to implement a network monitor in a simulation platform for network developing. This must be a hard task and is out of the scope for the simulation platform to be developed. It can be concluded that such a feature in a network simulator must run on some hardware in order to make the simulation possible, and for verifying the correctness of the simulation. Furthermore, a network monitor must be one of the most important features in network management systems.

3.3 Human Computer Interaction

A perfect and a successful simulation platform must have a graphical user interface (GUI). This is needed in order to make a user friendly interaction between the user and the platform. But developing human computer interaction (HCI) is time consuming and requires some analysis of the users demands. This is in order to get a user friendly platform, from the user's point of view. Therefore, this section will give an overview of what is needed in such a GUI in order to provide the user with the needed information and usage.

For a simulation platform the following opportunities and information have to be given in a user interface in order to satisfy a user:

- Graphical Editor.
- Status bar.
- Help.

3.3.1 Graphical Editor

The possibility of drawing a topology virtually in a GUI is concidered as a high demand from the users perspective in order to enhance the usability and performance of the simulation platform.

3.3.2 Status Bar

The status bar provides information to the user. For a user will it be appropriate to know the estimated remaining simulation time. It will also be convenient for the user to see a virtual bar or counter showing that the simulation is actual running and thus in progress. For the use of time, an accurate reference clock is required. This allows time stamping of traffic sent or received and measurement of quality of service (QOS) parameters over the network. Therefore, a status bar is a must in a simulation platform, to inform the user what the platform is doing.

3.3.3 Help

In order to give the user at perfect program and thereby a useful HCI the platform must provide a help function. The help function can be a description of how to use the platform, such as a user manual for the user.

3.4 Random Traffic Generator

Different users are sending different amounts of data through the network. It would be convenient to simulate this by using a random traffic generator (RTG), which gives the simulation of a network configuration a more realistic appearance from the users point of view. Therefore, in order to make a realistic traffic generator, the RTG has to be running in real-time. The generator must have a large set of different parameters and manage simultaneous IP connections (TCP or UDP). Furthermore, a random traffic generator can in practical approaches, e.g., be used in a simulator to test performance in network devices, such as routers, to ensure they are working correctly.

3.5 Protocols

This section states two protocols for simulations in a simulation platform. It could be convenient to simulate a given network with different network protocols. The most basic protocols to use in a simulation platform for network are the following:

- TCP.
- UDP.

These protocols are chosen since they are the two main protocols used for the Internet. The protocols are connection less (UDP) and connection-oriented (TCP) protocols respectively[3].

Furthermore, there must be some network control protocols (NCP) for each network layer supported. The reason for simulating different protocols is to test differences in performance in the traffic parameters.

3.6 Summary

In this chapter some different parameters for developing a simulation platform have been described and discussed. It can be concluded that the most important issue in developing a simulation platform has to be clear; namely that the platform must be able to be scaled. Therefore, a platform programmed in a language which supports the ability to be expanded is a strict requirement.

[3]UDP is the abbreviation for User Datagram Protocol and TCP is the abbreviation for Transmission Control Protocol

Chapter 4

Designing a Large-Scale Simulation Platform

In this chapter the motivation for designing a large-scale network is described. The reasons for building such a tool and if it is worth the effort is answered in the proceeding section. Furthermore, the requirements of the simulation tool, and ways of representing graphs are described. The last section consists of a discussion of the programming language, which is going to be used to design and implement the simulation tool.

4.1 Motivation

The first thing to do is to define simulation:

- A system that represents or emulates the behavior of another system over time. A computer simulation is a simulation where the system doing the emulating is a computer program [7].

To make this clearer, a computer simulation is a computer program that models the behavior of a physical system over time. The program variables (state variables) represent the current state of the physical system. A simulation program modifies state variables to model the evolution of the physical system over time.

In other words, a computer program that simulates a network is the goal for this project. But why do simulations? In designing a network, simulations make experiements less costly than real world experiments. The simulation would be used as a decision tool after testing different topologies for some parameters.

A network simulation tool is used for evaluating network hardware, software, protocol and services.

Parallel simulation is used in network simulations. Parallel simulation refers to the technology concerned with executing computer simulations over computer systems containing multiple processors, which can be:

- Tightly coupled multi processor systems.
- Workstations interconnected via a network.
- Hand held computers connected by wireless links.

The reasons for executing over multiple CPUs are:

- Reduced model executing time, up to n-fold reduction using n CPUs.
- May not have enough memory on a single machine.
- Scalable performance, maintaining the same execution speed for bigger models by using more CPUs.

Network simulation is an indispensable tool for testing network performance parameters and it is a very useful tool in network planning.

4.2 Limitations and Requirements for the Simulator

Elements that would be appropriate in a good network simulation tool is presented in chapter 3 on page 10. Developing a platform containing all these elements is a huge task, and will be nearly impossible in the time scope of this project. To make the implementation of the platform manageable, some limitations are made.

The simulator must have a pure text interface, in a command prompt. Menus will navigate the user through the program. An overview of the program flow through the menus is depicted in figure 7.1 on page 34. Commands from the user are only given from keyboard, – no use of mouse.

A network can be generated by entering the number of nodes, and then select the node capacity. The node capacity can be entered manually for each specific node, or automaticly assigned with a given value. The user must be able to do the generation of links between the nodes in two ways, both with automatic or manual assigning of capacity; either by automatic generation of standard network structures or by manually entering. The platform must be able to generate five standard network structures: a simple ring, fully connected, wheel, double ring and N2Rpq. These network structures will be described later in this report. There must be a possibility that the platform could randomly generates links between a given number of nodes.

If the manual link generation is used, and some nodes are not assigned any link when the user ends link generation, the user must be notified of this. An editor for creation, editing and presenting networks in a graphical manner is out of project goal.

The representation of a network must be as lists of links and nodes. These lists contain central data for the network elements, such as node number, capacity and load for nodes, and start node, stop node, capacity, and load for links.

When a network is created, three files must be created. One for node, one for link, and one for result data. When a network is opened, the user must be able to edit the network; remove and insert nodes and links, and to change their capacity. If a node is removed, the simulator must check if any link is connected to this node, and remove them if so is the case. If a node is created, the platform must also here alert the user if no link connects this node to the rest of the network.

Then the platform must be able to do some simulation of networks. A lot of different simulation functions can be implemented, but again due to the limitations in time, only the following three simulation tasks will be implemented:

- **Number of hops** – This function must count number of hops (NoH) from every node to all others in the network, and use this data to calculate the sum of NoH, average NoH and maximum NoH.
- **Traffic load** – In this simulation the user must have two options, either to simulate the load in the network when sending a packet from a given node to another given node, or when sending a packet from a given node to every other node in the network. In the latter option, the simulator must route the packets in a clever manner where the shortest path is used. Where several paths has equal distance, the load must be distributed on links with lowest load. The output of this simulation must be sum of, maximum and average values for node, and link load.
- **Maximum flow** – To find the maximum data flow from a given source node to another given destination node.

During the simulation, a status bar should show the elapsed time, an estimate of remaining time and a percentage of simulation progress must be currently updated and printed to the screen.

It must be possible to create projects, and include networks in these projects. Furthermore must it be possibilities for plotting the simulation results, where there is one plot for each parameter in the result file. All networks in the project must be in these plots, to be able to compare their performance.

4.2.1 Summary

In order to make the implementation of the simulation platform a manageable task in the given time schedule, some simplification compared to a perfect network simulation tool is made. There is no graphical user interface, only a menu based, keyboard entry interface through a command prompt. Only the three simulations described above will be implemented. The queuing technologies at the nodes, and a random traffic generator to simulate an "actual" data flow in the network will not be implemented.

4.3 Representing Graphs

There are several appropriate ways to represent a graph for implementation in a simulation program. Adjacency matrices, adjacency lists and link lists are discussed here.

In an adjacency matrix is a link represented by a one. If a link is connected to node 1 and 2, there is a one in matrix element [1][2] and [2][1] if the link is bidirectional. Between nodes with no connection there is a zero. Practical applications of graphs usually requires that they be annotated with additional information, a label. Such information may be attached to the links of the graphs. An undirected graph with labeled links can represent geographic information in which the nodes represent geographical locations and the links represent possible routes between locations. The label on the links can then represent distances between the end points. Figure 4.1 shows an example of such a labeled undirected graph.

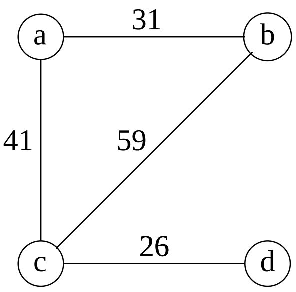


Figure 4.1: Undirected graph with labeled links.

To represent a labeled graph the adjacency matrix again is used, but not with 0 and 1 to represent connection or not. Here with the labels to represent connection and ∞ to represent no connection. The adjacency matrix for the graph in figure 4.1 is shown in table 4.1.

If O is the space needed per element in the adjacency matrix and take in consideration that the adjacency matrix has $|V|^2$ entries, the amount of space needed to represent the links of a graph is $O(|V|^2)$. This amount of space is regardless of the actual number of links in the graph. If the graph contains relatively few links, $|E| \lll |V|^2$, then most elements of the adjacency matrix will be zero (or ∞). An adjacency matrix in which most of the elements are zero (or ∞) is called a *sparse* matrix. This leads to that a *sparse* graph is

Nodes	a	b	c	d
a	∞	31	41	∞
b	31	∞	59	∞
c	41	59	∞	26
d	∞	∞	26	∞

Table 4.1: Adjacency matrix for an undirected labeled graph.

a graph with relatively few links. A graph with many links relatively to the number of nodes is called a *dense* graph.

Adjacency list is also often used to represent a graph. This method of representing graphs uses $|V|$ linked list, one for each node. The linked list for a node contains the set of nodes adjacent to the specific node. An example of an adjacency list representation is shown in figure 4.2. The labels are not considered here.

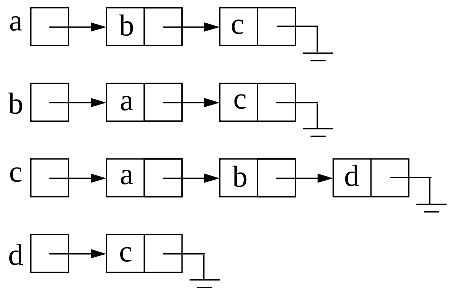


Figure 4.2: Adjacency list representation.

In figure 4.2, notice that the total number of list elements used to represent an undirected graph is $2 \times |E|$, therefore the total space required for the adjacency list is $2 \times O(|E|)$ for undirected graphs. If the number of nodes is relatively small compared to the number of links an adjacency list occupies less space in memory than the adjacency matrix.

The link list representation is containing the source node and sink node for each link. If a label is wanted to be included there have to be an extra element containing the label. An example of a link list is shown in table 4.2.

In table 4.2, is the the total space required for the link list with label is $3 \times |E|$ for undirected graphs.

The link list is decided to be used in the platform because it is easy readable from a file, and it easy to expand it to give the links more properties such as different types of labels.

Source	Sink	Label
a	b	5
a	c	2
a	d	1
b	c	4
c	d	2

Table 4.2: Link list for an undirected labeled graph.

4.4 Discussion of Programming Languages

To implement a large-scale network simulation tool, many programming languages can be used. The first thing is to decide whether structured or object oriented programming should be used. Object oriented programming is decided to be used for several reasons. One of these reasons is that it is easier to work in a group with object oriented software projects because the different tasks is divided into smaller parts.

Which object oriented language is best fitted to solve the problems during designing a simulation tool? Java was first considered because of its simplicity and the easiness of making a graphical user interface. But Java has a disadvantage of run time speed, and the speed is an important parameter in a large-scale simulation tool. Therefore, instead of Java the C++ is considered. In C++, it is relatively more difficult to implement a graphical user interface, and it is generally a more difficult programming language for non experienced software designers. In spite of this C++ is chosen because of the fast running time.

4.5 Summary

This chapter begins with reasons for building a simulation tool, and continues with some requirements for the simulation tool in which is going to be implemented. After this different ways of representing graphs is described, and it is decided that the simulation tool is going to use link lists to represent a network. Then the reasons for the choice of the programming language comes in the last section. The choice of C++ leads to an object oriented analysis and design, which will make the C++ coding easier, and give a better overview of how to solve the different problems. This procedure is described in chapter 5 and chapter 6.

Part II

Development of the Simulation Platform

Chapter 5

Object Oriented Analysis

Working with an object oriented software project, it is proper to build a model in order to manage the complexity. The goal of the model is to create a proper abstraction of the real world. This section is mostly based on [8]

The iterative design progress is listed below.

- Conceptualization
- Analysis
- Design
- Implementation
- Testing
- Roll out

The Conceptualization and the Analysis is covered in this chapter.

5.1 Conceptualization

The conceptualization for the simulation platform is:

Design a simulation tool to test performance in large-scale network topologies. The test parameters are number of hops, traffic load, and maximum flow. It must be possible to make changes in the network topology to optimize the performance.

5.2 Requirements Analysis

To move on to the analysis part, an understanding of how the product will be used and how it must perform is necessary.

5.2.1 Use Case Analysis

A use case is a high-level description of how the product will be used. It is a description of the interaction between an actor and the system itself. An actor is any person or system that interacts with the system being developed.

Actors for this system are the user and the platform.

Use cases:

- The user creates a new project.
- The user opens a project.
- The user closes the project.
- The user specifies a new network in the project.
- The user adds an existing network to the project.
- The user opens an existing network in the project.
- The user removes an existing network from the project.
- The user makes changes in a network.
- The user will auto generate a network topology.
- The user closes the active network in the project.
- The user selects wanted simulation parameters and starts simulation.
- The user stops the simulation.
- The platform plots the simulation results from the different networks.
- The platform shows a status bar during simulation.

5.2.2 The Domain Model

The domain model captures all knowledge of the domain which one is working in. A part of the domain model will contain domain objects, which describes all objects in the use cases. This is not a description of the design objects, but rather the objects in the domain.

Domain objects in this system are: user, platform, project, network, nodes, links, status bar, simulator, simulation results, files, and plot.

Before deriving the domain model, the attributes and behavior of the objects are listed:

- **User**, Attributes: Actor.
- **Platform**, Attributes: Actor and Name. Behaviors: close the platform.
- **Project**, Attributes: Name. Behaviors: Define parameters, create, save, close and open a project.
- **Network**, Attributes: Name, structure. Behaviors: create and edit network.
- **Nodes**, Attributes: Number, capacity, input-output-transit load.
- **Links**, Attributes: Number, source node, sink node, capacity, connection, load.
- **Status bar**, Attributes: size, color, time. Behaviors: update/start/stop status bar
- **Simulator**, Attributes: runtime. Behaviors: start/stop simulation, simulate number of hops/packets per second/traffic load.
- **Simulation results**, Attributes: data, simulation time.
- **Files**, Attributes: Name, extension. Behaviors: open/close file.
- **Plot**, Attributes: x/y coordinates, color. Behaviors: create chart.

Relations

Generalization describes the relationship between the objects, that one object can be a subtype of another. It is often equated with inheritance in programming, but there is a clear difference between the two. Inheritance is the programming implementation of generalization.

As in the example given above, one object is composed of many sub objects. There is a *has a* relationship, the network *has* nodes and links. This is called *containment*, and is illustrated in figure 5.1, showing the containment relationship, as an arrow with a diamond in the tail.

Another relationship is *association*, which suggests that two objects know of one another, and they interact in some way. An example is the user and the platform. At this state, their interaction is just suggested. In figure 5.1, *association* is illustrated with a straight line. It is important to mention that 5.1 is not design objects, but rather objects in the domain. This is a documentation of how the world works, not documentation of how the system will work. The design objects will be documented in chapter 6 on page 30.

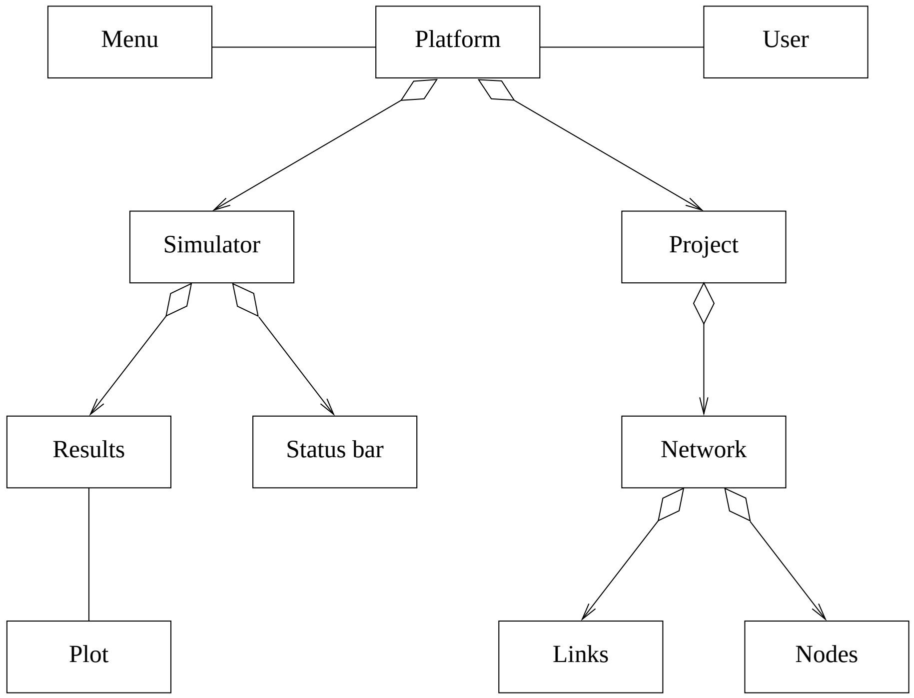


Figure 5.1: Generalization in the simulation platform.

5.2.3 Scenarios and Guidelines

The use cases have to be given more depth by breaking each of them into different scenarios.

Guidelines is used to document each scenario to test that if each scenario includes the following:

- Preconditions - what must be true for the scenario to begin.
- Triggers - what causes the scenario to begin.
- What actions the actors take.
- What results or changes are caused by the system.
- What feedback the actors receive.
- Whether there are repeating activities, and what causes them to conclude.
- A description of the logical flow of the scenario.
- What causes the scenario to end.
- Postconditions - what must be true when the scenario is complete.

In the following one use cases is analysed; with the scenarios first and the guideline itemized under the scenarios.

The User Creates a New Project

- User choose "create new project" from the start menu, the platform asks for a name, the user types a name, the platform accepts name and a new project is created.
  - Preconditions: The platform is in the start menu.
  - Trigger: The user creates a new project.
  - Description: The user choose "create new project" in the start menu, and give the project a unique name and the platform creates a project with the given name.
  - Post Conditions: A new project is created, and the platform enters the project menu.
- The user choose "create new project" from the start menu, the platform asks for a name and the user types a name. Then the platform finds an equal project name and asks for a new name. The user types a new name, the platform accepts the name and a new project is created.
  - Preconditions: The platform is in the start menu and projects with equal name exists.
  - Trigger: The user creates a new project.
  - Description: The user choose "create new project", and give the project an equal name as an existing project. Then the platform asks for a new project name, the user gives the project a new name and the platform creates a new project.
  - Post Conditions: A new project is created, and the platform enters the project menu.

The rest of the guidelines of the remaining use case scenarios and guidelines can be read in appendix B on page 108.

5.3 Summary

The object oriented analysis has a main goal: To make the software designer sure that he or she understands the customer‘s needs. This chapter has analysed the task of designing a simulation tool in a very abstract and detailed way. In order to get an overview of the problems in implementing this tool, the object oriented analysis is a helpful part. The next chapter describes the object oriented design, this is the last step before the implementation of the simulation platform.

Chapter 6

Object Oriented Design

The analysing part is described in chapter 5 on page 25. The analysis focuses on understanding the problem domain, whereas design focuses on creating the solution. This chapter will describe the process of transforming the understanding of the requirements into a model, which can be implemented in software. This will focus on the object oriented design (OOD).

When programming using C++, classes are to be created. Design classes and C++ classes are isomorphic, each class in the design will correspond to a class in the code. On the other hand it is possible for the design classes to be implemented in other languages than C++.

From the scenarios in section 5.2.3 on page 28 the following classes can be pulled out:

- The Project - contains the different networks to compare in a simulation.
- The Network - a collection of nodes and links.
- The Simulation - is responsible for starting the specified simulation and all other actions when a simulation is started.
- The Links - contains a list of connections, capacities and load.
- The Nodes - contains number of nodes, capacities and load.
- The Number Of Hops - starts the number of hops simulation.
- The Traffic Load - starts the traffic load simulation.
- The Maximum Flow - starts the maximum flow simulation.
- The Status Bar - shows the progress during simulations.

6.1 The Static Model

The static model focuses on three areas of concern: responsibilities, attributes and relationships. The responsibility is the most important part, and the guiding principle here is that each class must be responsible for one thing. CRC cards can be used to get a handle on the responsibilities of the classes. CRC is abbreviation for Class, Responsibility and Collaboration, and is used to get an overview of the primary responsibilities of the initial set of classes. At this point there is no focus on relationships; the class interface or which methods will be public and which will be private. The focus is only on understanding what each class is doing. After a CRC card session the CRC cards are to be transformed into UML. Responsibilities are translated into class method and whatever attributes captured are added as well. The program Together is used to implement a C++ code from the UML classes. All the UML classes are written into this program and the aggregation is specialized. The diagram written in Together is shown in figure 6.1, just showing the classes and the aggregation.

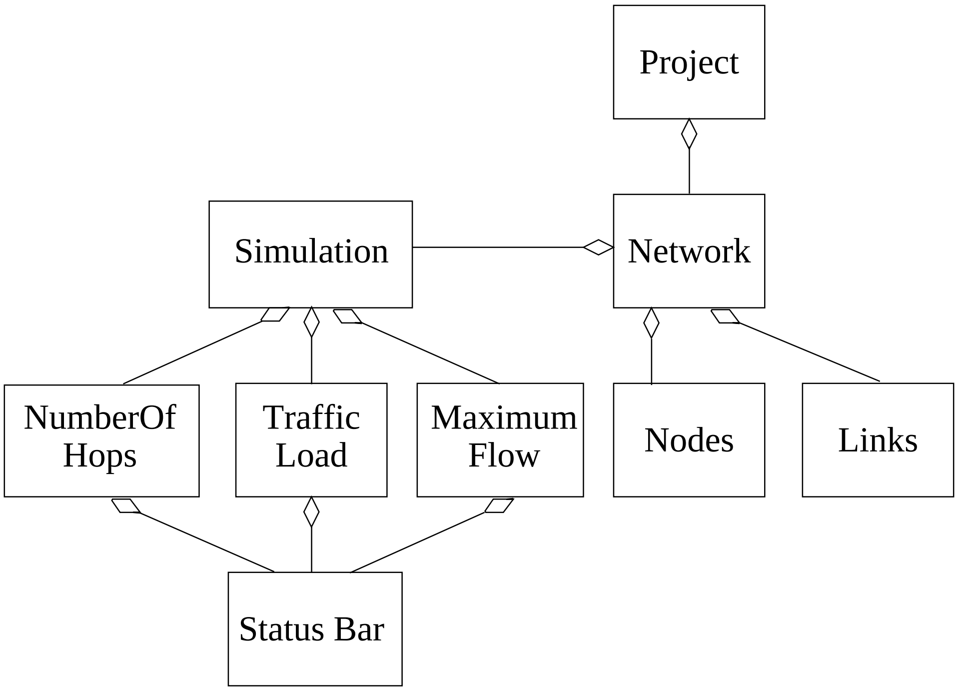


Figure 6.1: The Together design (without the class methods).

The methods and variables in the classes are specified in Together. Figure C.2 in appendix C on page 117 is showing the UML diagrams for the project, the network and the simulation class. The + and - in front of the data members means public and private data members, respectively. The first data members are class variables, and the data members under the line are class methods.

The Project class has methods, in order to have control of the creation and opening and saving of projects. A project is a collection of networks and different results from simulations of these network. From here new network can be be created and existing networks can be added or removed from the project.

The Network class has a control of creation of network. Also manipulation of a network can be performed with the methods *AddNodeToList()*, *DeleteLink()*, *DeleteNode()*. The connection of the nodes can be typed in manually, or connected automaticly by the Network class. The capacity of the nodes and links can be typed in for each node and link, but also by typing the same capacity for all nodes and links. This class contains a check of the network, e.g. it checks that all nodes have a connection to one or more other nodes. The network class initiates three different files when a network is created. For example if a network is named Petersen, the Network class makes petersen.lnk, petersen.nod and petersen.res. These files contains information about the network, one file for link information, one file for node information and one file for simulation results. From the network class the simulation class can be called in order to simulate the "active" network.

The Simulation class allows simulation for the parameters max flow, number of hops and traffic. The class has methods for reading and writing in the result file in order to save the results. It has also a *CountLinks()* and *CountNodes()* methods which are used to find the arguments in the specific simulation classes.

Figure C.1 on page 117 in appendix C is showing the UML class diagrams for the nodes and the links classes. The Nodes class has different methods for setting and getting the node parameters like node number, node capacity and loads. The Links class has almost the same methods for the links.

Figure C.3 in appendix C on page 117 shows the UML class diagrams for the traffic load, the number of hops and the maximum flow classes. The Traffic load class contains methods which are needed to perform a traffic load simulation. The method performing the traffic load algorithm is *SimulateTrafficLoad()*. This algorithm will be more explained in the implementation part in chapter 7. This will also be the case for the max flow and the number of hops classes.

Together generates the class header files for all the classes which are defined in figure 6.1. It has made the platform for further C++ coding, and this will be described in the following chapter.

6.2 Summary

This chapter has concerned about the object oriented design, it has modeled the object oriented model in which to be implemented. Since Together has generated all the class header files, then so the class definitions can be coded now. The next chapter describes the process of implementation of the simulation tool. The development of the simulation algorithms will be described more detailed, since these algorithms are considered as the most important part of the simulation tool.

CHAPTER 7

IMPLEMENTATION OF THE SIMULATION PLATFORM

Chapter 6 modelled the simulation tool and the Together program prepared the class header files for implementation. This chapter describes the process of implementing the class definitions and the main program with a menu. In order to get help in the implementation, [9] and [10] is mostly used. The source code can be found on the enclosed CD-ROM[1].

In order to have a clue how the user should use the simulation tool, a proposal to a user interface is made. It is not a graphical user interface, but a simple menu interface. Figure 7.1 shows this menu interface and the flow when using the program.

7.1 The Platform

Platform contains the *main()* function, and is therefore the driver for the simulation tool. The main menu of the program is implemented here, and from this the user is able to create a new or open an existing project, or to exit the program.

The create new network option initializes an object of type Project, and calls the method *CreateNewProject()* for this object as described in section 7.4. To open a project, a similar procedure is used.

7.2 The Links and Nodes Classes

When developing a network simulator with object oriented programming, it will be appropriate to have a class for Links, and another for Nodes. It is then possible to model nodes and links with their relevant properties.

When initializing a network, three files are created. The name of these three files are the same as the name of the network. Their extension are .lnk, .nod

[1] CDROM/Sourcecode

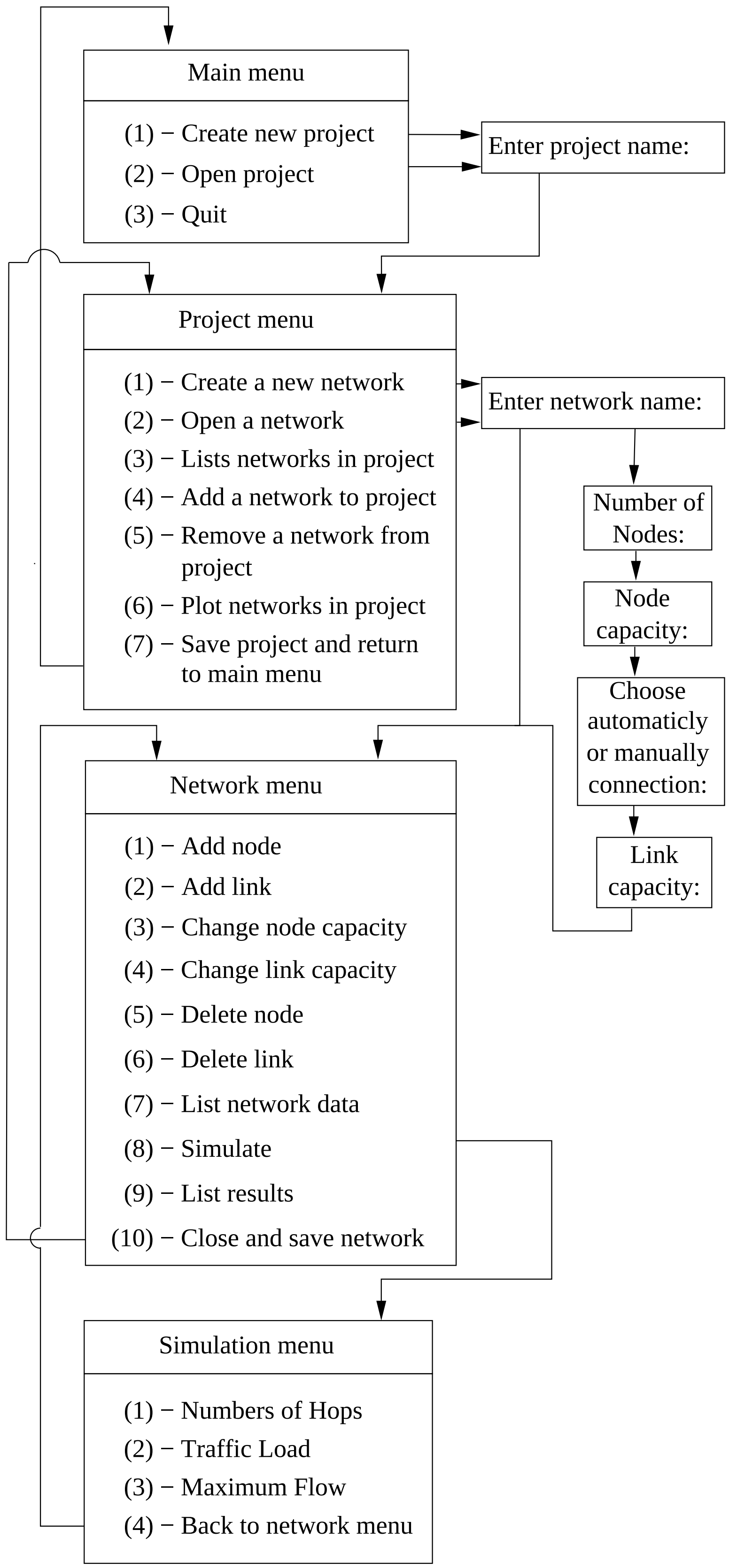


Figure 7.1: Program flow.

or .res as if they contain link-, node- or simulation result data as described in the previous chapter.

A network will typically contain several nodes and links. But how many? The nodes and links have to be ordered somehow in the network. One of the solutions could be to use arrays. But then it will be needed to specify a length of these, and the network will not be able to contain more elements than these given lengths. Of course it is just to use large array lengths that is never exceeded, but then a lot of memory will be occupied to no use. The solution to this problem is to use pointer lists. Each network element (node or link) includes a pointer to another element of same type. Then the elements are chained together, and a dynamic data structure is obtained. An example of this is shown in figure 7.2. In figure 7.2 a list of links is shown. The pointer *LinkListPtr* points to the first elements in the list. To be able to edit the list, inserting new and deleting links, a pointer to the last element is needed. From the figure it can be seen that every element has a pointer of data type *Links* which points to the next element in the list. The pointer in the last element is set to point to *NULL*.

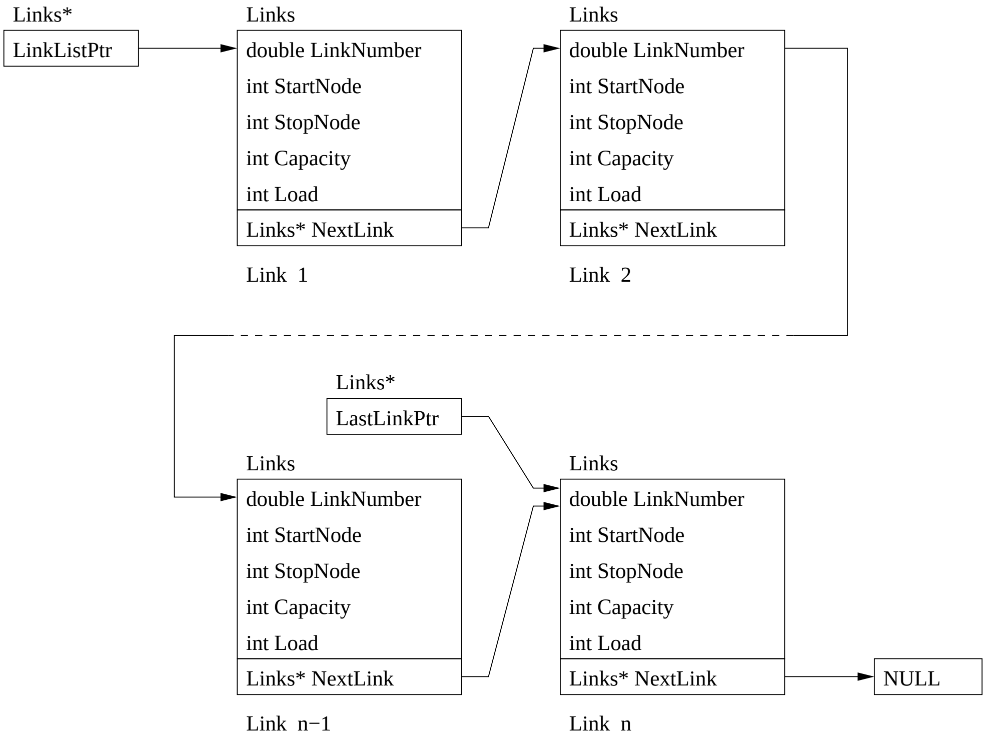


Figure 7.2: Pointer list of links.

Figure 7.2 shows the private data members in the Links class. It is possible to edit the data, using set- and get methods, for any link in the list by first initialize a temporary Links-pointer and assigning this the value of *LinkListPtr*. The temporary pointer can then be manipulated to point to link_i by assigning it the value of *NextLink* in link_{i-1}, and link_i can be edited.

The constructor for the Links class has four arguments, the first argument is a link counter, the two next are the node numbers in both ends of the link, and the fourth is for setting the capacity. If the capacity argument is zero, the constructor will ask the user to enter a capacity.

A similar structure is used in the Nodes class. The private data members in this class are properties for the nodes that is relevant for the simulator: *int NodeNumber, int InputLoad, int Outputload, int TransitLoad, int Capacity* and *Nodes* NextNode*

The constructor in the Nodes class has two arguments, number, and capacity of the node. Also for a new node, the user will be asked to set the capacity if this argument is zero.

Both classes also contain a method for printing all data to the screen.

7.3 The Network Class

A large-scale network will typically have a lot of different elements, but in this simulator only the nodes and links are considered. The Network class therefore has to include the Links and Nodes classes to be able to create a network.

The constructor in the Network class initializes the *LinkListPtr*, and *NodeListPtr* to point to NULL.

To create a new network, the method *CreateNetwork()* has to be called for an object of Network type. The user will be asked to enter number of nodes, and to enter their capacity. The links can be arranged in standard structures as wheel and double ring, or by manually entering of start- and stop node for each link.

When a network is created, a menu appears on the screen. From this the user is able to edit the network, add, delete and change the capacity of nodes and links. Simulation of the network is also called from this menu. All network data, as node and link traffic, and simulation results, are, as mentioned before written to files. To be able to check this data while running the program, there is also possibilities in the menu for listing this data on the screen.

Before simulation, all network data is written to files, and deleted from memory using *DeleteNetworkMem()*. In this method a temporary pointer is set to point to the same address as *LinkListPtr*, and until this pointer points to NULL, which means the last element in the list, each link is deleted, see figure 7.2. A similar operation are done for the nodes. Each simulation reads the data in the needed files to run the simulation. Since the network structure is not manipulated during simulation, a dynamic data structure is not used there.

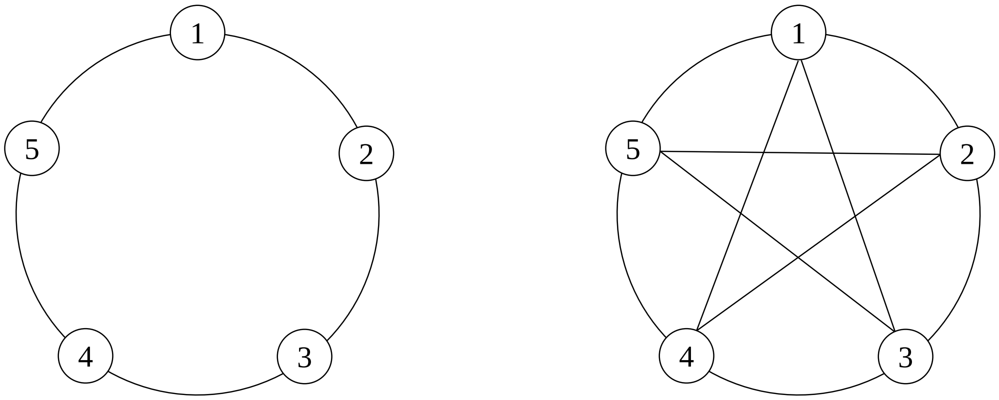
The number of nodes and links are given from the files, and therefore arrays of fixed lengths, the actual number of network elements, can be used.

If one wants to add a node to the list, the method *AddNodeToList(int)* is used. The argument is the node number that is controlled by a node counter for the network. A link can be added with *AddLinkToList(int, int, int)*. The three arguments are link number, start node and stop node, respectively. Before this method is called, there are possibilities for checking if the nodes given as arguments actually exist in the network by *DoesNodeExist(int)* where the argument is the node number.

It is possible to delete a single link by *DeleteLink(int, int)*, the arguments are the node number in each end of the link. The method will check if the link actually exist by *SearchLink(int, int)*, before deleting it. A node can be deleted by sending a node number as the argument in *DeleteNode(int)*. The node number will be checked to confirm that the node actually exists. Then the method *CkeckLinks(int)* searches if any links are connected to this node, and these will then be deleted.

ListData() is used to list Network data on the screen. It calls *PrintNodes()* and *PrintLinks()* that list node and link data.

There are different methods for generating standard network structures. The simplest one is *CreateRing()*, which creates a links from node 1 to 2, 2 to 3, ... , $n-1$ to n, and at last from node n to 1, where n is the number of nodes in the network. *CreateFullyConnected()* will create a fully connected network, where every node have a link to every other node. Figure 7.3 shows an example of these structures for a network of five nodes.



(a)
Ring

(b)
Fully connected

Figure 7.3: Standard network structures.

The simulator has possibilities for generating three degree 3 networks, wheel, double ring and N2Rpq. When the nodes are connected in a ring, a wheel structure easily can be made. Links are added from node 1 to $n/2+1$, 2 to $n/2+2$, ... , $n/2$ to $n/2+n/2$, figure 7.4 (a). Studying this structure, and figure 7.4 (b), it can be seen that they are equal. Then a double ring structure, see figure 7.4 (c), can easily be made by manipulating two links. The Petersen structure is described in chapter 9, and is an example of an N2Rpq graph. Such structures can be generated by manipulating the links in the inner circle of double ring. The methods *CreateWheel()*, *CreateDoubleRing()* and *CreateN2Rpq()* handles this.

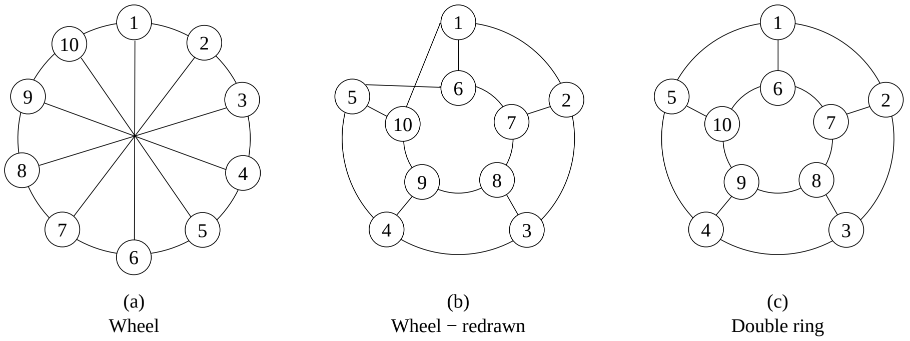


Figure 7.4: Degree three network structures.

The method *CreateRandom()* randomly creates links between the nodes. The user is asked to enter the maximum degree of (the highest number of links connected to) the nodes. The platform then generates links until every node is connected to the network. Figure 7.5 shows an example of such a network with 10 nodes and a maximum degree of 3.

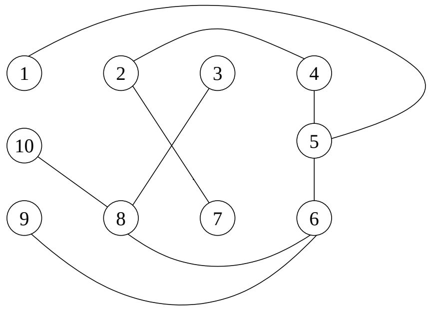


Figure 7.5: Random generated network.

The method prevents generation of links where start and stop node is the same, and copying of links generated before. However, the situation depicted in figure 7.6 may occur. If a maximum degree of 3 is set, and all nodes should be connected, the loop doing the link generation will never terminate in this situation. Node 3 can not be connected to the other nodes since the maximum degree is 3. Therefore, after a certain time, depending of the number of nodes, all links are deleted, and the link generation is restarted.

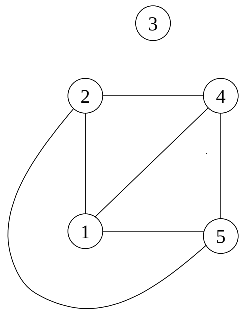


Figure 7.6: Possible problem in generation of random networks.

The last way to generate a network is to enter the links manually. If *CreateManually()* is chosen, the user has to specify the start- and stop node of every link. The method will ask for new links until '0' is chosen as start node, and then link registration is stopped. Then the network will be checked to verify that all nodes are assigned at least one link. The nodes that fail this test will be listed on the screen.

As mentioned in section 7.2, three files are created to contain network data. These are created by *CreateFiles()*. The user will be asked to enter a network name. If a network of equal name already exists, a new name has to be entered. The link and node files are just created and left empty. The .res file that will contain simulation data are initialized by *InitResFile()* with all values set to '0'.

A requirement for the simulation algorithms used in this simulator is that if there is a node with node number 1, and that the rest follows in descending order. To ensure this, *RearrangeNetwork()* is always called before writing network data to the files. If a "hole" in the list of node is found, the next node will be assigned this node number. The links that are connected to this node also has to be updated. This is done with *FixLinks(int, int)*, the first argument is the old, and the second is the new node number. The method searches trough all links and updates the connections. An example of this is shown in figure 7.7

This rearrangement of the network may be confusing for the user. If an actual, or a planned network is entered into the simulator, he or she might has an overview of the network in a geographical perspective. If then a node is deleted, the node numbers, that in this platform is the identifier of a node, may change, and the overview may be lost. A solution to this problem could be to use one extra parameter for each node, namely a node name. The the node number could be a internal parameter for the simulation platform, hidden for the user.

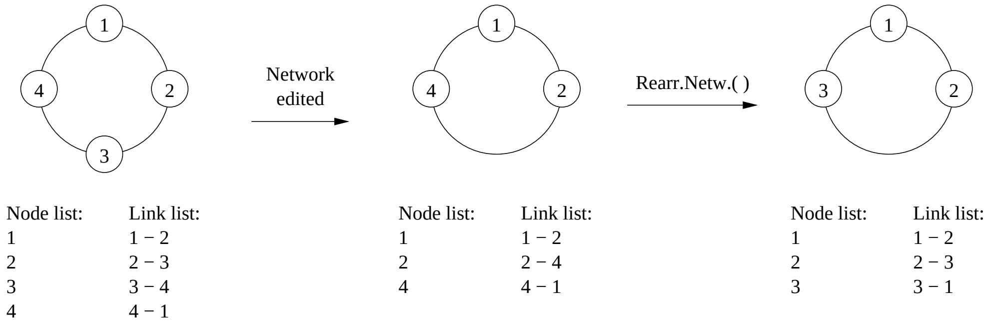


Figure 7.7: Use of the *RearrangeNetwork()* method.

When the network editing is finished, *SaveNetwork()* calls the methods *WriteLinks()* and *WriteNodes()* which writes the network data into files.

To open an earlier created and saved network, *OpenNetwork(char*)* is used. The argument is the network name. If no network with this name exist, the user is notified, and the operation is aborted. If the network does exist, the link and node files are read, and written into memory in the same structure as shown in figure 7.2.

There is also several set- and get methods to set and read the private data members of a Network object.

7.4 The Project Class

To be able to compare the simulation results from different networks, a Project class is included in the simulation platform. A project is just a simple list of networks. The idea is that one should be able to compare the networks in the list. In the main menu of the platform one has to create a new, or open an existing project. Then the project menu appears. From this the user is able to create new, or open an existing network. Other possibilities are to add or remove networks from project, and to list the included networks to the screen. An open project keeps the networks in a dynamic data structure. In the project menu is it also possibilities to call the plotter, in which plot the different networks included in the project according to different parameters. The plotter implementation will be explained in section 7.10.

The constructor of this class is very similar to the one in the Network class. This one initializes a *NetworkListPtr* to point to NULL, and sets the *Network-Counter* equal to 0.

A new project is created in *CreateNewProject()*, the user is asked to enter a project name, and if it exists, another name has to be entered. When a new name is created, an empty file with the given name, and extension .prj is created. Later the content of this file is a list of the networks included in the project.

The *OpenProject()* method is first asking the user to enter the name of the project to be opened, if there is no file with the given name, the operation is aborted. Does the project exist, the .prj file is read and the listed network is stored in memory, and the project menu appears.

If "Create new network" is chosen from the menu, *NewNetwork()* will initialize an object of type Network, and the *CreateNewProject()* method from Network class will be called for this Network object, and the network menu is entered. When the user goes back to project menu, the network is as described earlier saved to files, and deleted from memory.

Another possibility is to open a network, also here first an object of type network is initialized. The user is asked to enter a network name, and if the network exists the method *OpenNetwork(char*)* from the Network class is called. When the user is done with editing/simulating the network, the network is written to files and deleted from memory.

The two methods, *AddNetwork()* and *RemoveNetwork()* are used to add and remove networks from the list in a project. When the user closes a project, the network list is saved in the .prj file by *SaveNetwork()*. Then *DeleteProjectMem()* is used to delete the network list from memory.

From the project menu, there is a possibility for plotting the parameters from the result files. The actual parameter for each network in a project are depicted in the same plot, and hence the networks can be compared. The method *PlotResults()* reads all the results files in a project, and write only the actual simulation values to a new file, "plotdata.dat." Then it runs a Java application that reads the .dat file, and plots the results.

7.5 The Simulation Class

As described in section 7.1 the simulation of a network is started from the network menu. A simulation menu is entered, and the user can choose between the three simulations, number of hops, traffic load and maximum flow. Before the simulation menu is entered, the network is written to files.

Using the methods *ReadResultFile()* and *WriteResultFile()*, the Simulation class has the responsibility of read and write simulation results to the .res file. This operation is done at this level, and not in the specific simulations to avoid the possibility of that one simulation discarding data from another simulation. The class has a private data member for all of the different values in the .res file. When a simulation object is initialized, the *ReadResultFile()* method is called in the constructor, and the values in the result file is copied to the private data members mentioned above.

The argument in the constructor is the network name, this name is used to create string variables containing the name of the three files associated with a network. This class also takes care of counting the nodes and links in the network, which is done with the methods *CountLinks()* and *CountNodes()* that is also called from the constructor.

The Simulation class contains a method for each simulation type. An object of the wanted simulation is initialized by sending the required data for the constructor as arguments. The results of a simulation are copied to the results variables in the Simulation class by using get functions from the class of this specific simulation.

7.6 The Maximum Flow Class

In order to find the maximum flow in a network an algorithm made by Ford Fulkerson is used together with a breadth first searching algorithm. The task of finding the maximum flow can be summarized as this:

Given a directed link with a source and a sink and capacities assigned to the links, determines the maximum flow from the source to the sink. For each link, the flow must not exceed the link's capacity, and for each node, the incoming flow must be equal to the outgoing flow [11].

Since the network considered here have bidirected links, the links are made bidirected in this class with a simple for loop in the capacity matrix initialization. This is done by setting the capacity from e.g. node 1 to node 2 equal to the capacity from node 2 to node 1.

The Ford Fulkerson algorithm is also called Ford Fulkerson Labeling algorithm and it works like this [11]:

- Initialization: Let x be an initial feasible flow, e.g. $x(e) = 0$ for all in E.
- Flow Augmentation: If there are no augmenting path from s to t on the residual network, then stop. The present x is a maximum flow. If there is a flow augmenting path p, replace the flow x as:
  - $x(e) = x(e) + \Delta$ if e is forward arc on p.
  - $x(e) = x(e) - \Delta$ if e is backward arc on p.

  Where Δ is a minimum value of residual capacity on p. Repeat this step.

The definition of a augmenting path is:

A path with alternating free and matched links which begins and ends with free nodes. Used to augment (improve or increase) a matching or flow [12].

The augmenting path is stored in a array called *pred[]* in order to have control of the different path in which there is a flow. The capacity matrix is also initialized, this shows all the link capacities together with source and sink node. A flow matrix is also needed to have control of the flow in the actual links when going through the Ford Fulkerson algorithm.

The maximum flow class is supposed to be called from the simulation class. As arguments in call to the maximum flow class, number of nodes and number of links have to be sent as arguments in order to initialize the capacity matrix, the flow matrix, the color matrix and the link file reading. The source and sink node must also be sent as arguments in order to get the class a knowledge about from which, and to which node the maximum flow is to be calculated. The last argument is the link filename to the actual network. The maximum flow class has to know which file to read, in order to get the links source and start node and the different capacities. To summarize this, in a call to the maximum flow class, these arguments should be sent as arguments:

- Number of nodes.
- Number of links.
- The start node in which the maximum flow is wanted to be calculated from.
- The stop node in which the maximum flow is wanted to be calculated to.
- The filename to the links for the actual network.

The maximum flow class constructor calls its own method *max_flow()* in order to run the simulation when a Maximum Flow object is created. In the *max_flow* method the reading of the link is first performed by the *ReadLink()* method. This method is also taking care of a few other other operations than just reading the link file. One of them is setting up the capacity matrix, which is described above. Another operation the *ReadLink()* method is doing, is to rearrange the node numbers. This is done in order to set the start node to node 1 and the stop node to the last node in the network. For example, if the Maximum Flow class is to perform a maximum flow simulation from node 3 to node 7 in a network with 10 nodes, node 3 is switched to node 1, and node 7 is switched to node 10. Now, also the capacity on the link between node 3 and 7 has the capacity on link between 1 and 10. This node and link switch operation is done in order to reduce the Ford Fulkerson algorithm‘s running time.

After this, the flow matrix is calculated in the Ford Fulkerson algorithm. This algorithm incrementing the flow matrix until the target node is colored black by the breadth first search method. In other words, to find where to stop the Ford Fulkerson algorithm and stop incrementing the flow, a breadth first search algorithm is performed. The Ford Fulkerson algorithm is continuing incrementing the flow until the breadth first search is finished.

The idea behind a breadth first algorithm is to process all nodes in a given level before proceeding to next level. As a comparison of this is a depth first search defined as: process next level as soon as possible. Breadth first search can be used to test whether an arbitrary graph G with n nodes is connected. It can also be used to find minimum length paths in an unweighted graph from a fixed node n to all other nodes. Dijkstra's shortest path algorithm for weighted graphs used in the number of hops algorithm can be considered as a generalization of the breadth first search. An array called *color[]* is needed by the breadth first search. This array take care of coloring of nodes. If the source node is colored black, the breadth first search is finished.

To give an example is figure 7.8 showing a network which the maximum flow from node 1 to node 9 is wanted to be calculated.

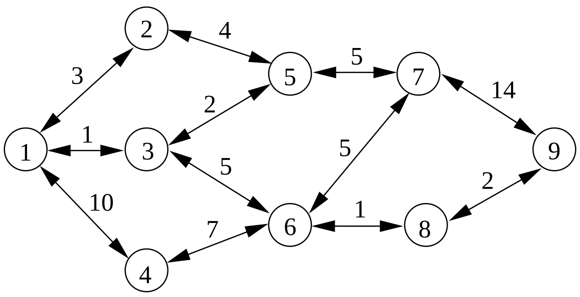


Figure 7.8: A network which the maximum flow is wanted to be calculated.

Figure 7.9 shows the results on running the maximum flow simulation on the network illustrated in figure 7.8.

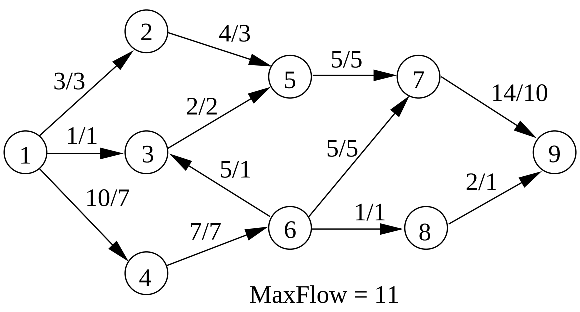


Figure 7.9: The result of running the maximum flow algorithm on the network in figure 7.8 , also showing the path for the maximum flow. The x/y is can be read as capacity/flow.

This max flow algorithm is not considering the node capacities, and in the most large-scale networks it is not the link capacities which will limit the flow trough the network, but actually the node capacities. So the max flow algorithm is not completed in this point of view. Furthermore, work on the platform can be including the node capacities in the calculation of the maximum flow trough a network.

7.7 The Number Of Hops Class

The number of hops algorithm finds the shortest distance between all nodes in the given network. It calculates a $n \cdot n$ distance matrix (where n is the number of nodes), showing the shortest distance between all the nodes. Many different algorithms, to find the shortest path from one node to another, are existing. The two most popular algorithms are Dijkstra's Algorithm and Bellman-Ford's algorithm.

The complexity of Dijkstra's algorithm is n^2, and the complexity of Bellman-Ford's algorithm is $m \cdot n$, where m is the number of links and n is the number of nodes. It is chosen to use the Dijkstra's algorithm, since all the networks considered in this platform have $m \geq n$.

The constructor for the NumberOfHops class has three input arguments
NumberOfHops(char LinkFile, unsigned long NOL, int NON).* The first is a pointer to the link file, the second is the number of links in the file, and the third is the number of nodes. For huge networks the number of links increases rapidly, for a full connected 1500 nodes network there is above one million links, which give the reason for making the data type an unsigned long.

The first thing that happens in the NumberOfHops class is reading the links from the link file into arrays. The links are all assumed to be bidirectional, as in the maximum flow algorithm, a matrix called *BidirectedLinks* containing all the links in both directions is made. Afterwards, an array of the nodes used is made.

The algorithm is explained by use of an example. Let's take a look at a wheel graph with 10 nodes as in figure 7.10.

A connection matrix *ConnectionMatrix* like table 7.1 is made from the *BidirectedLinks* array.

What is needed to calculate here is a shortest distance matrix, so a matrix *DistanceMatrix* is initialized and set equal to the connection matrix.

Now the actual algorithm starts. At first, it finds the node with the shortest distance to the start node, it is called *NearestNodeNumber.* For that node it finds out which nodes are connected to it. It then updates the distance matrix for the nodes connected to *NearestNodeNumber.* This is done for all nodes (n

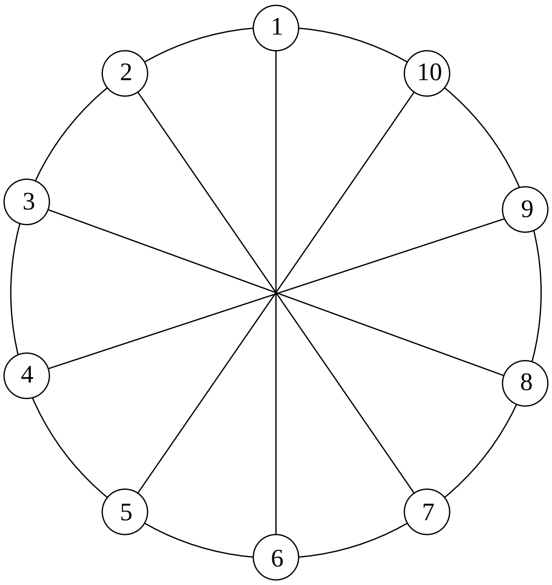


Figure 7.10: A 10 node wheel graph.

Nodes	1	2	3	4	5	6	7	8	9	10
1	0	1	∞	∞	∞	1	∞	∞	∞	1
2	1	0	1	∞	∞	∞	1	∞	∞	∞
3	∞	1	0	1	∞	∞	∞	1	∞	∞
4	∞	∞	1	0	1	∞	∞	∞	1	∞
5	∞	∞	∞	1	0	1	∞	∞	∞	1
6	1	∞	∞	∞	1	0	1	∞	∞	∞
7	∞	1	∞	∞	∞	1	0	1	∞	∞
8	∞	∞	1	∞	∞	∞	1	0	1	∞
9	∞	∞	∞	1	∞	∞	∞	1	0	1
10	1	∞	∞	∞	1	∞	∞	∞	1	0

Table 7.1: Connection matrix for a 10 node wheel graph.

times) and then the distance matrix will look like table 7.2.

Nodes	1	2	3	4	5	6	7	8	9	10
1	0	1	2	3	2	1	2	3	2	1
2	1	0	1	∞	∞	∞	1	∞	∞	∞
3	∞	1	0	1	∞	∞	∞	1	∞	∞
4	∞	∞	1	0	1	∞	∞	∞	1	∞
5	∞	∞	∞	1	0	1	∞	∞	∞	1
6	1	∞	∞	∞	1	0	1	∞	∞	∞
7	∞	1	∞	∞	∞	1	0	1	∞	∞
8	∞	∞	1	∞	∞	∞	1	0	1	∞
9	∞	∞	∞	1	∞	∞	∞	1	0	1
10	1	∞	∞	∞	1	∞	∞	∞	1	0

Table 7.2: Distance matrix after one iteration.

A for loop runs n times to update the distance matrix. Every time it runs it sets a start node and calculates the shortest distance to all other nodes (it updates one row at a time in the distance matrix). The Dijkstra's algorithm then runs n times.

It means that the actual complexity of this algorithm is n^3. The only things happening in the algorithm is adding and comparison, so it only takes a few minutes for a up to date computer to calculate this algorithm, since $n < 1500$ for this simulation tool.

When the distance matrix is made (see table 7.3) a variable *SumNumberOfHops* is made, containing the sum of all entities in the matrix added together, in this case it becomes 170.

Nodes	1	2	3	4	5	6	7	8	9	10
1	0	1	2	3	2	1	2	3	2	1
2	1	0	1	2	3	2	1	2	3	2
3	2	1	0	1	2	3	2	1	2	3
4	3	2	1	0	1	2	3	2	1	2
5	2	3	2	1	0	1	2	3	2	1
6	1	2	3	2	1	0	1	2	3	2
7	2	1	2	3	2	1	0	1	2	3
8	3	2	1	2	3	2	1	0	1	2
9	2	3	2	1	2	3	2	1	0	1
10	1	2	3	2	1	2	3	2	1	0

Table 7.3: Distance matrix when Dijkstra's algorithm is used.

The *MaximumNumberOfHops* is found to 3, and finally the *AverageNumberOfHops* is found to $170/90 \approx 1.89$.

These are the results for this simulation, and they are written on the screen when the simulation is finished and saved in the corresponding .res file.

This algorithm also calculates the maximum number of shortest paths a node has to another. The maximum number of shortest paths is used for the traffic load algorithm, and it will be explained in the next section.

Every second during the simulation, the number of hops algorithm updates the status bar. The status bar implementation will be described in section 7.9

During the simulation the user can press 'p' to pause the simulation and press 'q' to quit the simulation before it is finished. When the simulation is paused, the user can continue the simulation by pressing any key, or quit the simulation by pressing 'q'.

7.8 The Traffic Load Class

The traffic load simulation measures the load of links and nodes when packets are send through the network. With this simulation it is possible to simulate two different sending methods. It can simulate one node sending a packet to another node, or one node sending a packet to all other nodes.

In the one-to-one simulation, it finds the shortest path between the nodes and updates the link- and node load for the links and nodes used. This is done with Dijkstra's algorithm in the same way, and of the same reasons as in the number of hops simulation.

In the one-to-all simulation, the simulator also finds the shortest path to all nodes from the transfer node and updates the link and node load for the links and nodes used to transfer the packet. If there is more than one shortest path from the transfer node to a receiving node, it will send the packet through the path with the lowest load. If there is more than one shortest path and more than one path with the lowest load, the algorithm must decide which path to send the packet by looking at which path all other nodes is able to use.

The constructor for the TrafficLoad class has seven input arguments, *TrafficLoad(char* Network, int StartNode, int StopNode, int NON, unsigned long NOL, int MaxNOH, int MaxNOSP)*. The first is a pointer to the network to be simulated, the second is the node to send the packets from (transfer node), the third is receiving node (if a one-to-all simulation is wanted the value is 0). The fourth input argument is the number of nodes, the fifth is the number of links, input argument six is the maximum number of hops and the last one is the maximum number of shortest paths. The last two input arguments are found in the number of hops simulation, which means that the number of hops algorithm has to be ran before the traffic load algorithm.

The traffic load algorithm will be explained by the same wheel graph as used in the number of hops section. (See figure 7.10)

The first thing to happen in the simulation is reading the network from the node and link files. The nodes are stored in a matrix *Nodes* where the input-, output- and transit load has a column each, it is reflecting the file structure. The links are again stored in a *BidirectedLinks* variable, also with columns for the load, and again the connection matrix and distance matrix is made. In this algorithm a *LinkNr* matrix is made, containing the link number used to connect the two nodes. It is shown in table 7.4.

Nodes	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	2	0	0	0	3
2	16	0	4	0	0	0	5	0	0	0
3	0	19	0	6	0	0	0	7	0	0
4	0	0	21	0	8	0	0	0	9	0
5	0	0	0	23	0	10	0	0	0	11
6	17	0	0	0	25	0	12	0	0	0
7	0	20	0	0	0	27	0	13	0	0
8	0	0	22	0	0	0	28	0	14	0
9	0	0	0	24	0	0	0	29	0	15
10	18	0	0	0	26	0	0	0	30	0

Table 7.4: LinkNr matrix showing the link number used to connect the two edges for a 10 node wheel graph.

Table 7.4 is for later use, it will save the program for a lot of computations when he or she is updating the link load.

For this simulation all the shortest paths going from the transfer node to all receiving nodes must be known. This is done by use of a 3 dimensional array *NodeMatrix*, where all information is gathered. The x-axis contains the nodes respectively, in the y-direction is the different layers, each which contains a shortest path from the receiving node to the start node. This dimension has the length of maximum number of shortest paths, which is input argument seven (it is calculated in the number of hops algorithm).

In the z-direction, the nodes are used as transit nodes for getting to the receiving node. This 3 dimensional array is shown in table 7.5 for the 10 node wheel graph, where node 1 is the transfer node, sending one packet to all other nodes.

As seen in table 7.5 node 4 and 8 has three different shortest paths. For node 4 the shortest paths are: 9-10-1, 5-10-1, 5-6-1 and 3-2-1, and for node 8 the shortest paths are: 9-10-1, 7-6-1, 7-2-1, 3-2-1. Two counter arrays are keeping track of how many paths and number of transit nodes that have been filled in the 3 dimensional array for each node, they are shown in table 7.6.

The algorithm starts by finding the nearest nodes to the start node. It is node

Entity	1	2	3	1	2	3	1	2	3	1	2	3
Node	Layer 1			Layer 2			Layer 3			Layer 4		
1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0
3	2	1	0	0	0	0	0	0	0	0	0	0
4	9	10	1	5	10	1	5	6	1	3	2	1
5	10	1	0	6	1	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0	0	0	0
7	2	1	0	6	1	0	0	0	0	0	0	0
8	9	10	1	7	6	1	7	2	1	3	2	1
9	10	1	0	0	0	0	0	0	0	0	0	0
10	1	0	0	0	0	0	0	0	0	0	0	0

Table 7.5: A 3 dimensional array (*NodeMatrix*) containing all paths from all nodes to the transfer node.

Node	*LayerCounter*	*EntityCounter*
1	0	0
2	1	1
3	1	2
4	4	3
5	2	2
6	1	1
7	2	2
8	4	3
9	1	2
10	1	1

Table 7.6: The counter arrays for *NodeMatrix*.

2, 6 and 10. It then updates the node input- and output load for these nodes and the output load for the transfer node. It also updates the link load for the links used (link 1, 2 and 3), it is here the LinkNr matrix is used to find the correct link to update.

For each of these nodes (2, 6 and 10) the algorithm finds all the nodes connected to it, as in the number of hops algorithm. But here is also all the paths with the same distance from the transfer node found. For node 2 it will update the path in *NodeMatrix* for node 3 and 7 to go through node 2 to node 1. Node 6 is updating node 5 and 7 and node 10 is updating *NodeMatrix* for node 5 and 9.

Then the node with the shortest distance is found to be node 3. It will then go trough all its layers in *NodeMatrix* to find the path with the smallest load. There is only one shortest path to node 3, so it will use that and then update the load at the links and transit nodes used to get there.

Only nodes with one shortest path will receive a packet at this first iteration. If there is more than one shortest path, all paths to all other nodes must be taken into account. Therefore, an array *NodesWithMoreThanOneShortestPath* contains all the nodes with more than one shortest path.

When the load for all paths for all nodes with only one shortest path is updated (node 2,3,6,9,10 see *LayerCounter* in table 7.6), a look at which paths the rest of the nodes can use must be performed.

An array *NodeUse* is made to count how many times each node can be used as a transit node when sending a packet to all nodes with more than one shortest path. All nodes in *NodesWithMoreThanOneShortestPath* finds all its possible paths and counts up *NodeUse* for all nodes used in all paths. This is done to decide which path should be used to send to the nodes. In the 10 node wheel example, *NodeUse* will look as shown in table 7.7.

Now a decision about which path must be used to send a packet to the nodes in *NodesWithMoreThanOneShortestPath* is made. Starting with the nodes with the lowest distance to the transfer node, which is node 5 and 7. There are two different paths to choose between to get to node 5. This is 1-10-5 and 1-6-5. The algorithm finds the node with the highest load for each layer, and then chooses the layer with the lowest highest load to update. If two paths have the same lowest load the path with the node with the lowest value in *NodeUse* is chosen. This path might be the path which can use fewest nodes, and therefore the most optimal. When the path is found is the node- and link loads for all the nodes and links in the path are updated.

When a packet is sent to a node, all paths to that node will be subtracted in *NodeUse*, because a packet to the node is sent, and therefore it should not be taken into consideration when the algorithm finds the paths to the nodes that not have been updated yet.

Node	NodeUse
1	0
2	4
3	2
4	0
5	2
6	4
7	2
8	0
9	2
10	4

Table 7.7: The *NodeUse* array is showing how many times all nodes can be used as transit nodes for the nodes with more than one shortest path (node 4, 5, 7 and 8).

The two final nodes to be updated are nodes 4 and 8, since they are most far away from the transfer node. The paths to those nodes are found in the same way as the paths to node 5 and 7.

This was all about how to simulate one-to-all traffic, if it is wanted to simulate sending traffic from one node to another, it is done in the same manner, just then there is no need for a 3 dimensional array, because it is not necessary to find more than one path.

At the end of the algorithm the results are found as in the number of hops algorithm, and the results are saved to the .res file for the simulated network, and the loads of the links and nodes is written to the .lnk and .nod files.

During this simulation the status bar is updated every second, and the user can pause or quit the simulation as in the number of hops simulation.

7.9 The Status Bar Class

The status bar is implemented to give the user an idea of the time needed for a specific started simulation. It is good for the user to see the progress in the simulation. The user must have a feeling of something is happening, instead of that the user maybe will think that something is wrong if the simulation takes 10 minutes, and nothing is happening on the screen.

The status bar is implemented as a class. All the three simulation classes, maximum flow, traffic load, and number of hops have calls to this class throughout in the algorithms. This in order to create, update and reprint the status bar to a screen.

When calling the update status bar method in the status bar, two arguments has to be sent. The first is a variable called *tot*, which is set according to the for loops specifications. A simple example of setting the *tot* variable is that if there are three for loops that counts from zero to a variable NumberOfNodes, *tot* is set to 3 * NumberOfNodes. This means that this *tot* has to be calculated in the specific simulation classes and sent as arguments to the status bar. The other variable the update status bar method needs as argument is a counter, *cnt*, in which the specific algorithm in the simulations have to count. In the example mentioned above, this counter has to be incremented in the for loops and have to be equal to *tot* when the for loops is finished. In other words, the first call to the status bar in the beginning of the algorithms the *cnt* is set to 0, and it should be counted up to the same number as *tot* in the end of the algorithm. In the number of hops class a status bar object is made in the start of the algorithm. And the status bar method update is called for the first time with $tot = 4 \cdot NumberOfNodes + NumberOfBidirectedLinks + (NumberOfNodes \cdot NumberOfNodes)$ and $cnt = 0$. Then a status bar should appear at the screen for the first time. The status bar is then repeatly updated throughout the algorithm with *cnt* incremented such that the status bar shows the progress. When $cnt = tot$ the status bar shows 100% and the simulation is finished.

7.10 The Plotter

In order to visualize the results, a plotter tool is also developed and integrated with the simulation platform. From the simulator menu, the user can plot results of saved networks and compare the difference between them. Moreover, these plots can be saved as images in PNG format. Since the plotter tool is based on GUI, Java programming language is used. The reason to choose the Java technology is that it has developed GUI libraries such as Abstract Windows Toolkit (AWT) and Swing etc. These libraries support to have a faster development of GUIs, compared to C++, for a non advance programmer.

The plotter consists of three classes. The detail of classes and their important methods is given in the following sections. For the detailed information of all the class methods, a documentation has been generated using Java Doc utility and can be found in the enclosed CDROM[2].

7.10.1 PlotApp Class

The PlotApp class is the main class for the plotter and yields the main method to run the application. In this class an object for the window's frame is created and also the window's size and visibility is specified.

[2]CDROM/Plotterdocumentation

7.10.2 SimPlot Class

The SimPlot class extends the JFrame which is a Java swing component and creates the window. The SimPlot class also creates other components such as menu bar, buttons, combo selection box and canvas to draw the plots. The constructor*SimPlot()* initializes all the components. For event handling an action listener *addActionListener()* is added to each component, which is used to perform an action. Furthermore, *actionPerformed()* method triggers a certain action.

7.10.3 PlotCanvas Class

The PlotCanvas class provides most of the functionality to the plotter. This class extends the Canvas component which is added in SimPlot. A Canvas component represents a rectangular area of the screen onto which the application draw something useful. In order to create custom plot drawings, the *paint()* method is called to perform custom graphics on canvas. The method *readFile()* reads the data saved by simulator from the plotdata.dat file. Other methods like
setAverageNumberOfHops(), *setMaximumNumberOfHops()* and similar for rest of the parameters call the *paint()* method to draw the plot for its particular type.

7.11 Summary

The implementation of the driver program and the different classes in the network simulation platform is described in this chapter. How the classes are linked together, and which classes that initialize objects of other classes, and runs the methods for these objects is described in the text. A better picture of this interaction is given in figure 6.1 on page 31.

During implementation, the Network class is given more functionality than first planned. The final class became therefore relative big with all its methods and data members. To get a quick overview is difficult. The class should therefore maybe be splitted into two smaller, more readable classes. All network editing functionality could for instance be placed in a seperate class. However, the Network class is in this project kept as one single class.

Chapter 8

Performance Test vs. Acceptance Test

The system acceptance test is a way to evaluate the overall requirements for the product, that are found in section 4.2 on page 17. The most important reason for testing the software of the simulation platform is to detect any possible software error or fault, and to improve the overall quality of the software. In this chapter the simulation platform's function's will be tested. In general the overall software will be tested, this will also include the menu system and the status bar, which are the interaction or interface between the user and the simulation platform.

8.1 Software Testing Strategies

The testing strategies used in this project is to perform tests of single modules and then performing integration tests. There will be made a test plan for the "product" which has been developed in order to know, how to test the whole program and/or modules of it. The test of the simulation platform is meant to verify the status of the product to conclude how far or close the product is e.g. to be sent out on the marked to be used by customers in the "real world." The software will be tested using structural "white box testing" and functional "black box testing." Many of the tests which will be performed will occur simultaneously.

8.2 Performance Test

The following sections are the preparation of tests. There will in this project be performed the integration test all-at-once. This method provides a useful solution for simple integration problems, involving a small program possibly using a few previously tested modules. When all modules have been integrated and tested a system test will be performed. The system test or performance test will be to load the simulation program and perform stress tests in order to try determining the failure point of a system under extreme pressure by simulating as many nodes as the program can handle. This will depend on the computer's CPU-power and RAM. A drawback of performing a stress test is that, this test can easily confirm that the system can handle heavy loads, but it will maybe not so easily determine if the system is producing the correct information! This will be discussed at the end of this chapter.

8.3 Test Plan

This section will provide a plan of how the tests will proceed. The following objects which are presented will be tested, these objects are the use cases from appendix B on page 108. Each of the user cases functions will now be specified.

8.3.1 Test Case Specification and Procedure

Uses Cases are very important in the test of the software and will in this section be described how they should be tested. In the description, the expected result will be included. The following use cases will be described and tested:

1 The user opens a project.

2 The user closes the project.

3 The user specifies a new network in the project.

4 The user adds an existing network to the project.

5 The user opens an existing network in the project.

6 The user removes an existing network from the project.

7 The user makes changes in the network.

8 The user will auto generate a network topology.

9 The user close the active network in the project.

10 The user selects wanted simulation parameters and starts simulation.

11 The user stops the simulation.

12 The platform plots the simulation results from different networks.

13 The platform shows a status bar during simulation.

Each of the use cases will be tested in refer to the requirement specification.

Test Case 1 - the User Opens a Project

The first test case "the user opens a project" will now be described how to test. Firstly, the user have to create a new project and then enter the project's name. Hereafter, the user must save the project and return to main menu. Then a project has been created and now it is possible to open it. To open a project, the choice number 2 must be chosen. There will now be a list of project to open and there will be the one which has created. For example the project is called TestCase1, this name must be entered in order to the if a project can be open.

The above test case has been tested and work correctly. Therefore, this function in the program has been concluded to work successful.

Test Case 2 - the User Close the Project

The next test case is "the user close the project" will be tested in the way that, the user choose the menu number 7 in the project menu. This is called "Save the project and return to Main menu." This will menu function will save and close the project.

Test Case 3 - the User Specifies a New Network in the Project

In order to specify a network in the project, project must be given, i.e., a project must is created or opened. Then the user can specify a new network configuration by choosing the project menu number 1, called "Create a new network." Thus a network will be specified in a project, the user must enter a name for the network. Hereafter the number of nodes must be specified and their capacities. The capacities can either be the same or the individual node's capacity can be entered. Now the user must select the network topology which can be one of the following configurations:

1 Simple ring structure.

2 Fully connected.

3 Wheel structure.

4 Double ring structure.

5 N2Rpq structure.

6 Random generation.

7 Manually.

The user must. after a selected network configuration select capacities for the links. Either the same capacity can be assigned to each link or individual capacities can be entered. Now the user has specified a network in a project. When leaving the network menu the user chooses "close and save network" and then it will return to the project menu.

Test Case 4 - the User Adds an Existing Network to the Project

In the project menu. the user can add en existing network to a project, by choosing the project menu number 4 "Add a network to project." Then network which are already included in the project will be listed and the ones that can be selected are listed. The user must then enter the network name in order to include it in the project.

Test Case 5 - the User Opens an Existing Network in the Project

The user must open an existing network in the project menu, by choosing the number 2 in the project menu, called "Open a network." Then a list of the available networks will be presented. A network is added by entering the name of the network.

Test Case 6 - the User Removes an Existing Network from the Project

To remove an exiting network from the project, the user chose the menu number 5 "Remove a network from project." A list of the networks in the project will be presented and the user can then choose which network(s) to remove.

Test Case 7 - the User Makes Changes in the Network

In order to reconfigure a network, the user must be in the network menu. Here are the following possibilities which can be changed:

- - Add node.
- - Add link.
- - Change node capacity.
- - Change link capacity.
- - Delete node.
- - Delete link.
- - List network data.

Test Case 8 - the User will Auto Generate a Network Topology

To generate a network topology automatically, the user must first choose "create a new network," see test case 3. Thus, it is possible to chose to generate a random network, where the links are randomly generated between the nodes. The user has to set the highest allowed degree of the nodes.

Test Case 9 - the User Close the Active Network in the Project

To close an active network in a project, the user must select number 10 in the network menu, which is called "Close and save network." This will now return the user to the project menu, then a network is saved.

Test Case 10 - the User Selects Wanted Simulation Parameters

In order to test which simulations parameters the user wants to simulate a network configurations must be presented. Then in the network menu, the user must select the number 8 option "Simulate." Here the following options will be presented and each of them must be tested.

- Number of hops.
- Traffic load.
- Maximum flow.

Before the traffic load can be tested, the number of hops must be simulated.

Test Case 11 - the User Stops the Simulation

When a simulation is in progress, a simulation can stop or pause by pressing "p" on the keyboard. In order to continue, the user presses any key and then simulation will run from where it was stopped. When a simulation is running, the user can choose to interrupt the program. Thus, to quit the program this is done by pressing "q" on the keyboard and the simulation will be stopped.

Test Case 12 - the Platform Plots the Simulation Results from Different Networks

To test this use case, the user must be in the project menu. Here, the user must select option 6 "Plot networks in project." The plot is a Java program and will take some time to open. The plot window must be closed in order to return to the menu.

Test Case 13 - the Platform shows a Status Bar During Simulation

This test case will be tested by running a simulation and then at the same time seeing the status bar is updating during the simulating. This can only be tested by inspecting that the time vs. percentage is correct.

8.3.2 Stress Test

In this section, a stress test on the simulation program will be described and performed. The purpose of the stress test is to test if the tool fulfills the demand about simulation up to 1500 nodes.

The tool is able to simulate all the different 1500 nodes networks which are have tested. The simulation time for the number of hops and traffic load simulations for the most demanding network structure (double ring) is shown in table 8.1 for 300, 600, 900, 1200 and 1500 nodes to see how the simulation time evolves over number of nodes. The different network topologies are also compared with max load on 1500 nodes in table 8.1.

The 3 dimensional array, *NodeMatrix* in the traffic load algorithm, can become very large. It is initialized to have the dimensions $n \cdot MaxNOSP \cdot MaxNOH$. If n is large ≈ 1500 the array can have up to $5 \cdot 10^8$ element and each element has a size of an integer (4 bytes) we need $2GB$ of RAM, which is very much. But the problem is not that big for the most of the networks to be simulated with this tool. In table 8.1 it is also shown how much RAM is needed to simulate some common large-scale networks.

Topology	*NON*	NumberOfHops Simulation	TrafficLoad Simulation	MB RAM needed
Double Ring	300	0:00 min	0:01 min	3.3
Double Ring	600	0:05 min	0:01 min	26.1
Double Ring	900	0:14 min	0:03 min	87.7
Double Ring	1200	0:31 min	2:26 min	207
Double Ring	1500	1:01 min	7:21 min	1618
Ring	1500	0:58 min	0:06 min	8.6
Full Connected	1500	2:42 min	2:22 min	0.1
Wheel	1500	0:59 min	2:49 min	805
N2Rpq, p=750, q=50	1500	1:09 min	0:04 min	10.1
Random	1500	1:18 min	0:04 min	1.5

Table 8.1: The simulation time for the number of hops and the traffic load simulation for different network topologies. At the right the amount of RAM need to make the traffic load simulation is shown.

The simulation times for a double ring structure are plotted in figure 8.1.

The *NodeMatrix* array is not the only array to use RAM, but it is for sure this

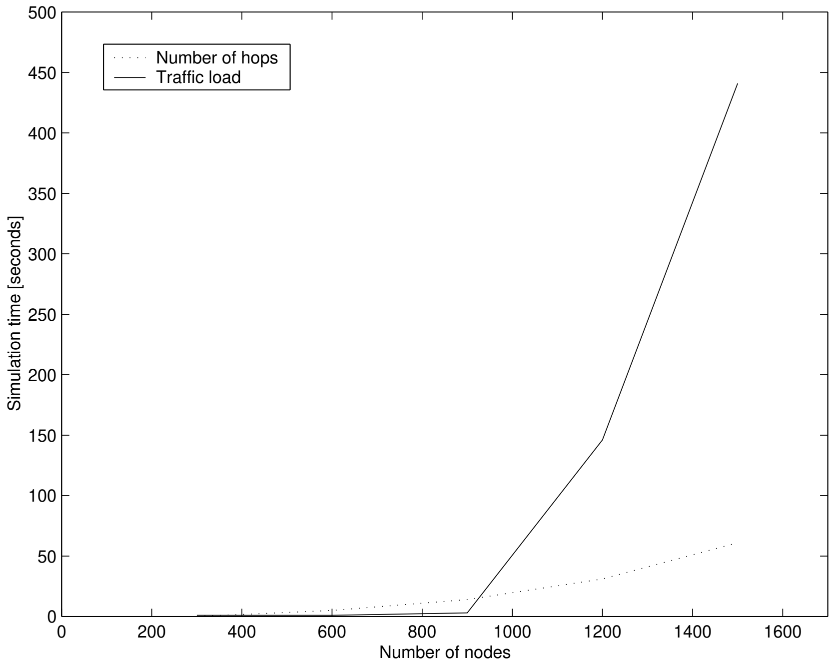


Figure 8.1: Plot of simulation times for a double ring structure.

array that can cause memory problems. Not any 2-dimensional array will use more than 20 MB no matter what network below 1500 nodes which is wanted to simulate.

It can be concluded that the simulation program can handle 1500 nodes. It is possible to simulate networks with more than 1500 nodes, but it is what the simulation program can guarantee to simulate. The amount of nodes is simulated successful when status bar shows 100%.

8.4 Performance Test Results and Discussion

This section will discuss the results from the user-test for each of the test cases in the previous section. The test cases which has been performed as described worked correctly. Though the test case 7 in section 8.3.1 has been implemented different than described in the use case description, which in found is appendix B on page 108.

8.5 Acceptance Test

Now the user acceptance test will be performed. The goal with the acceptance test is to check the system against the "Requirements." Normally it is the customer who will test the system, not the developer. The customer in this project is the project group. Therefore, in order to perform the acceptance test, the customer or user of the system will be the project group. Unfortunately, at this point there will usually not be a chance of changing big errors.

The acceptance test has been performed by using the program in general.

8.6 Summary

The tests has been carried out in an iterative process, a module has been programmed, then tested to detect faults, which then has been corrected by reprogramming the module. The performance test showed that the individual user case work as defined.

Part III

Simulations and Analysis of Networks

Chapter 9

Network Description and Simulation

This chapter describes different network topologies. These are described and then simulated using the platform. An analysis is performed according to some of the parameters defined in appendix A.1. The topologies described are a simple ring, a fully connected, a Petersen, a double ring and a wheel structure. The ring and the fully connected structure are compared in the first section, since these structures are a sort of worst and best case of node connections. The graphs are analysed, and the platform is used for simulating the graphs in order to compare the platform simulation results against the analysis.

9.1 Ring Structure vs. Fully Meshed Structure

Figure 9.1 illustrates a 10 node ring, and a 10 node fully connected graph.

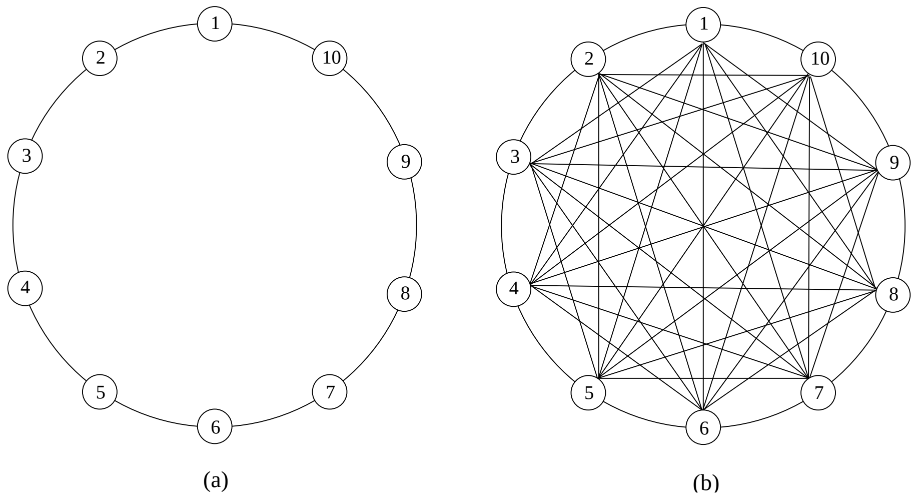


Figure 9.1: A 10 node ring and a fully meshed structure.

The ring structure uses 10 links to connect 10 nodes. The fully meshed structure used 45 links, which can be calculated by equation 2.1 on page 7. In the platform a ring structure is generated by choosing *make a ring structure* on a 10 node network. The platform is connecting the links as in figure 9.1 (a). The fully meshed structure as in figure 9.1 (b) is generated in the platform by choosing *make a full mesh structure* on a 10 node network. Looking at the platform's link file it can be seen that the platform is connecting the links correct.

Both of the graphs are simulated in the platform for number of hops, maximum flow, and traffic load by letting one node sending packets to all other nodes.

Since both graphs are symmetric, it does not matter which node is chosen as start and stop node in the maximum flow simulation. The capacity is for simplicity chosen to be 1 in each of the links. Maximum flow for the ring graph is then 2, it is a two degree graph. Maximum flow for the full meshed network becomes 9. All nodes are connected to all other nodes, then it is obvious that the maximum flow becomes 9. The maximum flow simulation works properly on these two graphs.

$$AvgNumOfHops = \frac{SumNumOfHops}{NumberOfNodes \cdot (NumberOfNodes - 1)} \tag{9.1}$$

The sum of number of hops in the ring graph becomes 250. A look at figure 9.1 to find number of hops from one node to all other nodes gives a number of 25. The sum of number of hops is then this number is multiplied by 10 nodes since the ring graph is symmetric. Maximum number of hops is 5, to send a packet from node 1 to node 6 in figure 9.1, 5 hops is needed. The average number of hops becomes 2.778, and can be calculated manually by equation 9.1, giving the same result. For the fully meshed structure the number of hops simulation results can be summarized as this:

- Sum number of hops: 90.
- Maximum number of hops: 1.
- Average number of hops: 1.

The traffic load simulation of the ring graph is performed by letting node 1 send out packets to all other nodes. Figure 9.2 shows an example of how the node one can send out one packet to all other nodes.

From figure 9.2 the traffic load parameters can be found. The following shows the traffic load parameters for a ring structure versus a fully meshed structure. The results for the ring come first, and then for fully meshed.

- Maximum input load for nodes: 5 versus 1.
- Maximum output load for nodes: 9 for both.

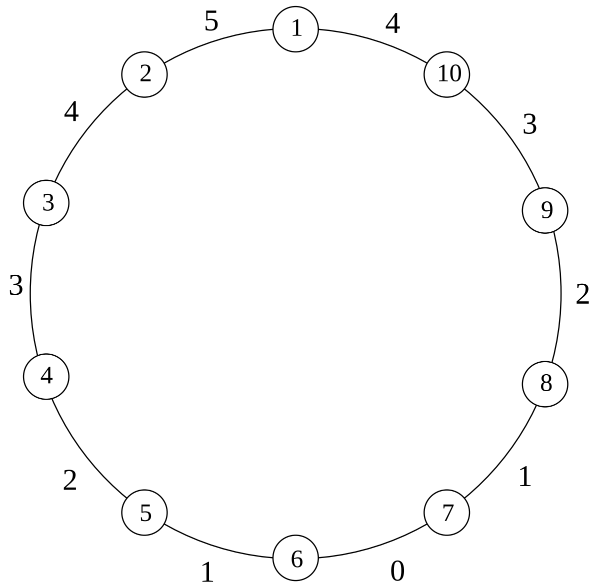


Figure 9.2: Traffic load route in a ring structure.

- Maximum transit load for nodes: 4 versus 0.
- Sum input load for nodes: 25 versus 9.
- Sum output load for nodes: 18 for both.
- Sum transit load for nodes: 16 versus 0.
- Average input load for nodes: 2.5 versus 0.9.
- Average output load for nodes: 1.8 versus 0.9.
- Average transit load for nodes: 1.6 versus 0.
- Maximum link load: 5 versus 1.
- Sum link load: 25 versus 9.
- Average link load: 2.5 versus 0.2.
- Links not used: 1 versus 36.

The traffic load simulation results above can be seen as worst and best case for a 10 node network. The following section is describing and simulating network structures, which is in the middle of the extreme cases discussed in this section.

9.2 The Petersen Graph

This section introduces the Petersen graph, and the description is mostly based on [13]. The Petersen graph has fascinated many graph theorists over the years because of its appearance as a counterexample in many places.

The Petersen graph was named because of its appearance in 1898 in a paper by J. Petersen. One of the problem Petersen researched is: Let λ be a 2 edge connected graph where every vertex is connected with three edges. Is it possible to color the edges in λ with 3 colors, and then have two edges, which is incident with the same vertex to always get different color? Petersen claimed the answer no to this question, and proved it with the Petersen graph illustrated in figure 9.3.

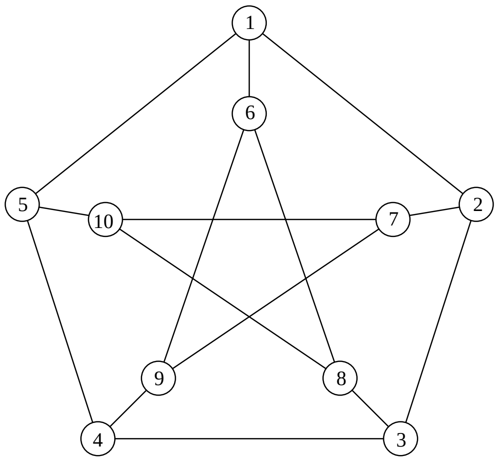


Figure 9.3: The Petersen graph

The Petersen graph can be illustrated in several representations, possessing 10 nodes, all of whose nodes have degree three, two examples are given in figure 9.4.

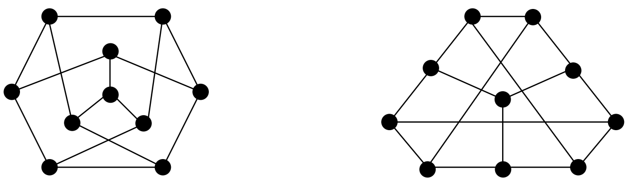


Figure 9.4: Different representations of the Petersen graph.

The Petersen graph is a non planar graph, which means that it is possible to draw this graph with two crossing edges on a paper. The Petersen graph is also both symmetric and cubic. The girth is defined as the length of the shortest graph cycle, and become for this graph 5. The graph diameter is 2. A 10 vertices Petersen graph is the smallest Hypohamiltonian graph. A Hypohamilonian graph is a graph which is not a Hamiltonian, but whose vertex deleted subgraphs are all Hamiltonian. If a cycle has the properties that a walk through the graph (closed loop), visits each node only once, it is a Hamiltonian cycle. A graph is Hamiltonian if it has a Hamiltonian cycle.

Actually the Petersen graph with 10 nodes is a N2Rpq graph [14] with N = 10, p = 5 and q = 2. In an N2Rpq graph, N is the number of nodes, and p the number of nodes in each ring (2 rings). Q is a variable which tells the platform how many nodes a link jump in the inner ring before connecting to a node in the inner ring of a N2Rpq graph. In the simulation platform a Petersen structure can be generated by the *Create a N2Rpq structure* in a network created with 10 nodes. Then the platform will ask for the q. A choice of 2 here will make a network equal to figure 9.3. To test that the simulation platform actually is making a Petersen graph, the source and sink node from the link file is listed in table 9.1.

Link Number	Source Node	Sink Node
1	1	2
2	2	3
3	3	4
4	4	5
5	7	10
6	6	8
7	7	9
8	8	10
9	9	6
10	1	5
11	1	6
12	2	7
13	3	8
14	4	9
15	5	10

Table 9.1: List of Petersen graph generated by the platform.

Comparing table 9.1 with figure 9.3, it can be seen that the platform has written all links to the link file in an appropriate way.

The number of hops in the alternative paths is also a property. If this number is relative low, the functionality of a network can be good, even if some links or nodes breaks down. For the graphs described in this section, the three shortest routes from a given start node to the others will be shown. This will not be repeated where number of hops in the three paths is equal.

Figure 9.5 (a) depicts the situation where the minimum distance in the Petersen graph is one hop from a start node to a another node. The number of hops in the three paths are 1, 4, and 4. The alternative paths will have the same number of hops for any other node where the minimum distance is one. The number of hops is 2, 3, and 3 for the minimum distance two-nodes, this is shown in figure 9.5 (b).

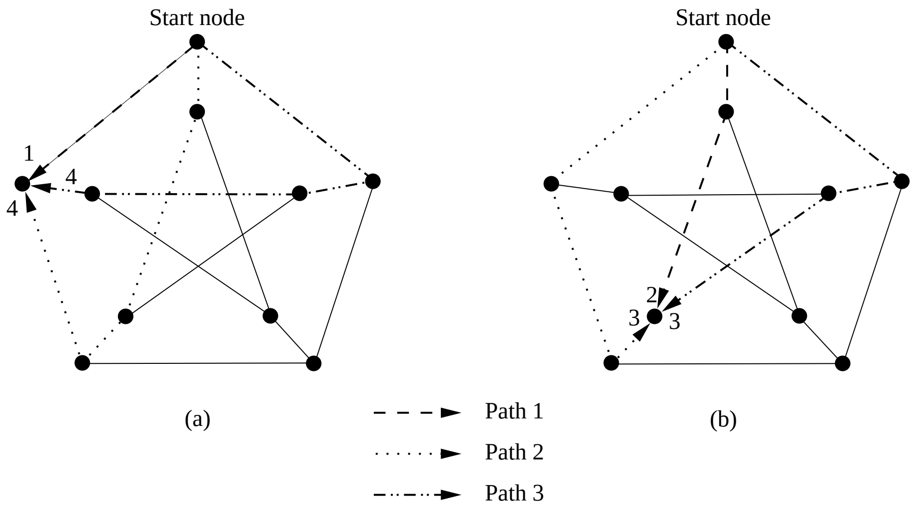


Figure 9.5: Three independent paths in the Petersen graph.

9.2.1 Number of Hops in the Petersen Graph

In figure 9.6 the number of hops is shown for the Petersen graph. The sum of the number of hops for a Petersen graph is 15. For a 10 node double ring and a wheel graph the sum of number of hops is 17.

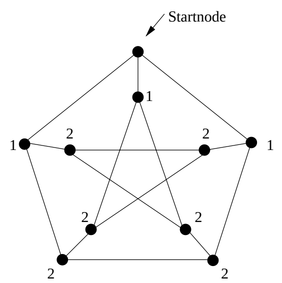


Figure 9.6: Counting number of hops to each node in the Petersen graph.

The Number of hops simulation in the platform simulates the sum of number of hops, the maximum number of hops and the average number of hops. The sum of number of hops is 150 for a 10 node Petersen graph, because the number of 15 found in figure 9.6 has to be multiplied with 10 nodes to find the sum of hops in the network. This can be done since the Petersen graph is symmetric. The maximum number of hops is 2 in a Petersen graph. If a packet is sent from a random node to another random node, the maximum number of hops will never exceed 2. The average number of hops can be calculated by equation 9.1.

In the Petersen graph, the average number of hops is 1.667. The simulation platform gives the following results when simulating number of hops:

- Sum of number of hops: 150.
- Maximum number of hops: 2.
- Average number of hops: 1.667.

The number of hops simulation of the Petersen graph is giving correct results concerning the number of hops parameter.

9.2.2 Maximum Flow in a Petersen Graph

If a Petersen graph has equal capacities in all of the links, the maximum flow from one node to another becomes three times the capacities since this graph has three independent paths. This is true independent of which nodes are set as start node and stop node for the maximum flow calculation. In the platform the capacity of the links is all set to three. Running a maximum flow simulation on this network gives a maximum flow of nine, independent of the choice of start and stop node to the algorithm. The conclusion is that the maximum flow algorithm works as expected on the Petersen graph.

9.2.3 Traffic Load in a Petersen Graph

The traffic load is calculated by pointing out a start node and sending out packets to all other nodes. Figure 9.7 shows how the traffic will flow if node 1 is sending a packet to all other nodes following the shortest path.

From figure 9.7 it can be seen that maximum input load to a node is 3, maximum output load from a node is 1 and maximum transit load is 2. The output load will always be maximum 9, because the packets all nodes is receiving the nodes has to send to another node another place in the network (outside this report‘s scope). The sum of input load to the nodes is 15, and the sum of the output load from the nodes is 18. The sum of the transit load in the nodes becomes 6. Average values is also calculated. The average input, output, and transit load for the nodes is simply the sum of these nodes input, output, and transit load divided by the number of nodes. In the Petersen graph is the average input load to the nodes 1.5, and the average output load 1.8, and average

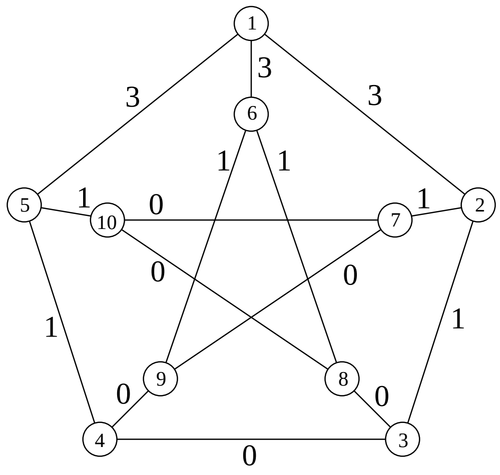


Figure 9.7: The traffic load in a Petersen graph.

transit load 0.6. The maximum link load in figure 9.7 is 3, and the sum of the link load is all the link loads added, and becomes 15. Since there is 15 nodes, the average link load is 1 for the Petersen graph.

When running a traffic load simulation on the platform with the Petersen graph structure, the same results appear. Therefore, the traffic load simulation on a Petersen graph is working.

9.3 The Double Ring Graph

To improve the redundancy in a single ring structure, a second ring can be added to the main ring. The result is a double ring graph, as shown in figure 9.8.

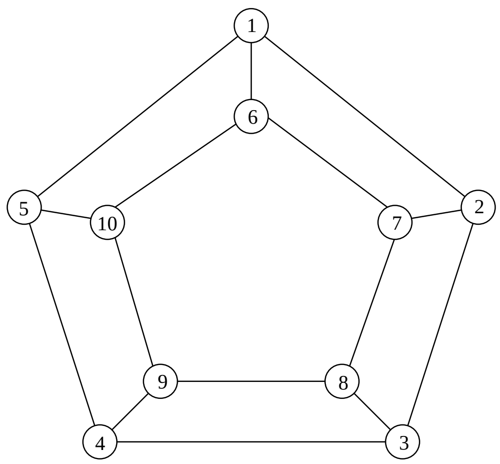


Figure 9.8: The double ring graph.

For the double ring graph, there are two situations where the minimum distance is one hop, figure 9.9 (a) and (b). The number of hops for these is 1, 3 and, 4, and 1, 3, and 3, respectively. Where the shortest path is two hops, there is also two possibilities: 2, 2, and 5, figure 9.9 (c), and 2, 3, and 4 figure 9.9 (d). The last one is shown in figure 9.9 (e), here the minimum distance is 3, and the alternative paths have 3, and 4 hops.

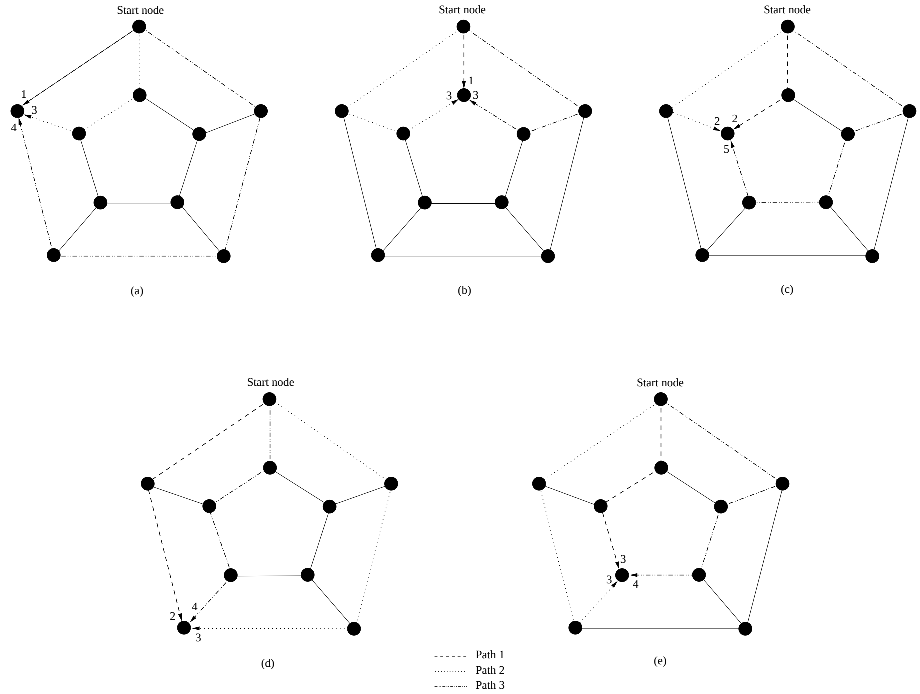


Figure 9.9: Three independent paths in the double ring graph

The double ring graph shown in figure 9.8 can be generated in the platform by creating a network with 10 nodes and choose a *double ring structure* to connect the links. The link file generated by the platform is checked against figure 9.8, and the platform connects the links in a double ring structure. The maximum flow simulation will not be explained in this section, because it gives the same results as in section 9.2. The double ring is as the Petersen graph a symmetric degree 3 graph, and the maximum flow becomes the same.

9.3.1 Number of Hops in a Double Ring

As mentioned in section 9.2, the minimum sum number of hops from a start node to any other node in the wheel graph is 17. This is calculated by adding the numbers in figure 9.10.

The sum of number of hops in this graph is 170, it is 17 multiplied by 10 (the number of nodes). Keep in mind that this can only be done with symmetric

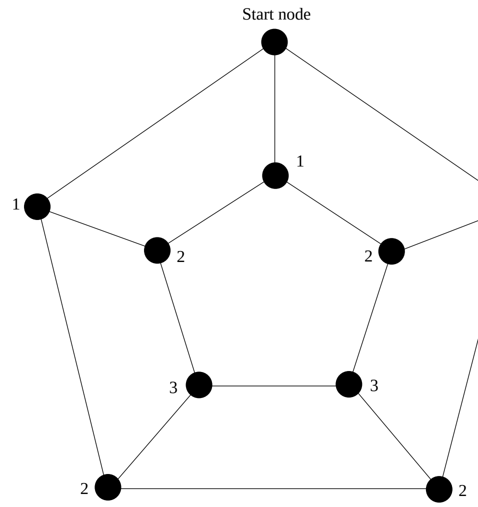


Figure 9.10: Hops in a double ring graph.

graphs. By looking at figure 9.10 there can not be found no more than 3 hops as maximum number of hops. The average number of hops is calculated by equation 9.1 to be 1.889. The simulation platform gives as expected the following results when simulating number of hops:

- Sum number of hops: 170.
- Maximum number of hops: 3.
- Average number of hops: 1.889.

9.3.2 Traffic Load in a Double Ring

As in the section 9.2.3 traffic load is calculated by sending packets from one node to all other nodes. The double ring graph has more than one shortest path from one node to another, therefore the traffic flow can be executed in not only one embedding. Figure 9.11 shows an example of how the traffic can flow when sending packets from node 1 to all other nodes.

It can be seen in figure 9.11 that maximum input load to a node is 3, maximum output load from a node is 1 and maximum transit load is 2. The sum of input load to the nodes is 17, and sum of the output load from the nodes is 9. The sum of the transit load in the nodes becomes 8. Some average values can also be calculated. The average input, output, and transit load for the nodes is simply the sum of these nodes input, output, and transit load divided by the number of nodes. The average values are an average input load to the nodes of 1.7, an average output load of 1.8, and an average transit load of 0.8. The maximum link load in figure 9.7 is 3. The sum of the link load is all the link

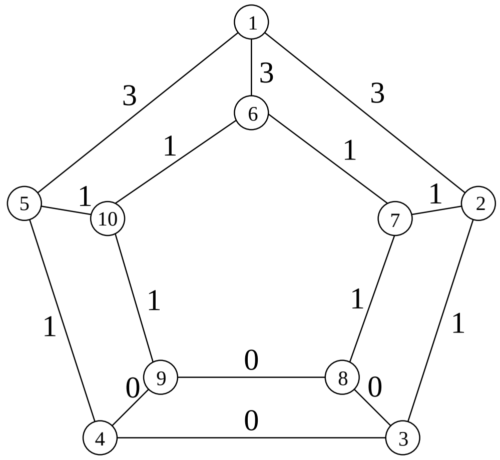


Figure 9.11: An example of the traffic flow route in a double ring graph.

loads added, and it becomes 17. Since there is 10 nodes the average link load becomes 1.133 for the double ring graph. The number of links not used is 4.

When running a traffic load simulation on this graph, the result becomes a little bit different. The results the platform is gives are:

- Maximum input load for nodes: 3.
- Maximum output load for nodes: 9.
- Maximum transit load for nodes: 2.
- Sum input load for nodes: 17.
- Sum output load for nodes: 18.
- Sum transit load for nodes: 8.
- Average input load for nodes: 1.7.
- Average output load for nodes: 1.8.
- Average transit load for nodes: 0.8.
- Maximum link load for nodes: 3.
- Sum link load: 17.
- Average link load: 1.1333.
- Links not used: 6.

The only parameter which differ from the manual traffic load analysis is actually the parameter of links not used. The reason for this differ is that the platform choose another shortest path when simulating the traffic load. The path, which the platform choose is shown in figure 9.12.

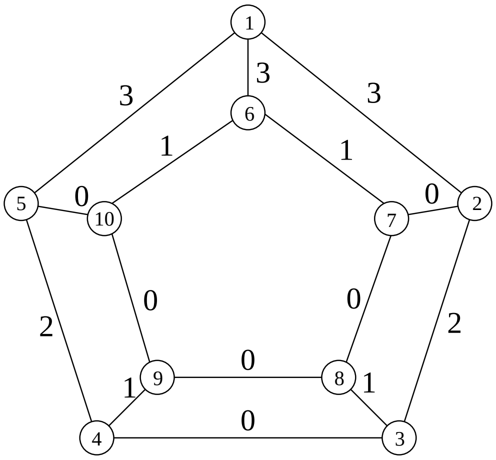


Figure 9.12: An alternative traffic flow route in a double ring graph.

It can be concluded that the platform is performing a traffic load simulation as good as the manual traffic load analysis performed above.

9.4 Wheel Graph (Möbius Ring)

The wheel graph with 10 nodes is shown in figure 9.13.

If two links are switched in the double ring graph in figure 9.8, the same graph as in figure 9.13 actually appears (see figure 9.14).

Where the shortest path is one hop, the two alternative paths have both five hops for the wheel graph, figure 9.15 (a). There are two situations where the minimum distance is two, the three paths have 2, 4, and 4 hops, figure 9.15 (b), and 2, 2, and 4 hops, figure 9.15 (c). In the last possibility, figure 9.15 (d), the number of hops is equal in all three paths, namely 3.

A wheel graph as in figure 9.13 can be generated by the platform. Choose the links to be coupled in a *wheel structure* when choosing the link connection. The link file is checked against figure 9.13, and the platform is connecting the links in a correct way.

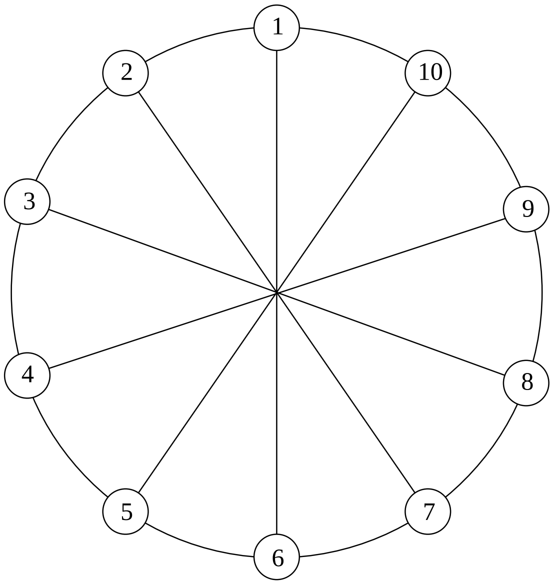


Figure 9.13: The wheel graph (Möbius graph).

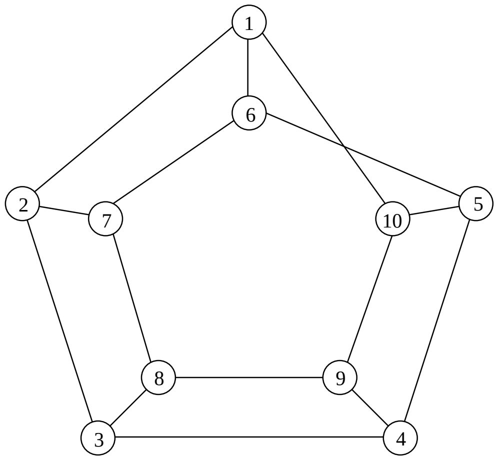


Figure 9.14: The equivalent wheel graph.

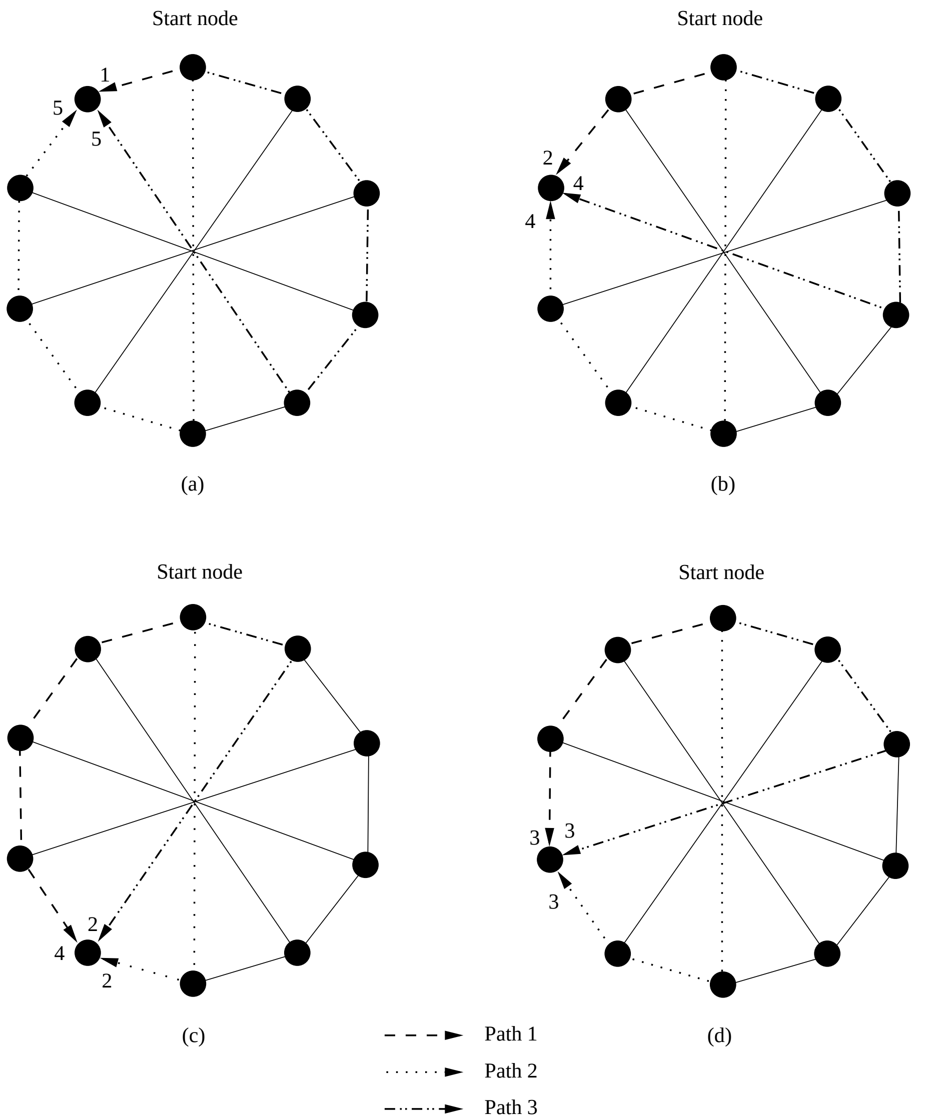


Figure 9.15: Three independent paths in the wheel graph.

9.4.1 Number of Hops in a Wheel Graph

The minimum number of hops from a start node to any other node in the graph is shown in figure 9.16, the sum of these distances is 17.

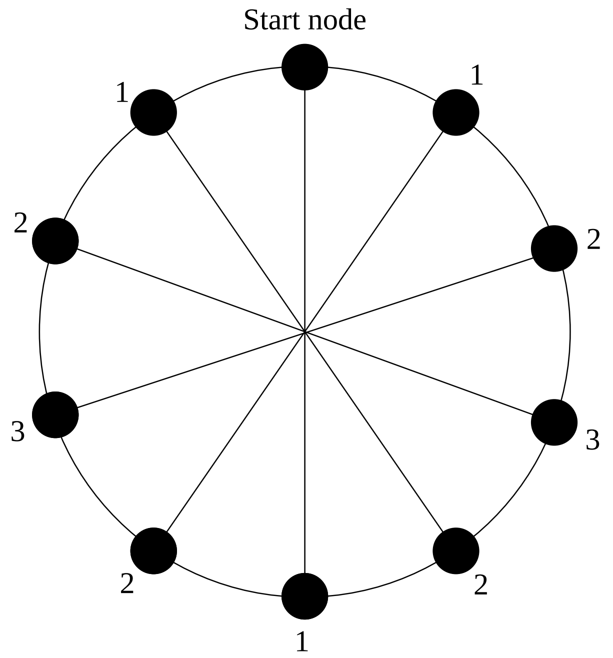


Figure 9.16: Hops in the wheel graph.

The sum of number of hops in this graph is 170, it is 17 multiplied by 10 (the number of nodes). Again, it is important to have in mind that this can be done because the wheel graph is a symmetric graph. By looking at figure 9.16, there can not be found more than 3 hops as maximum number of hops. The average number of hops is calculated by equation 9.1 to be 1.889. The simulation platform gives as expected the following results while simulating number of hops:

- Sum of number of hops: 170.
- Maximum number of hops: 3.
- Average number of hops: 1.889.

The number of hops simulation results are similar to the simulation for the double ring.

9.4.2 Traffic Load in a Wheel Graph

The traffic in a wheel graph while node 1 is sending packets to all other nodes can be routed as in figure 9.17.

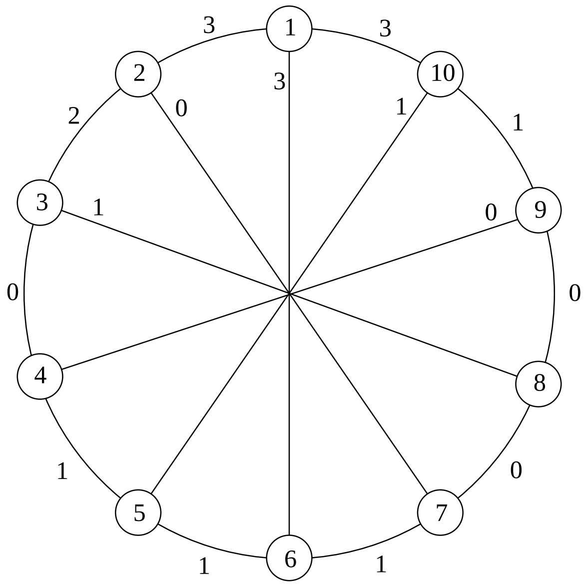


Figure 9.17: Traffic load in the wheel graph when node 1 is sending a packet to all other nodes.

When analysing the traffic load in figure 9.17, almost the same results as in the double ring traffic load analysis is calculated. The only difference is the number of links not used, which becomes 6 in figure 9.17. The platform is also routing the traffic as in figure 9.17 when simulating traffic load, and gives the results as expected, including that links not used are calculated to 5.

9.5 Comparing the Different Network Structures

The different networks described in this chapter are here compared by using the plotting tool in the platform. The comparison is done in order to get a better view of the advantages and disadvantages of the different networks.

In all the plotting figures N0 is the ring structure, N1 is the Petersen Graph, N2 is the double ring, N3 is the wheel graph and N4 is the full mesh structure discussed and simulated in this chapter.

In figure 9.18 the maximum number of hops is plotted for all the structures. The average number of hops is plotted in figure 9.19. It is worth mentioning that the Petersen graph has a low number of hops while the numbers of links is the same as in the double ring and the wheel graph. The plot of the sum of number of hops is found in appendix D on page 120.

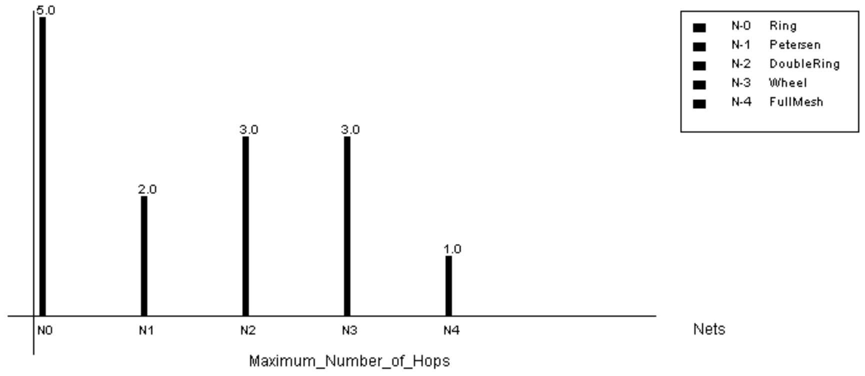


Figure 9.18: Maximum number of hops.

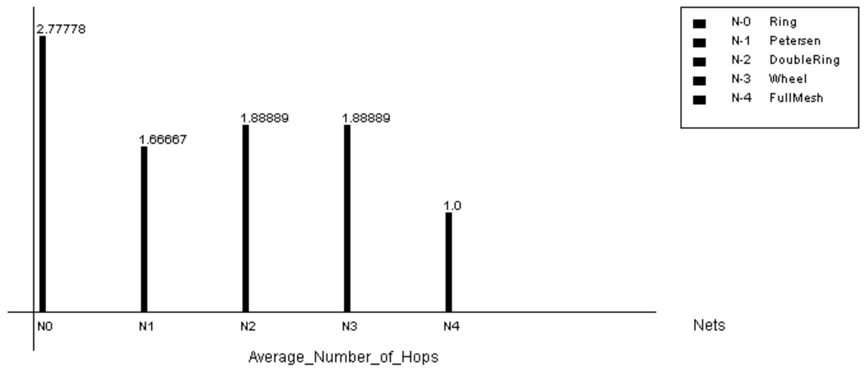


Figure 9.19: Average number of hops.

The maximum input load for the nodes in the networks is shown in figure 9.20. This figure is also showing the maximum link load, and the maximum transit load for the nodes is also equal except for the full mesh structure which becomes 0. The plots for maximum link load and maximum transit load is found in appendix D on page 120.

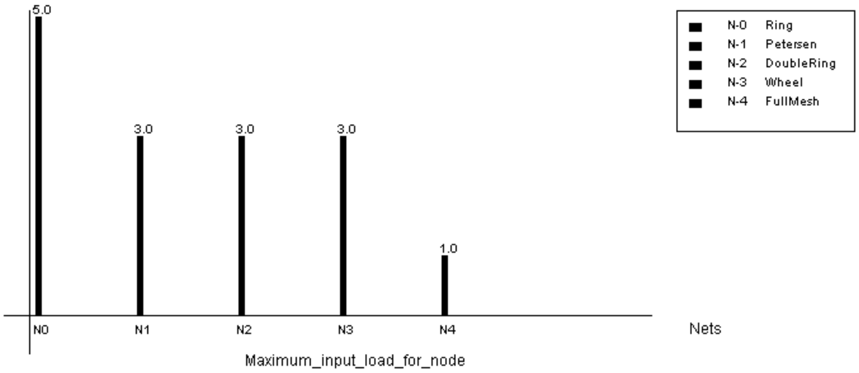


Figure 9.20: Maximum input load for nodes.

The average input load is plotted in figure 9.21. The plots is shows that the Petersen graph gives the best results except for the full mesh structure.

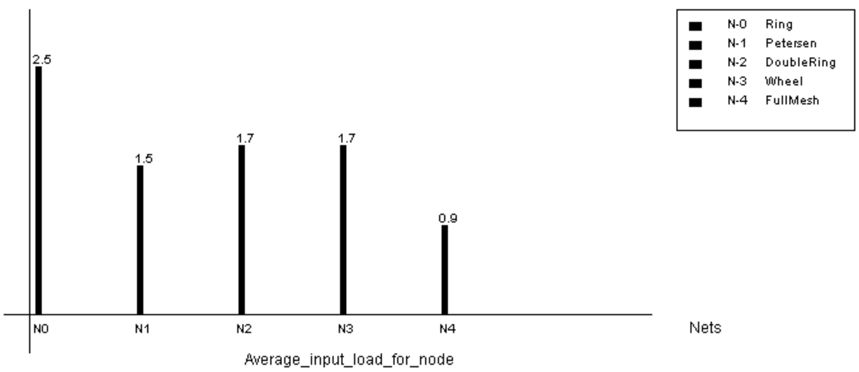


Figure 9.21: Average input load for nodes.

The plots for average link load and of links not used is found in appendix D on page 120. Figure D.6 on page 122 in appendix D shows the numbers of links not used for all the networks. For the full mesh structure this number becomes 36, which means that there is a high number of links not used. In chapter 10 on page 85 large-scale networks is simulated, and then the numbers of unused links becomes a very high number for full mesh structures. This is the main reason why a full mesh structure is rarely, or maybe never, used practically in large-scale networks.

Part IV

The Large-Scale use of the Platform

Chapter 10

Large-Scale Simulations using the Platform

Until now, only networks with a small amount of nodes have been tested in the simulation platform. According to the requirements set earlier in the report, the platform should be able to simulate large-scale networks of up to 1500 nodes. Different topologies with 1500 nodes will be simulated. The next section compares the properties of these different networks according to the simulation results. After this a case study about the 1500 nodes N2Rpq is described, concentrating on the number of hops parameter simulation choosing some different values of the variable q.

10.1 Simulations of Different 1500 Nodes Networks

This section describes the simulation of different network containing 1500 nodes. The structures simulated are a simple ring, fully connected, double ring, wheel, N2Rpq and a structure with random generated links. In the next section, the simulation results are compared using plot.

10.1.1 Ring Structure

A 1500 node network is created, the capacity is set to 1 on all nodes and all links for simplicity. The maximum flow simulation is done at first. The result of this simulation becomes 2, independent of the choice of start and stop node. The result is the same as for a 10 node ring, which is expected. The maximum flow simulation is executed in order to test if the platform gives the same results.

The number of hops simulation is then executed. This simulation takes a short time to process, dependent of which CPU power which is available on the system the platform is running on. The number of hops simulation results given by the platform are:

- Sum number of hops: 843 750 000.

- Maximum number of hops: 750.
- Average number of hops: 375.25.

In section 9.1, the maximum numbers of hops for a 10 node ring is simulated to be 5, this is the same as $\frac{NumbersOfNodes}{2}$. For the 1500 node ring structure the same results appears, and this can be an argument in to prove that the number of hops simulation is calculating the correct maximum numbers of hops.

A traffic load simulation is performed on a computer in which fulfill the hardware requirements set in section 7.8 on page 48.

10.1.2 Full Mesh Structure

A full mesh network of 1500 nodes is generated as in section 10.1.1, but choosing *full mesh structure* to connect the links. The link capacity is for simplicity set to 1 for all of the links. A maximum flow simulation gives a result of 1499, which is correct according to equation 10.1. Equation 10.1 is only valid for full mesh structures.

$$MaxFlow = (NumberOfNodes - 1) \cdot LinkCapacity \tag{10.1}$$

The number of hops simulation on this full mesh network gives the following results:

- Sum number of hops: 2 248 500.
- Maximum number of hops: 1.
- Average number of hops: 1.

These results are not surprising, the sum of number of hops is actually the same as 2 times the number of links according to equation 2.1 on page 7. This is caused by the fact that the links are bidirected. Furthermore, the traffic load simulation is performed.

10.1.3 Double Ring Structure

A 1500 nodes double ring network is generated. The capacity is also here set to 1 for all links. A maximum flow simulation gives a result of 3, the same as for a double ring with 10 nodes.

The number of hops simulation gives the following results:

- Sum number of hops: 423 000 000.

- Maximum number of hops: 376.
- Average number of hops: 188.125.

10.1.4 Wheel Structure

The links are connected by choosing *wheel structure* when connecting the links.All the link capacities are set to 1. A maximum flow simulation gives a result of 3 since it is a three degree structure.

The number of hops simulation gives the following results:

- Sum number of hops: 422 998 000.
- Maximum number of hops: 375.
- Average number of hops: 188.125.

These results are almost the same as in double ring, "only" 2000 hops less in sum numbers of hops and 1 less in maximum numbers of hops. The traffic load simulation results are shown in section 10.2.

10.1.5 N2Rpq Structure

Here the links are connected by choosing *N2Rpq structure*, and the q set to 50. The link capacities are set to one, and again the maximum flow is 3 because the structure has a degree of three.

The number of hops simulation gives the following results:

- Sum number of hops: 39 574 500.
- Maximum number of hops: 31.
- Average number of hops: 17.6004.

10.1.6 Random Generation of Links

When choosing *random generation* while connecting the links the links are connected randomly to the nodes. The simulation results will of course differ in such networks. So the results are here just listed to test that the platform is able to simulate a random network. Maximum degree of the nodes were set to 3. A maximum flow simulation from node 1 to node 1500 is calculated to be 9 when a capacity of 3 is chosen on the links.

The number of hops simulation gives the following results:

- Sum number of hops: 19 551 900.
- Maximum number of hops: 15.
- Average number of hops: 8.69554.

These numbers are difficult to analyse. An option could be to read the link file and analyse the network for numbers of hops. This is a very time consuming analysis considering that the network has 1500 nodes and 2239 links, it is not appropriate to perform here.

10.2 Comparing the Different Network Structures

In section 10.1, a simulation of different 1500 nodes networks is performed. This section shows a comparison of these networks. In all the plots N0 are the ring, N1 the N2Rpq, N2 the double ring, N3 the wheel, N4 the random link generation, and N5 the full mesh.

10.2.1 Number of Hops

Figure 10.1 is the maximum number of hops plotted for each of the networks simulated in section 10.1.

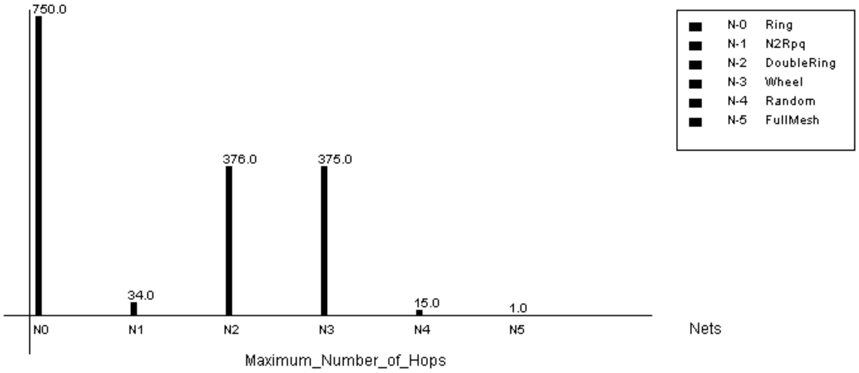


Figure 10.1: The maximum number of hops for 1500 nodes networks.

Figure 10.2 shows the average number of hops plotted for the same networks.

The sum number of hops simulation results can be found in appendix E on page 123.

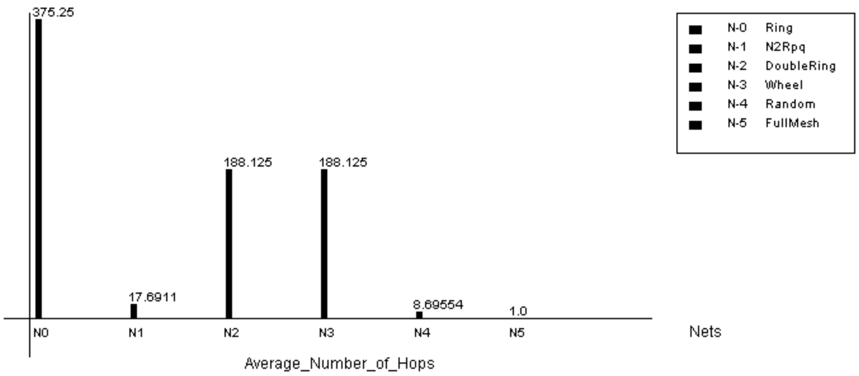


Figure 10.2: The average number of hops for 1500 nodes networks.

10.2.2 Traffic Load

All the traffic load simulations are simulated by letting node 1 sending out packets to all other nodes in the network. Maximum input load for nodes is shown in figure 10.3, and average input load for nodes is shown in figure 10.4.

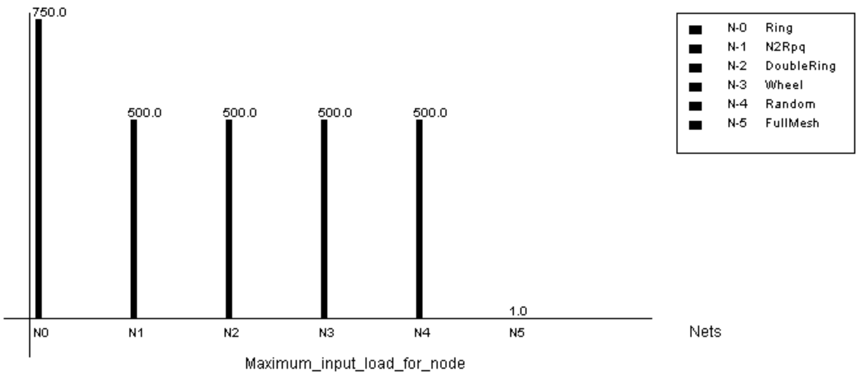


Figure 10.3: Maximum input load for nodes for the 1500 nodes networks.

The sum of input load for nodes can be found in appendix E on page 123 in figure E.2. The maximum transit load for nodes is shown in figure 10.5.

The average and sum transit load for nodes are found in appendix E, figure E.4 and E.3, respectively. The link load parameters are almost equal as the input load for node parameters, but the plotting results is found in figure E.5, E.6, and E.7 in appendix E on page 123. The number of links not used is shown in figure 10.6.

A plot of the maximum flow simulation results can be seen in appendix E on page 123, figure E.8.

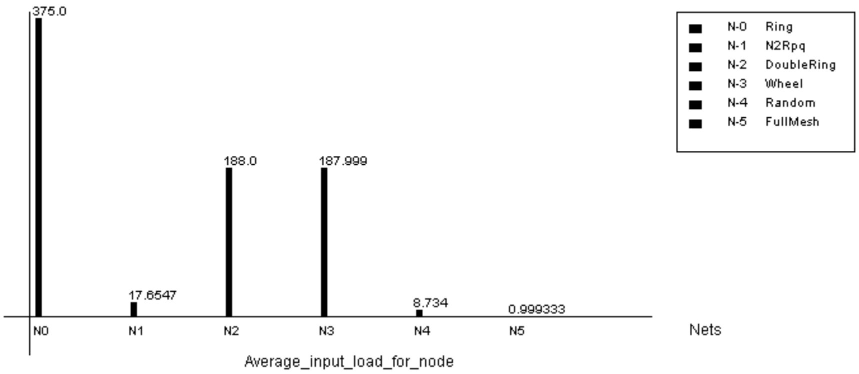


Figure 10.4: Average input load for nodes in the 1500 nodes networks.

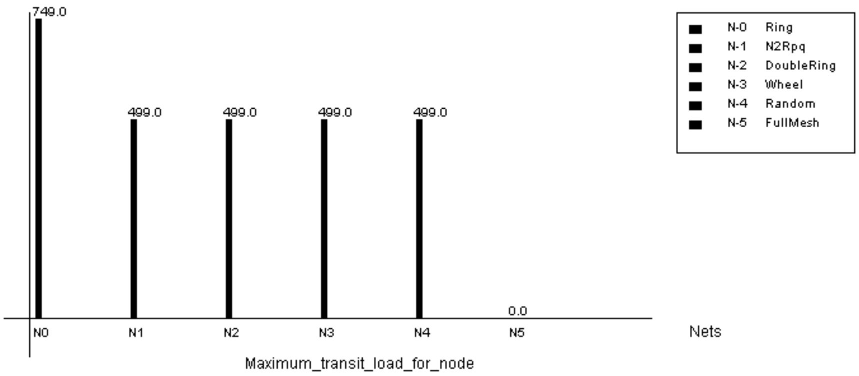


Figure 10.5: Maximum transit load for nodes in the 1500 nodes networks.

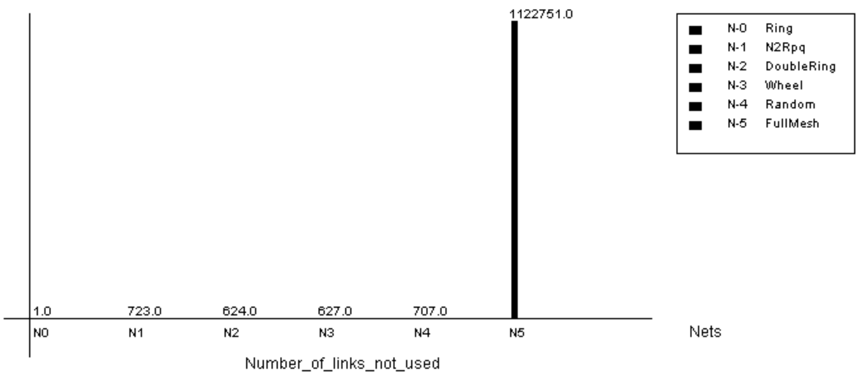


Figure 10.6: Links not used in the 1500 nodes networks.

10.3 Case Analysis of the N2Rpq Structure

In section 10.1.5, a 1500 nodes N2Rpq with q set to 50 is simulated. In this section different 1500 nodes N2Rpq structures are simulated to see if the performance can be optimized according to the q parameter. First a 1500 nodes N2Rpq network with a q set to 2 is simulated, a q with value 1 is not considered since it gives a double ring structure. The maximum flow is not considered since this parameter is independent of the of the q-parameter.

The N2Rpq Structure with q set to 2 is first tried to simulate. The number of hops simulation gives the following results:

- Sum number of hops: 214 305 000.
- Maximum number of hops: 190.
- Average number of hops: 95.3102.

These results are not as good as in 10.1.5. This is the minimum value of q, now the maximum value of q is considered, which can be determined by equations 10.2 and 10.3 [14]:

$$q \leq \frac{p-2}{2} \quad \text{if p is even number} \tag{10.2}$$

$$q \leq \frac{p-1}{2} \quad \text{if p is odd number} \tag{10.3}$$

The value p is the number of nodes divided by 2, $p = 750$. This is an even number so equation 10.2 is used to find maximum q. This equation gives a maximum q equal to 374. The number of hops simulation on q = 374 gives the following results:

- Sum number of hops: 21 319 500.
- Maximum number of hops:189.
- Average number of hops: 94.8145.

Different values of q are plotted for maximum numbers of hops in figure 10.7.

The average number of hops for the same structures as in figure 10.7 is shown in figure 10.8.

Both figure 10.7 and 10.8 are showing no single optimal q parameter. But there are indications that q around 25 or around 100 can give a optimized N2Rpq structure according to the number of hops parameter.

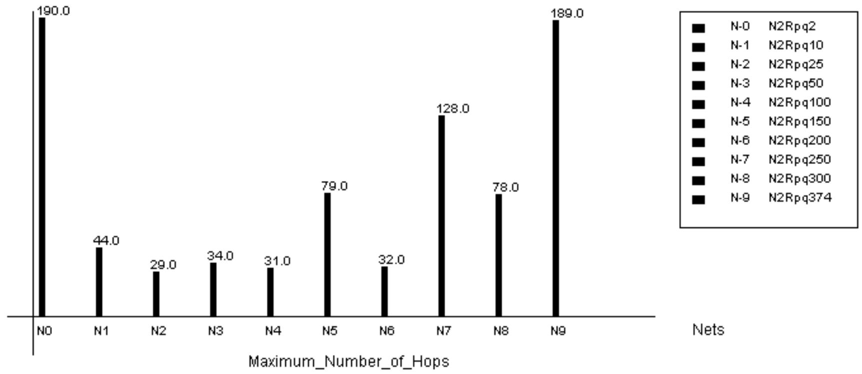


Figure 10.7: Maximum numbers of hops for different values of q for the N2Rpq, $p = 750$.

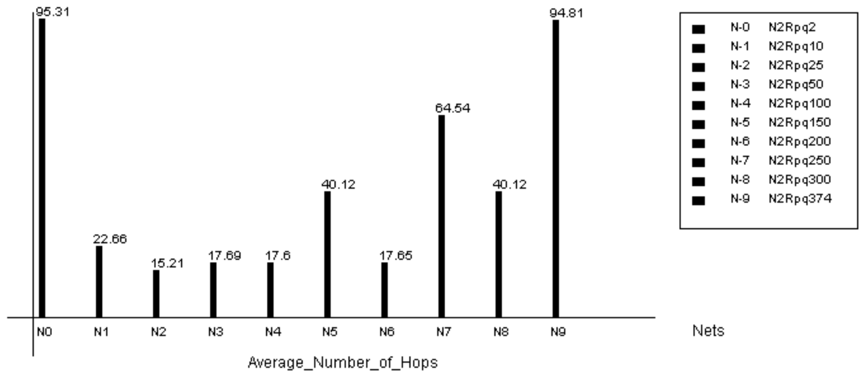


Figure 10.8: Average numbers of hops for different values of q for the N2Rpq, $p = 750$.

10.4 Summary

A goal of being able to simulate networks with 1500 nodes is set for this simulation platform. The platform is tested for the large-scale use with different network structures of 1500 nodes. These simulations is performed in a Windows OS on a regular desktop PC, with 1.8 GHz CPU and 512 MB of RAM.

The traffic load simulation for a 1500 node double ring network requires most hardware resources, around 1.6 GB of memory according to table 8.1 on page 60. To make this simulation run on the given computer, a part of the hard disk is allocated for "virtual RAM". The simulation time of the network is about 12 minutes. The double ring has the highest demand to memory because of the large amount of shortest paths in the network structure.

Not all of the results given here is checked to be true and some of them are not easy to calculate by hand, especially the results from the random generation of links.

In the next chapter possibilities for simulating even larger networks using the simulation platform will be introduced.

Chapter 11

Large-Scale Network Simulations

In this chapter, the feasibility of large-scale network simulation is discussed. The demand of being able to simulate networks of 1500 nodes must said to be fulfilled. But, what if even larger networks should be simulated, then a regular desktop PC reaches its limitations. What are hardware and software limitations of large-scale network simulation on PCs will be answered in this chapter. Moreover, a suggestion will be made according to how it can be possible to bring large-scale network simulations.

11.1 Network Simulations and its Limitation

Simulation has always been an indispensable tool in the design and analysis of networks. Due to the performance limitations of the simulators, usually simulations are done for small network and for short time scales. A PC meets its limitations while conducting large-simulations. As increasing the nodes, the demand of computation power increases proportionally. If the power of a regular desktop PC is insufficient, a super computer, or Linux cluster could be used for running the simulation.

- Use existing super computer.
  - Not easy to access.
  - Very expensive to buy.
  - Few resources are available.
  - Difficult to extend it.
- New area of parallel computation (Linux clustering).
  - Faster computation.
  - Less required memory.
  - Cheaper solution.
  - More technical support available.
  - Easy to extend it.

11.2 An Appropriate Choice

By looking at the mentioned choices in the previous section, the Linux clustering can be more suitable for the case with large-scale network simulations. One of the big advantages of Linux clustering is a cheaper and faster computation solution, compared with the price and power of super computers. Super computers tend to be very expensive and commonly not offered by small or medium organizations and institutions with limited funding resources.

Linux OS is freely available and open source codes, which makes it even more flexible and appealing choice. More and more business, and particularly academic world is joining Linux cluster community. Since Linux OS is a part of open source code, there is significant technical support available as well at low cost. One of disadvantages of using clustering is complex programming.

In the following section, Linux clusters is described with more details.

11.2.1 Using Linux Clusters

In its simplest form, a cluster is two or more computers that work together to provide a solution [15]. The main idea behind clusters is to join the computing power of nodes involved to provide higher scalability, more combined computer power, or to build in redundancy to provide higher availability. Computation power is increased with the coordination of work being done on different cluster's nodes. This leads in complex connectivity, configuration and sophisticated inter process communication between the nodes of cluster.

There are different types of Linux clusters available, for a high performance computation Beowulf Linux cluster is very famous. Beowulf cluster is a multicomputer architecture that can be used for parallel computations. It consists of one head node and rest computer nodes connected together via Ethernet or other networks. The system can be build on x86 computer architecture (PC) capable of running Linux OS, and other standard Ethernet adapters and switches. There is Beowulf type of software available, Beowulf is just one type of Linux Cluster. The main software components are, Linux OS, PVM (Parallel Virtual Machine) and MPI (Message Passing Interface). The head node controls the whole clusters and also serves files to compute nodes. PVM is a message passing system that enables a cluster computes to be used as a single distributed memory parallel computer. This network also referred as the virtual machine.

In order to run simulator software on a Linux cluster it must be programmed using parallel programming. And among the components of a parallel system, there must be a suitable language. The basic requirements for parallel programming are:

- Ability to spawn tasks on other nodes' processors.
- Sharing the data between tasks.
- Task co-ordination, via synchronization.

To achieve above mentioned goals, C++ uses PVM library.

11.3 Summary

As mentioned in section 10.4 on page 93, the simulation platform's requirement of being able to simulate 1500 nodes is fulfilled. Even if it is running on a regular desktop computer. A Linux cluster will probably speed up the simulation process, and will may be the only solution if larger networks should be simulated. However, a performance test of the platform for network with more than 1500 nodes is not covered in this report.

Part V

Evaluation and Discussion

Chapter 12

Conclusion

The purpose of this project was to make a simulation tool to simulate large-scale network structures. The definition of large-scale networks is not very precise, however the requirements for this tool was set to be able to simulate minimum 1500 nodes.

The tool is able to auto generate 6 network structues. These are: Simple ring, fully connected, wheel, double ring, N2Rpq, and random structure. Methods for generating other structures can easily be added. It is possible to for manually link generation.

Three different siremulations are conducted in the tool. Those are the maximum flow, the Number of Hops and the traffic load. The maximum flow simulation finds the maximum flow in the network, where each link is given a capacity. This simulation uses a breadth first algorithm, where the computation time is low, about 1-2 seconds, no matter how large the network is. When the platform is implemented, only the capacities of the links are considered, the capacities of the nodes are considered to be infinite.

In the number of hops simulation, the shortest distance from one node to any other node can be found. The algorithm which was used is Dijkstra's shortest path algorithm. The output of the number of hops simulation is the sum of hops, the maximum, and the average number of hops in the given network. The simulation time depends upon processing power of CPU, the number of nodes in the network and the network structure. The simulation time using a PC with a 1.8 GHz CPU and a fully connected network (which is the most demanding network structure for this simulation) with 600 nodes is 9 seconds. For networks with 1500 nodes the simulation time will be maximum 2 minutes and 42 seconds. The simulation time of the number of hops algorithm is satisfactory.

In the traffic load simulation there are two options. One option is to send one packet from one node to another, the other option is to send one packet from one node to all other nodes. When one packet is sent from one node to another, the simulator finds the shortest path and sends the packet through that path. When the simulator sends one packet to all nodes, the simulator also finds the shortest path from the sending node to all other nodes, but if there is more than one shortest path the simulator must decide which path to choose by looking at the load at each link and node. The algorithm used to find the shortest path is Dijkstra's shortest path algorithm, and the algorithm used to decide which path to use is developed by the group. The output of the traffic load simulation is the sum, average and maximum node- and link-input, output and transit load for the given network.

The simulation time of the traffic load simulation depends on the number of nodes, the maximum number of hops and the maximum number of shortest paths from any node to the sending node. It also depends on how much RAM the computer has. The simulation time, using a computer with 512 MB of RAM and a double ring structure (which is the most demanding commonly used network structure for this algorithm) with 1500 nodes is 7 min and 21 seconds.

A 1500 node double ring traffic load simulation demands at least 1.6 GB of RAM. If the computer has less, the operating system starts to use the hard disk as RAM, and the access time of the hard disk is much longer that to the RAM. If the computer has enough memory, the simulation time will be shorter, but the access time measurements are not established. The maximum simulation time of 7.21 minutes for a 1500 node network seems satisfactory.

These simulations work perfect for all the networks tested, which primarily are the ring-, double ring-, wheel-, full mesh-, and N2Rpq graphs. They also work for small manually configured in networks where the maximum flow, shortest path, and load easily can be calculated by hand.

The tool is programmed using C++ and the user interface for the tool is build up in a command prompt. A user friendly graphical user interface is made using Java and is capable of plotting the simulation results of the networks in a project. It is developed to make it easy to compare the results of the simulations.

The tool is useful to provide an overview of a large-scale network. It only supports three simulations, which are some important simulations in order to see how the network performs.

This tool is also expendable for other simulation parameters, for instance a price parameter of the nodes and links. It can be done by making a new class for each simulation and add it to the existing tool. Furthermore, it would be nicer if the simulation tool had a more user friendly user interface. The improvements can be considered for the future‘s development of the simulation tool.

Bibliography

[1] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, 4 edition, 2003.

[2] *Cisco's home page*. Cisco. http://www.cisco.com/warp/public/103/19.html.

[3] Ph.D Prof. Peter Ladkin. 1998. http://nakula.rvs.uni-bielefeld.de/made/folie/folie05.html.

[4] Henrik Schiøler. *Note 1 from the first lecture in Traffic Analysis*.

[5] Douglas E. Comer. *Computer Networks and Internets*. Prentice Hall, 2 edition, 1999.

[6] Catalyst_Workgroup_Switch. *Cisco's home page*. Cisco, 1994. http://www.cisco.com/warp/public/614/4.html.

[7] Imagine_That_Inc. *Simulation Definition*. Imagine That Inc., 2003. http://www.imaginethatinc.com/sols_sim_def.html.

[8] Jesse Liberty. *C++ Unleashed*. Sams, 1 edition, 1999.

[9] H.M. Deitel and P.J. Deitel. *C++ How to program*. Prentice Hall, 2 edition, 1998.

[10] Andreas Quale and Johan Havnen. *C++ og objektorientert programmering*. Universitetsforlaget, Oslo, 2 edition, 1998.

[11] Richard Johnsonbaught. *Discrete Mathematics*. Prentice Hall, 4 edition, 2000.

[12] NIST. *Augmenting path*. NIST, 2003. http://www.nist.gov/dads/HTML/augmentngpth.html.

[13] D.A.Holton & J.Sheehan. *The Petersen Graph*. Cambridge University Press, 1 edition, 1993.

[14] Ole Brun Madsen. *Graphs Degree 3, N2Rpq*. Department of Control Engineering, Aalborg University, 1 edition, 2003.

[15] Manoel, Carlane, Ferreira, Hill, Leitko, and Zutenis. *Linux Clustering with CSM and GPFS*. IBM, 2 edition, 2002.

[16] Francis T.Boesch. *Large Scale Networks: Theory and Design*. IEEE Press, 1 edition, 1975.

[17] Ole Brun Madsen. *Graphs Degree 3*. Department of Control Engineering, Aalborg University, 1 edition, 2003.

Part VI

Appendices

Appendix A

Graph Theory

When studying network topologies and their performance, knowledge to graph theory is needed. The basic concepts of graph theory and flow in networks are the essential ingredients of an analytical approach to the design and analysis of large-scale networks.

The basic mathematical concept underlying large-scale networks is some set of vertices (nodes) which are interconnected by lines or curves called edges. Such an object is called a graph or a digraph[1], as if the edges are undirected or directed with arrows. Actually, if each edge in a graph is assigned a weight, the graph is a network. The weight may represent length of a cable, bit rate, probability of failure of a power line, etc. Many problems regarding the design or analysis of networks can be formulated via this basic mathematical model.

A.1 Graph Theory Definitions

The first question to answer is, what is a graph? A graph is a set of points in a plane (or in 3-space) and a set of line segments (possibly curved), each of which either joins two points or joins a point to itself. A mathematical definition of a graph is: A graph $G = (V, E)$ is a mathematical structure consisting of two sets V and E. The elements of V are called vertices (nodes), and the elements of E are called edges (links). A graph consisting of a single vertex is called a trivial graph. In other words, a graph is a mathematical object that can be used to model a network. From some graph theory parameters are defined [13].

A walk in a graph is an alternating sequence of nodes and lines, starting with a node and ending with a node. The number of lines in the walk is defined as the length of the walk [16].

[1] Digraph is the abbreviation for directed graph

- Trail: a walk with no repeated edges.
- Path: a walk with no repeated vertices.
- Cycle: a closed path with at least one edge. The length of the shortest graph cycle is called a girth.

A connected graph is a graph where every pair of distinct vertices has a walk between them. The distance between two nodes in a graph G, is the length of the shortest path between them. If there is no such path, the distance is infinite.

- Endpoints: a set of one or two vertices associated to each edge.
- Loop: an edge where both end points are the same.
- The degree of a vertex: the number of edges in the graph that has same the node as an endpoint (plus twice the number of self-loops).
- Adjacent vertices: Two nodes are adjacent if there is an edge that has them as endpoints.
- Incidence: The relationship between an edge and its endpoints.

If a graph is connected, the path of least length from a vertex u to a vertex v in the graph is called the shortest path from u to v, and its length is called the distance from u to v. The eccentricity of a vertex v is defined as the distance from v to the most distant vertex from v. A vertex of minimum eccentricity is called center. The eccentricity of a center of a graph is called the radius. The maximum eccentricity among all vertices of a graph is called the diameter. Furthermore, when a graph be weighted, there is a value associated with each edge (e.g. link speed, cost, etc.).

- Subgraph: A graph $\grave{G}$ = ($\grave{N}$, $\grave{A}$) is a subgraph of G = (N, A) if $\grave{N} \subseteq$ N and $\grave{A} \subseteq A$. A maximal connected subgraph of a graph is called a component. A subgraph is said to span its set of vertices.
- Isomorphism: Two graphs G_1 and G_2 are isomorphic if there is a 1-1 mapping f: $v_1 \rightarrow v_2$ such that (v_1, v_2) ϵ E_1 if and only if (f(v_1), f(v_2)).
- Tree: a connected, simple graph without cycles. Any tree with n nodes has n-1 edges.
- Spanning Tree: A tree T is a spanning tree of a graph G if T is a subgraph of G that contains all the vertices of G.
- Minimal Spanning Tree: Let G be a weighted graph. A minimal spanning tree of G is a spanning tree of G with minimum weight.

- Star: a tree where only 1 node has degree greater than 1.
- Chain: a tree where no node has degree greater than 2.

To make a navigation or a search in a graph there are two alternatives, namely breadth first search and depth first search. Breadth first search processes all nodes in a given level before proceeding to next level, and depth first search process next level as soon as possible. The searching algorithms has different advantages; breadth first quickly develops a big search space and finds the optimal solution. Depth first find a suboptimal solution faster. Both of them need a method for testing if a vertex has already been visited.

To optimize a connected graph, the minimum weight has to be found. For this purpose, a technique called Minimal Spanning Tree (MST), can be used. Both Kruskal's and Prim's algorithm can be used to find MST. The number of hops defined in section 2.11 on page 9 will grow past a reasonable level as the number of nodes and traffic grow, and MST is not a good solution. Then an another tree design is to be considered, namely the Shortest Path Trees (SPT). For this purpose Dijkstra's Algorithm is used. A comparison between SPT versus MST shows that:

- SPT has lower utilization of the links.
- MST has lower cost.
- Important: SPT finds smaller average number of hops.

The definition of a planar graph is a graph which is isomorphic to a plane graph, which means that it can be drawn as a graph with no edges crossing each other.

Since some graphs of degree 3 are analysed in the report, a general spanning tree for a 3 degree graph is shown in figure A.1 [17].

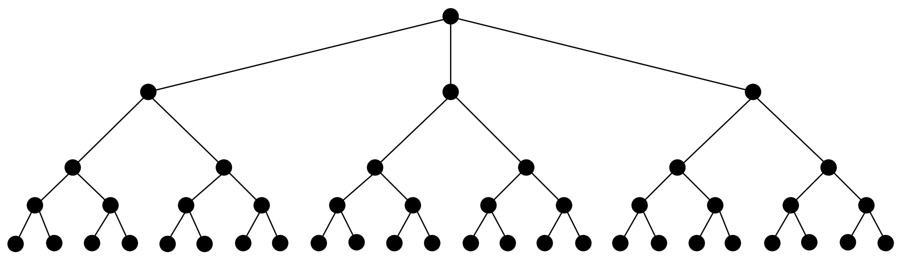


Figure A.1: A general spanning tree for a 3-degree graph.

Appendix B

Scenarios and Guidelines to the Use Cases

The guidelines and scenarios for the use case *1. The user creates a new project* is described in chapter 5.2.3 on page 28. Guidelines and scenarios for the rest of the use cases are discussed in this appendix.

B.1 The user opens a project

- The user choose "open project" from the project menu, the platform asks for a name, the user types a new name, the platform accepts name and the project opens.
  - Preconditions: The platform is in the start menu, and project or projects exist.
  - Trigger: The user opens an existing project.
  - Description: The user choose open project and type in the name for the given project. The platform opens the project.
  - Post Conditions: An existing project is opened and enters the project menu.
- The user choose "open project" from menu, the platform asks for a name and the user types a name. The platform do not find this project name and ask for new name, the user types a new name, the platform accepts name and the project opens.
  - Preconditions: The platform is in start menu and project or projects exists.
  - Trigger: The user opens an existing project.
  - Description: The user choose open project and type in the name for the given project, the platform does not find the project name and asks for a new name. The user types in a new name, the platform finds it and the project is opened.
  - Post Conditions: An existing project is opened and the platform enters the project menu.

B.2 The user close the project

- The user choose "close project" from the project menu and the project closes.
  - Preconditions: The platform is in project menu.
  - Trigger: The user closes the project.
  - Description: The user choose close project from the project menu and the project is saved and closed. The platform enters the start menu.
  - Post Conditions: The project is saved, and the platform enters the start menu.

B.3 The user specifies a new network in the project

- The user choose "new network" in project menu. The platform asks for a name, the user types a name and the name is accepted. Then the user choose enter manually in the design network menu, types the number of nodes and enter the link list manually.
  - Preconditions: The platform is in the project menu.
  - Trigger: The user defines a new network.
  - Description: The user choose "new network" in project menu and gives the network a name. The user choose enter manually in the design network menu and types in number of nodes and link list. The platform creates a new network and enters the network menu.
  - Post Conditions: A new network is created and the platform enters the network menu.
- The user choose "new network" in the project menu and the platform asks for a name, the user types a name. Then the platform finds a similar name and asks for new name, the user types new name which is accepted.Then the user choose enter manually in the design network menu, types the number of nodes and enter the link list.
  - Preconditions: The platform is in the project menu and other networks has to be defined.
  - Trigger: The user defines a new network.
  - Description: The user choose "new network" and give the network a name. The platform asks for a new name because the name is not unique and the user types in a new name.The user choose enter manually in the design network menu and types in number of nodes and link list. The platform is creating a new network and enters the network menu.

- Post Conditions: A new network is created and the platform enters the network menu.

B.4 The user adds an existing network to the project

- The user choose "add network" in the project menu, the user types in a name and the platform accepts the name.
  - Preconditions: The platform is in project menu and there have to be existing networks.
  - Trigger: The user adds an existing network to the project.
  - Description: The user choose "add network" in the project menu and types in a name. The platform adds the network to the project.
  - Post Conditions: The network is added to the project and platform is in project menu.
- The user choose "add network" in the project menu and types in a name, the name is not accepted by the platform because it is not existing, the user types in a new name and a network is added to the project.
  - Preconditions: The platform is in project menu, there have to be existing networks.
  - Trigger: The user adds an existing network to the project.
  - Description: The user choose add network in the "project menu" and types in a name, the platform does not find the network, the user types in a new name and the platform adds the network to the project.
  - Post Conditions: The network is added to the project and platform is in project menu.

B.5 The user opens an existing network in the project

- The user choose "open network" in the project menu and the user types in a name. The platform finds the name and open the network.
  - Preconditions: The platform is in the project menu and there have to be existing networks in the project.
  - Trigger: The user opens an existing network that is in the project.
  - Description: The user choose "open network" in the project menu and types in a name. The platform opens the network and the network menu appears.

  - Post Conditions: The network is opened, the platform is in the network menu and the network is ready to be reconfigured and simulated.

- The user choose "open network" in the project menu and the user types in a name. The name is not accepted by the platform because it is not existing, the user types in a new name and a network is opened.
  - Preconditions: The platform is in the project menu and there have to be existing networks in the project.
  - Trigger: The user opens an existing network that is in the project.
  - Description: The user choose "open network" in the project menu and types in a name. The platform asks for a new name because the network is not in the project. The user types in a new name, the platform opens the network and the network menu appears.
  - Post Conditions: The network is opened, the platform is in the network menu and the network is ready to be reconfigured and simulated.

B.6 The user removes a network from the project

- The user choose "remove a network" from the project and the platform removes the network.
  - Preconditions: The platform is in the project menu.
  - Trigger: The user removes a network from the project.
  - Description: The user choose "remove network" in the project menu and the user types in a network name to remove. The platform removes the network from the project. menu appears.
  - Post Conditions: The network is removed, and the platform shows the project menu.

B.7 The user makes changes in the network

- The user choose "reconfigure network" in the network menu. The platform suggests the same file name to overwrite the old file, the user accepts this and make changes. The platform saves changes in the same file.
  - Preconditions: The platform is in the network menu.
  - Trigger: The user makes changes in the network and save the changes in the same file.
  - Description: The user choose "reconfigure network" in the network menu and the platform suggest the same file name. The user accepts

and the user manipulate the network. The platform saves the new manipulated network with the same name.

  - Post Conditions: The manipulated network is saved, and the platform is in the network menu.

- The user selects "reconfigure network" in the network menu. The platform suggests the same file name to overwrite the old file. The user does not want this and writes another file name to keep the old network. The user make changes and the platform saves changes in a new file.

  - Preconditions: The platform is in network menu.
  - Trigger: The user makes changes in the network and save the unconfigured network.
  - Description: The user choose "reconfigure network" in the network menu. The platform suggests the same file name, the user types a new name in order to keep the unconfigured network. The user manipulates the network and the platform saves the new manipulated network with the new name.
  - Post Conditions: Both unconfigured and configured network is saved, and the platform is in the network menu with the manipulated network active.

- The user selects "reconfigure network" in the network menu and the platform suggests the same file name to overwrite the old file. The user writes another file name to keep the old network, then the platform asks for a new name because the file name exists. The user types a new name and make changes, the platform saves changes in a new file.

  - Preconditions: The platform is in the network menu and more than one network has to be active in the project.
  - Trigger: The user makes changes in the network and save the unconfigured network.
  - Description: The user choose "reconfigure network" in the network menu, the platform suggest the same file name and the user types a new name. The platform does not accept the filename because the name is not unique, the user types in a new filename and the user manipulates the network. The platform saves the new manipulated network with the new name.
  - Post Conditions: Both unconfigured and configured network is saved, and the platform is in the network menu with the manipulated network active.

B.8 The user will auto generate a network topology

- The user enters the choice for auto generate in the design network menu and types in number of nodes, then the menu for selecting a topology appears. The user choose fully connected network in the topology menu.
  - Preconditions: The platform is in the design network menu and the number of nodes have to be typed in.
  - Trigger: The user generates a fully connected network.
  - Description: The user choose auto generate in the network menu and types in the number of nodes. The user choose fully connected network in the auto generate menu, and the platform makes a link list and goes to the simulation menu.
  - Post Conditions: The simulation menu appears so the user can choose which parameters to simulate, and then simulate the network.

- The user enters the choice for auto generate in the design network menu and types in number of nodes, then the menu for selecting topology appears. The user choose "ring network" in the topology menu.
  - Preconditions: The platform is in the design network menu and the number of nodes have to be typed in.
  - Trigger: The user wants to generate a ring network.
  - Description: The user choose auto generate in the design network menu and types in the number of nodes. The user choose ring network in the auto generate menu, and the platform makes a link list and goes to the simulation menu.
  - Post Conditions: The simulation menu appears so the user can choose which parameters to simulate, and then simulate the network.

- The user enters the choice for auto generate in the design network menu and types in number of nodes, then the menu for selecting topology appears. The user choose "double ring network" in the topology menu.
  - Preconditions: The platform is in the design network menu and the number of nodes have to be typed in.
  - Trigger: The user wants to generate a double ring network.
  - Description: The user choose "auto generate" in the design network menu and types in the number of nodes. The user choose "double ring" network in the auto generate menu, and the platform makes a link list and goes to the simulation menu.

  - Post Conditions: The simulation menu appears so the user can choose which parameters to simulate, and then simulate the network.

B.9 The user closes the active network in the project

- The user choose "close network" from the network menu and the platform closes the active network.
  - Preconditions: The platform is in the network menu.
  - Trigger: The user closes the active network in the project.
  - Description: The user choose "close network" in the network menu and the platform saves the network and closes it. The project menu appears.
  - Post Conditions: The network is saved and the platform shows project menu.

B.10 User selects simulation parameters and starts simulation

- The user selects "maximum flow" and start simulation in the simulation menu .
  - Preconditions: The platform is in the simulation menu.
  - Trigger: The user simulates the network for the max flow parameter.
  - Description: The user selects "maximum flow" and choose start simulation in the network menu. The platform asks for source and sink, the user type the source node and sink node. The platform starts the simulation and saves the simulation results. The platform return to the network menu.
  - Post Conditions: The network has been simulated for the maximum flow parameter and is back in the network menu.
- The user select "number of hops" and start simulation in the simulation menu.
  - Preconditions: The platform is in the simulation menu.
  - Trigger: The user wants to simulate the network for the number of hops parameter.
  - Description: The user selects "number of hops" and start simulation. The platform starts the simulation and saves the simulation results. The platform return to the network menu.

 - Post Conditions: The network has been simulated for the max flow parameter and is back in the network menu.

- The user select traffic flow and start simulation in the simulation menu .

  - Preconditions: The platform is in the simulation menu.
  - Trigger: The user wants to simulate the network for the traffic load parameter.
  - Description: The user selects traffic load and start simulation. The platform starts the simulation and saves the simulation results. The platform return to the network menu.
  - Post Conditions: The network has been simulated for the traffic load parameter and is back in the network menu.

B.11 The user stops the simulation

- The user wants to stop the simulation and user pushes the assigned key for stopping simulation. The simulation stops and the platform return to the network menu.

  - Preconditions: A simulation is running.
  - Trigger: The user stops the simulation when the simulation is running.
  - Description: The platform is running a simulation, the user stops the simulation by pushing assigned key, the platform stops simulation and returns to the network menu.
  - Post Conditions: The platform has stopped the ongoing simulation and is back in the network menu.

B.12 The platform plots the simulation results from different networks

- The user selects "plot results" from project menu and the platform plots the results from simulations.

  - Preconditions: The platform is in project menu.
  - Trigger: The user plots the simulated networks.
  - Description: The user choose "plot results" in the project menu, the platform plots the simulation results.
  - Post Conditions: The network simulation results is plotted, and the platform is back in project menu.

- The user selects "plot results" from project menu and the platform can not find any simulation results to plot because no networks has been simulated.
  - Preconditions: The platform is in project menu.
  - Trigger: The user plots the simulated networks.
  - Description: The user choose "plot results" in the project menu, the platform return a message that no simulation results are available.
  - Post Conditions: No network simulation results is plotted, and the platform is back in the project menu.

B.13 The platform shows a status bar during simulation

- Simulator is running, after a given number of iterations a character is written to the screen.
  - Preconditions: A simulation is running.
  - Trigger: The platform shows the user the progress in the simulation.
  - Description: When simulation is started, a status bar is showing the simulation progress. When the simulation is ended, the status bar disappears.
  - Post Conditions: The platform has closed the status bar, and is back in the network menu.

Appendix C

The UML classes

This appendix is showing the UML classes for the classes used in the simulation class design.

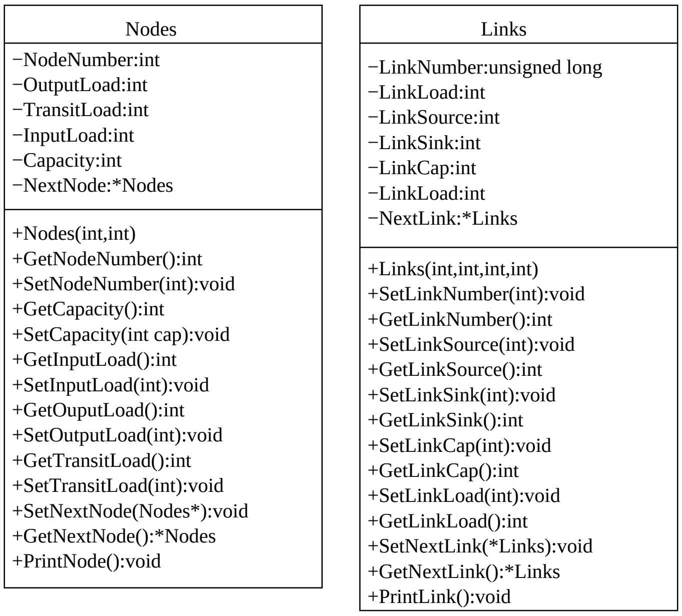


Figure C.1: Nodes and links class methods.

```
Project
−ProjectName:char
−NetworkCounter:int
−ProjectFileName:char
−OpenOK:bool
−NetwListPtr:Network*
+Project()
+CreateNewProject():void
+ListNetworksInProjects():void
+AddNetwork(char*):void
+AddNetworkInit():void
+DeleteNetwork():void
+SaveProject():void
+OpenProject(char*):void
+NewNetwork():void
+OpenProject():void
+GetOpenStatus():bool
+Menu():void
+RemoveNetwork():void
```

```
Simulation
−NumberOfLinks:unsigned long
−NumberOfNodes:int
−AvgNumberOfHops:float
−AvgLinkLoad:float
−AvgNodeInputLoad:float
−AvgNodeTransitLoad:float
−MaxNumberOfHops:int
−StartNode:int
−StopNode:int
−MaxFlow:int
−MaxNodeInputLoad:int
−MaxNodeOutputLoad:int
−MaxNodeTransitLoad:int
−MaxLinkLoad:int
−NetworkName:char
−NodeFileName:char
−LinkFileName:char
−ResultFileName:char
−AvgNodeOutputLoad:float
−LinksNotUsed:int
−SumLinkLoad:int
−SumNodeInputLoad:int
−SumNodeOutpuLoad:int
−SumNodeTransitLoad:int
−SumNumberOfHops:
  unsigned long
+Simulation(char*)
+ReadResultFile():void
+WriteResultFile():void
+CountLinks(char *name):void
+CountNodes(char *name):void
+SimMaxFlow():void
+SimNumbersOfHops():void
+SimTrafficFlow():void
+SimMenu():void
```

```
Network
−LastNodeCounter:int
−NumberOfNodes:int
−LastLinkCounter:int
−NetworkName:char
−NodeFileName:char
−LinkFileName:char
−ResultsFileName:char
−OpenOK:bool
−NodeListPtr:Nodes*
−LastNodePtr:Nodes*
−LinkListPtr:Links*
−LastListPtr:Links*
−NextNetwork:Network*
−NumOfLinks:unsigned long
+Network()
+CreateNewNetwork(*char):void
+DeleteNetworkMem():void
+AddNodeToList(int):void
+DeleteNode(int):void
+DeleteLink(int,int):void
+PrintNodes():void
+PrintLinks():void
+CreateRing():void
+CreateFullyConnnected():void
+CreateManually():void
+TestNetwork():void
+ChechLinks(int):void
+EditSpecLinkCap():void
+EditAllLinkCap():void
+EditSpecNodeCap():void
+EditAllNodeCap():void
+EditNodeCap(int,int):void
+EditLinkCap(int,int,int):int
+CapMenu():void
+CreateFiles():void
+RearrangeNetwork():void
+FixLinks(int,int):void
+SaveNetwork():void
+WriteNodes():void
+WriteLinks():void
+InitResFile():void
+Simulate():void
+GetNumberOfNodes():int
+GetLastNodeCounter():int
+GetNumOfLinks():unsigned long
+GetLastLinkCounter():int
+DoesNodeExist(int):bool
+SearchNode(int):*Nodes
+SearchLink(int,int):*Links
+SetNextNetwork(Network*):void
+GetNextNetwork():*Nodes
+SetNetworkName(char*):void
+GetOpenStatus():bool
+GetNetworkName():*char
+Menu():void
+OpenNetwork(char*):void
+AddLinkToList(int,int,int):void
+CreateDoubleRing():bool
+CreateN2Rpq():bool
+CreateRandom():bool
+CreateWheel():bool
+ListData():void
+NodeCapMenu():void
+Plot():void
```

Figure C.2: Project, network and simulation class methods.

Figure C.3: Traffic load, number of hops, maximum flow, and status bar class methods.

Appendix D

Plotting Results for Basic Network Structures

In this appendix the plotting results which is missing from chapter 9 are listed. In all plotting figures N0 is a ring structure, N1 a Petersen graph, N2 a double ring, N3 a wheel graph, and N4 a full meshed structure. All topologies have 10 nodes.

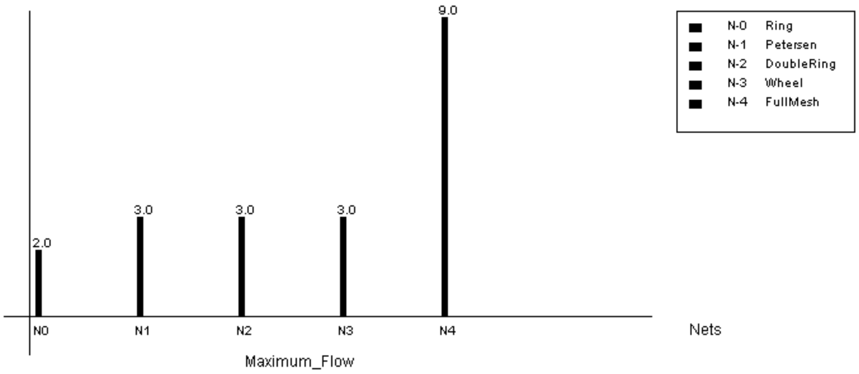


Figure D.1: Maximum flow.

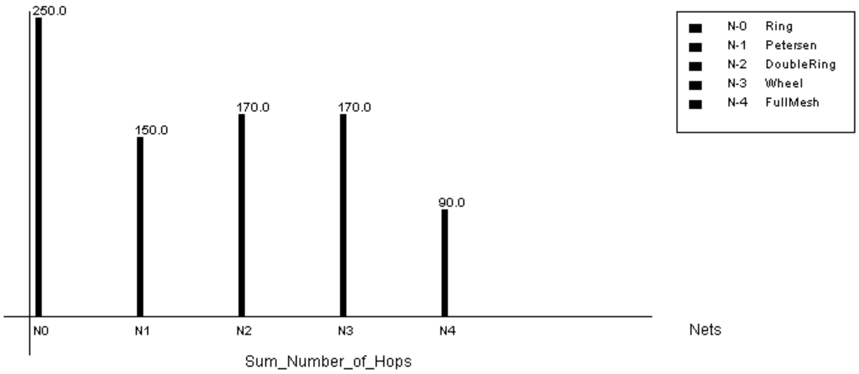


Figure D.2: Sum number of hops.

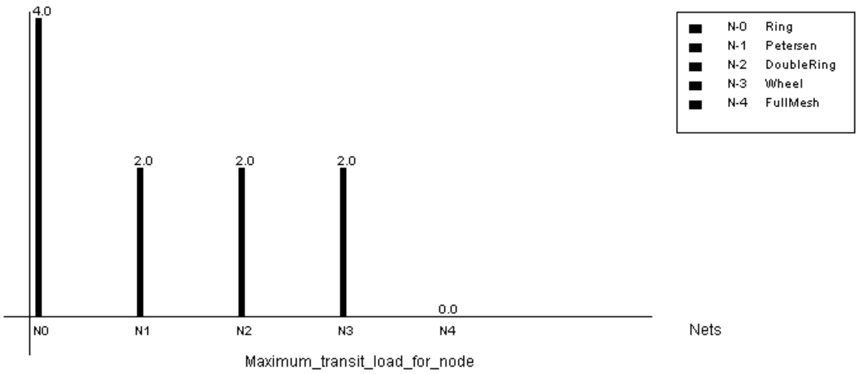


Figure D.3: Maximum transit load for nodes.

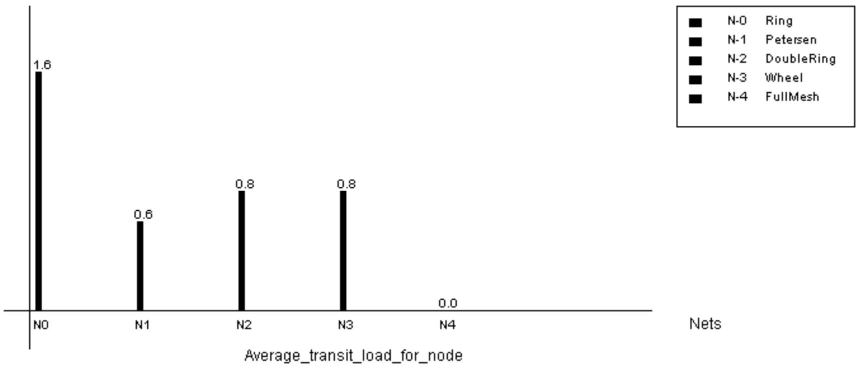


Figure D.4: Average transit load for nodes.

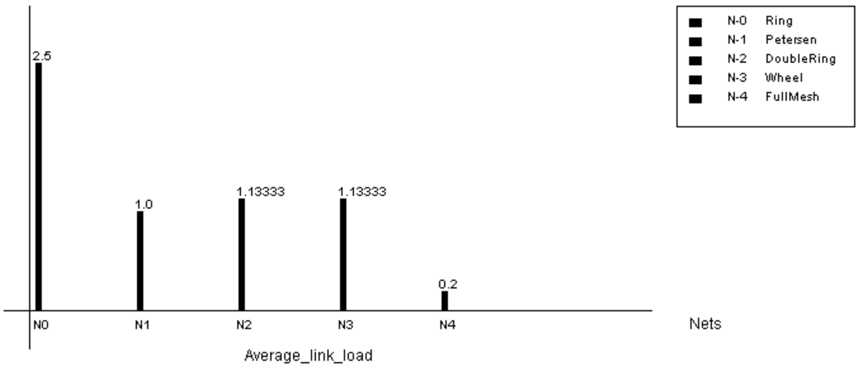


Figure D.5: Average link load.

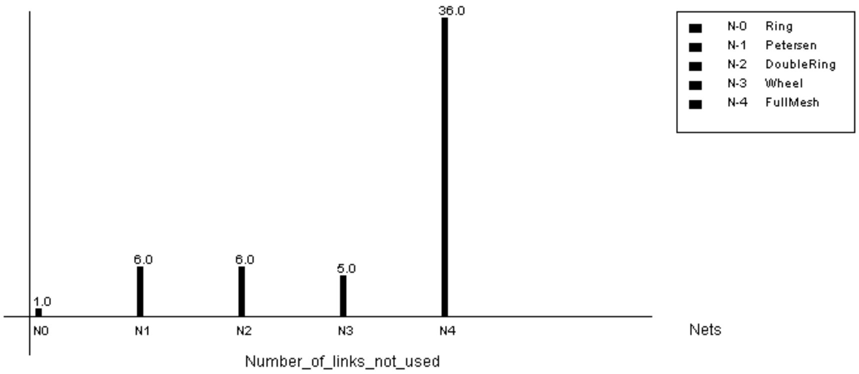


Figure D.6: Links not used.

Appendix E

Large-Scale Plotting Results

This appendix shows plotting results which are not shown in section 10.2 on page 88.

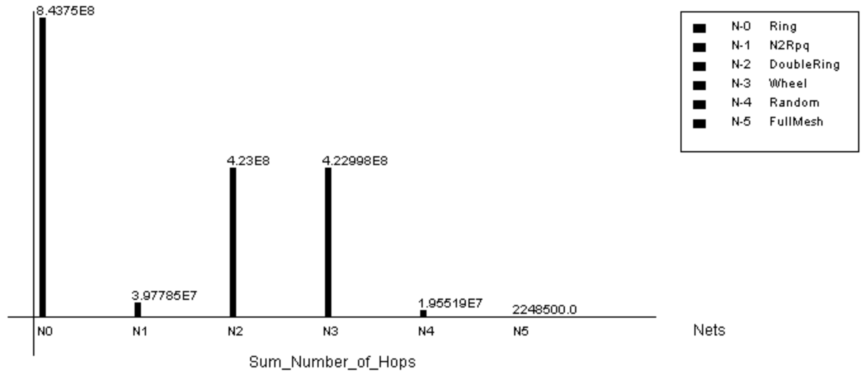


Figure E.1: Sum number of hops for the large-scale networks given in millions.

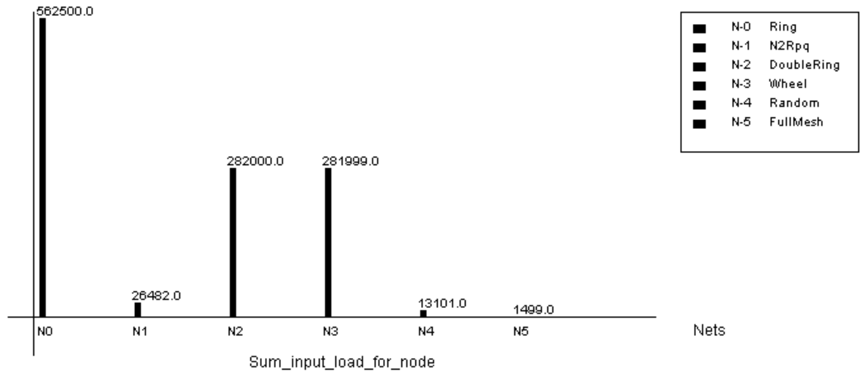


Figure E.2: Sum input load for nodes for the large-scale networks.

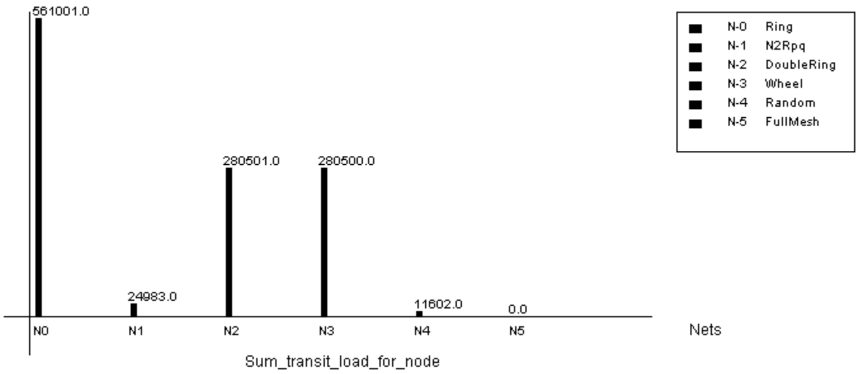


Figure E.3: Sum transit load for nodes for the large-scale networks.

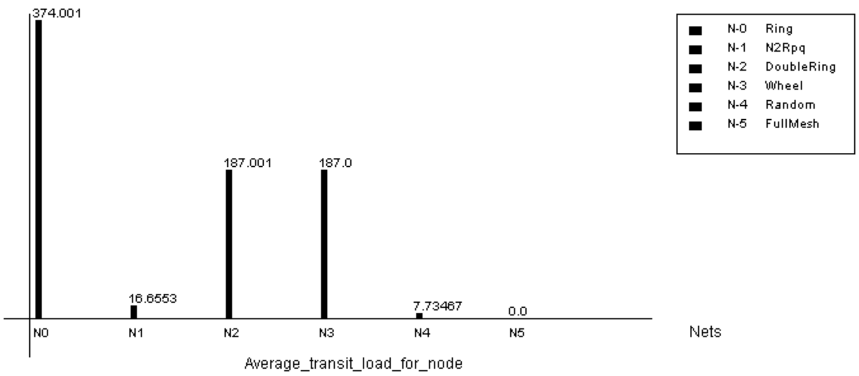


Figure E.4: Average transit load for nodes in the large-scale networks.

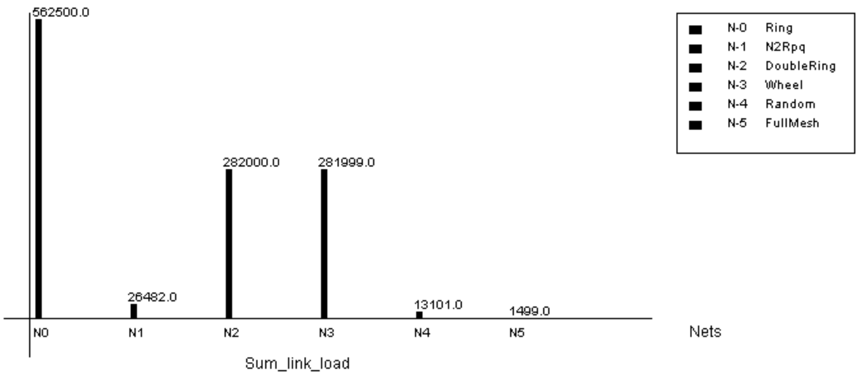


Figure E.5: Sum link load for the large-scale networks.

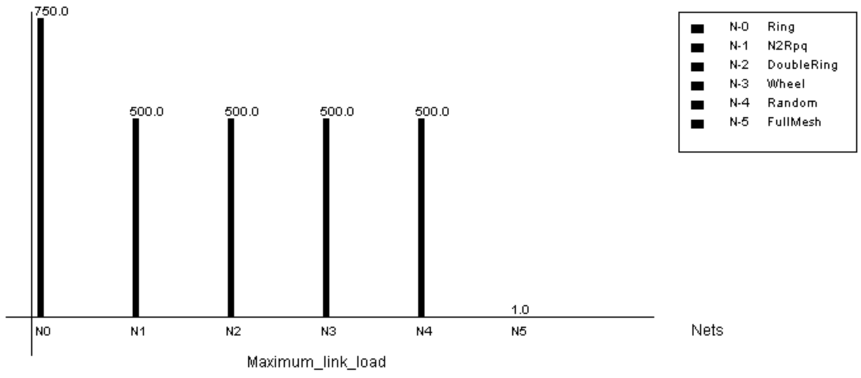


Figure E.6: Maximum link load for the large-scale networks.

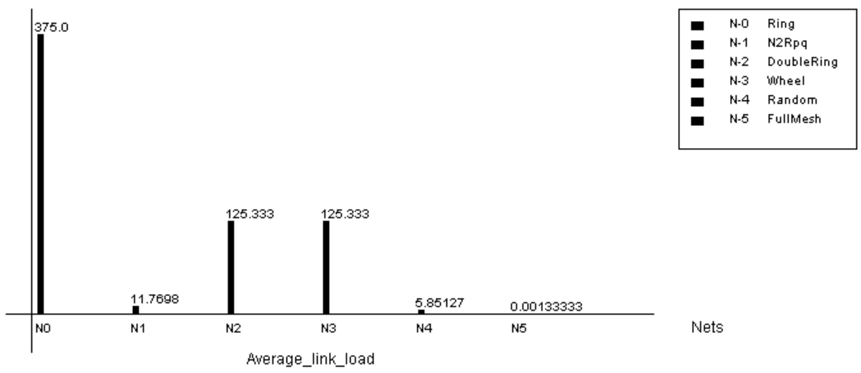


Figure E.7: Average link load for the large-scale networks.

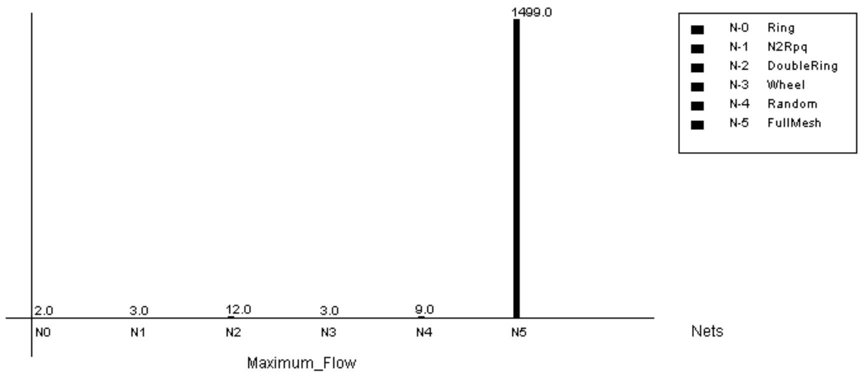


Figure E.8: Maximum flow for the large-scale networks.

Appendix F

List of Acronyms

AWT	Abstract Windows Toolkit
CRC	Class Responsibility and Collaboration
EIGRP	Enhanced Interior Gateway Routing Protocol
GUI	Graphical User Interface
HCI	Human Computer Interaction
IGRP	Interior Gateway Routing Protocol
IP	Internet Protocol
LAN	Local Area Network
MPI	Message Passing Interface
MST	Minimal Spanning Tree
NCP	Network Control Protocol
OOA	Object Oriented Analysis
OOD	Object Oriented Design
OS	Operating System
OSPF	Open Shortest Path First
PNG	Portable Network Grapichs
PVM	Parallel Virtual Machine
QoS	Quality of Service
RAM	Random Access Memory
RMON	Remote Monitoring
RFC	Request For Comment
RIP	Routing Information Protocol
RTG	Random Traffic Generator
RTI	Runtime Infrastructure
SPT	Shortest Path Trees
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UML	Unified Modeling Language