THEORY AND HANDS-ON EXERCISES WITH NETWORK SIMULATORS FOR E-LEARNING ON DISTRIBUTED SYSTEMS

Diplomarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Jana Krähenbühl
September 2007

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

Zusammenfassung

E-Learning übernimmt eine stetig wachsende Bedeutung in heutigen Ausbildungs- und Schulsystemen. Des Weiteren bringt ein ausgeprägtes, hohes Fachwissen den entscheidenden Wettbewerbsvorteil in der heutigen Berufswelt. Mit elektronischen Medien im Ausbildungsbereich ist es möglich viele, am Inhalt des Angebotes interessierte Studierende und Auszubildende zu erreichen. Aus diesen Gründen wurde entschieden im Rahmen des Projektes "End-to-End Quality of Service over heterogenous networks" (EuQoS) ein E-Learning Kurssystem zu entwickeln und zu Zwecken der internen Ausbildung und der Wissensverbreitung einzusetzen. Die Hauptziele von EuQoS sind die Erforschung, Integration, Untersuchung und Veranschaulichung von Quality of Service (QoS) Merkmalen in gängigen Netzwerktechnologien.

Das E-Learning Kurssystem, welches im Rahmen von EuQoS aufgebaut wurde, besteht aus verschiedenen Einheiten, die im Internet verteilt jede seine Rolle übernehmen. Einerseits wird die Infrastruktur zur Authentifizierung und Autorisierung von Kursteilnehmern, the Authentication and Authorization Infrastructure (AAI), vom Swiss Academic and Research Network (SWITCH) zur Verfügung gestellt. Andererseits stehen dem Studenten die kommerzielle E-Learning Umgebung WebCT für die Ansicht von theoretischen Inhalten und ein Laborsystem für die praktische Arbeit zur Auswahl bereit.

Die Diplomarbeit beinhaltet zwei grössere Teilbereiche. Einerseits geht es darum im Rahmen von EuQoS ein Kursmodul zu entwickeln. Andererseits benötigt der praktische Teil des Kursmoduls eine Software zur Visualisierung und Analyse von Resultaten, welche im Rahmen von praktischen Übungen erstellt werden.

Im Kursmodul "Implementing Protocols on Network Simulators" (Entwicklung von Protokollen in Netzwerksimulatoren) erhalten die Studenten eine Übersicht über das Konzept von Simulationen im Allgemeinen und Netzwerksimulationen im Besonderen. Netzwerksimulationen unterstützen die Entwicklung und Modellierung von neuen Netzwerkprotokollen und helfen bei der Erweiterung und Verbesserung von bereits existierenden Protokollimplementierungen. Es werden mehrere Netwzerksimulatoren verglichen, Vor- und Nachteile erläutert und gegenübergestellt. Das in der Arbeit erstellte Modul umfasst Theorie zum Netzwerksimulator "Network Simulator (ns-2)", einem der allgemein verbreitetsten Simulatoren im Bereich von Kommunikationsnetzen in Forschung, Entwicklung und Ausbildung. Aufbauend auf diesem Simulator, wird der Student schrittweise in die Arbeitsweise bei der Protokoll-Entwicklung eingeführt. Der theoretische Teil des Kurses umfasst alle nötigen Informationen und Grundlagen, um den zweiten, praktischen Teil erfolgreich zu bewältigen.

Der praktische Teil umfasst vier verschiedene Aufgaben. Die drei ersten Aufgaben bauen gemächlich auf dem in der Theorie vermittelten Wissen auf und gewöhnen den Studenten an die Laborumgebung und die Arbeit mit einem Netzwerksimulator. Die beiden ersten Aufgaben

beinhalten die Implementierung einfacher Netzwerksimulationen mit statischen und mobilen Netzwerkkomponenten. In der dritten Aufgabe soll der Student zur Einführung ein Protokoll mit geringem Programmieraufwand in ns-2 integrieren. Die letzte Aufgabe entspricht der Hauptaufgabe des Kurses und umfasst die Implementierung eines Routing-Protokolles für mobile ad-hoc Netzwerke (MANet).

Im praktischen Teil benötigt das Modul eine Software, die den Analyseprozess unterstützt und die Systembedingungen des Kursportals erfüllt. Da keine Applikation existiert, die diesen Vorgaben und Wünschen entspricht, wurde im Rahmen dieser Arbeit eine Visualisierungs- und Animations-Applikation entwickelt. Die Software "Visualization and Animation for Network Simulations" (VAT4Net) ist erhältlich als Java Applet und unterstützt damit die web-basierte Kursumgebung. Mit Hilfe der Software können Netzwerksimulationen animiert werden. Damit können visuelle Problemzonen, wie Engpässe und Blockierungen in Netzwerkstrukturen festgestellt und lokalisiert werden. Zudem stellt die Applikation statistische Kenndaten von Simulationen zur Verfügung. Der modulare Aufbau des statistischen Teils ermöglicht es, die Software je nach Bedürfnissen zu erweitern und weitere Module zu entwickeln. Das existierende Modul zur Berechnung der "Ende-zu-Ende Verzögerung" (End-to-End Delay) dient hier als Muster für weitere Entwicklungen.

Das Kursmodul wurde im Rahmen einer Informatikvorlesung der Universität Bern getestet und daraufhin angepasst und erweitert. Ein wesentliches Problem, welches sich herauskristallisiert hat, ist die fehlende Kenntnis des Studenten in der Programmiersprache C++. Ohne gute Kenntnisse dieser Programmiersprache, ist es sehr schwierig den praktischen Teil erfolgreich zu absolvieren. Anhand dieser Vorgaben wurde der Theorieteil ausgebaut und die praktischen Übungen mit erweiterten Vorlagen teilweise vereinfacht.
Die Animation von Simulationen stellt hohe Ansprüche an die Software VAT4Net. Ein grundlegendes Problem bei der Implementation der Software war die Grösse der Protokolldateien von Simulationen. Ab einer gewissen Grösse dieser Dateien war es unmöglich, die benötigten Daten komplett einzulesen. Die Software wurde mit einem Vorverarbeitungsschritt erweitert, welcher ressourcenschonendes zeilen-basiertes Einlesen der Daten ermöglicht. Somit lassen sich wesentlich grössere Protokolldateien verarbeiten.

Der Beitrag dieser Arbeit besteht aus dem Kursmodul und VAT4Net ermöglich die Einführung von Studenten in die Thematik von Netzwerksimulationen.

Abstract

E-learning in present-days educational systems takes an important position with a growing influence. In today's business, increasing the own value goes hand in hand with enhancing knowledge. E-learning is an easy way to reach diverse groups of people. Therefore, an e-learning course has been established for internal training and dissemination during the European project "End-to-End Quality of Service support over heterogenous networks" (EuQoS). The key objective of EuQoS is to research, integrate, test, validate and demonstrate end-to-end Quality of Service (QoS).

The EuQoS e-learning course system is a distributed system with different units. On one hand, the authentication and authorization is provided by the Authentication and Authorization Infrastructure (AAI), a infrastructure originated by the Swiss Academic and Research Network (SWITCH). On the other hand, the course is provided either by the course portal WebCT for content deployment or the laboratory portal for practical sessions.

This Diploma thesis contains two main parts. The first part is the work on one EuQoS course module. The second part is the practical part of the module which needs a software to visualize and analyze network simulations. This is contributed within this work.

In the module "Implementing Protocols on Network Simulators", students get a theoretical overview of network simulation concepts. Network simulation is an important basic strategy to develop and prototype new communication protocols as well as to improve existing protocols. The different concepts of network simulations are explained and classified, advantages and disadvantages are discussed. The most common network simulators and details of implementing simulations, especially protocols, in Network Simulator (ns-2) are introduced to the student in the theoretical part. The development of a protocol in ns-2 is described step-by-step. The knowledge to successfully pass the hands-on exercises is acquired in this theory chapters.

The practical work in the module is divided into four different hands-on sessions. Three out of four hands-on session are constructive exercises to get familiarized with the network simulator and the implementation methods presented in the theoretical part. In the first two exercises the student has to provide a wired and a wireless network simulation. In the third exercise the implementation of an easy protocol with little complexity is demanded. The fourth exercise represents the major task in the practical part of the module and includes the implementation of a routing protocol for Mobile Ad-hoc Networks (MANet).

For the module "Implementing Protocols on Network Simulators" inside the EuQoS course portal a tool for visualizing and animating simulations is required. The application "Visualization and Animation Tool for Network Simulations" (VAT4Net) must fulfil the system requirements of the EuQoS course.

For the web-based course infrastructure VAT4Net is available as a Java Applet. Thus, the application is designed to support the analysis step in a network simulation process. While animating a simulation, some problems and behaviors of the simulated environment are visualized and could be recognized, such as bottlenecks, packet drops, congestions, network structure, movement behavior of wireless nodes, etc. A second focus of VAT4Net is the processing of statistical representation, for example end-to-end delay, packet loss rate and other Quality of Service characteristics. The implementation of the end-to-end delay serves as an example for further module development.

The contribution of this work, including the course module and the software VAT4Net, enables the student to acquire knowledge in the network simulation topic.

Acknowledgement

I would like to thank Prof. Dr. Torsten Braun who gave me the opportunity to work out my Diploma thesis in his research group Computer Networks and Distributed Systems. My special thanks go to Thomas Bernoulli and Thomas Staub for supervising this work and providing the resources and material needed for my Diploma thesis.

Further thanks go to all people for their patience, encouragement and support, especially my family, my boyfriend, my colleagues and my friends.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

E-learning in present-days educational systems takes an important position with a growing influence. In today's business, increasing the own value goes hand in hand with enhancing knowledge. Therefore, an e-learning course has been established for internal training and dissemination during the European project "End-to-End Quality of Service support over heterogenous networks" (EuQoS). In the context of this e-learning course this Diploma thesis provides a module on network simulation. During the thesis theoretical content, hands-on exercises and supporting software tools have been developed.

EuQoS is a project of the 6th Framework Program of the European Union. The Computer Networks and Distributed Systems research group of the University of Bern acts as a partner in different fields of activity. The Diploma thesis is part of the training and dissemination activity and covers one e-learning module.

Chapter 2 briefly introduces the general terminology of e-learning and discusses some characteristics and statistics of e-learning in Europe. Chapter 3 and 4 show an overview of the whole course system, beginning with the EuQoS e-learning project coverage leading to the implementation details of the course system.The two main chapters of this thesis, Chapter 5 and 7, focus on the implementation of the supporting tool "Visualization and Animation Tool for Network Simulations" (VAT4Net) and the content of the course module. The Chapter 6 covers the laboratory system.

The animation of a simulation in VAT4Net can help to understand and to discover different behaviors of the simulated networks. Further, VAT4Net provides a plugin engine, which enables to process statistical data. The functionality and implementation of VAT4Net is described in Chapter 5.

The course module provides basic knowledge on network simulation in research and development. Network simulation is an important basic principle to develop and prototype new communication protocols as well as to improve existing protocols. It is therefore important to teach students basic knowledge on network simulation. Chapter 7 shows the module content as well as the practical exercises for the students.

Chapter 2

E-Learning

e-learning is a recurrent theme within the whole Diploma thesis. All work done for the purpose of the thesis is in fact a part of an e-learning system. The term e-learning will be briefly defined and a short statistical analysis will be made.

2.1 Terminology

e-learning (short for electronic learning) is a term used for all electronic media-based education, which utilizes different resources to acquire and communicate skills and knowledge. Online education is an extension of the traditional form of e-learning or distance learning. Typically, "it involves

- the use of the Internet to access learning materials,
- to interact with the content, instructor, and other learners,
- and to obtain support during the learning process, in order to acquire knowledge, to construct personal meaning, and to grow from the learning experience", [1].

Basically, e-learning can be divided into two types, synchronous and asynchronous e-learning.

Synchronous e-learning can be described as a virtual classroom, with a simultaneous, virtual presence of learners and its instructors. Achieved by different collaborative methods and technologies as video and audio conferences with streaming technologies, application sharing, real-time chat, Internet-based telephony or instant messaging, it becomes possible to impart knowledge in terms of lectures and tutorials, hold exercise and practice sessions and take exams. Besides real-time interaction between experts and their students synchronous e-learning benefits from location and distance independence, no physical attendance, convenient learning and teaching in best place for the participants, uniformity, reusability and repeatability of educational content.

While synchronous e-learning is only location independent, asynchronous e-learning adds time independence. It offers the possibility of accessing self-paced courses and enables intermittent access to studies, discussion groups and exercises. Some main methods of asynchronous e-learning can be listed as follows: online course material in form of electronic documents, multimedia material as video, animations, graphics or audio streams, simulations of real-life circumstances, discussion forums, quizzes, further readings, etc. Unlike synchronous e-learning where Internet technologies are a fundamental requirement for interacting in real-time, asynchronous e-learning can be based on CD or DVD mediums. Advantages of asynchronous e-learning are: learning and tutoring at anytime from anywhere, no access time restrictions, updated course material, enhanced group work, file sharing among students and including benefits of synchronous e-learning.

Like every other teaching method, e-learning holds disadvantages and risks in its preparation and use:

- Internet broadband access is a main requirement to attend video and audio conferencing.
- The use of new technologies to publish and broadcast educational content needs new consolidated knowledge and high cost investments.
- ”Personnel and operational expenses can hardly be reduced”, [2].

There are also a lot of new requirements the student has to fulfill. The learner has to

- develop a changed self-consciousness in self-governed and self-responsible learning,
- know the own learning pattern, learning behavior and individually appropriate learning strategy,
- and ”know as many as possible learning media and learning ways and is able to use them”, [3].

Generally, the risks involved in e-learning are

- growing lack of verbal communication abilities,
- possible lost of real life,
- and identification and authorization problems.

While the Internet is evolving, independent of the risks and disadvantages mentioned above, the same potential is assigned to e-learning, as earlier to e-commerce or e-business.

2.2 Research and Studies about E-Learning

Since 2000, the Economist Intelligence Unit, the business arm of The Economist Group, has published an annual e-readiness ranking of the world's 60 largest economies. In the year 2003 the Unit has done a complementary ranking: e-learning readiness in the same 60 countries.

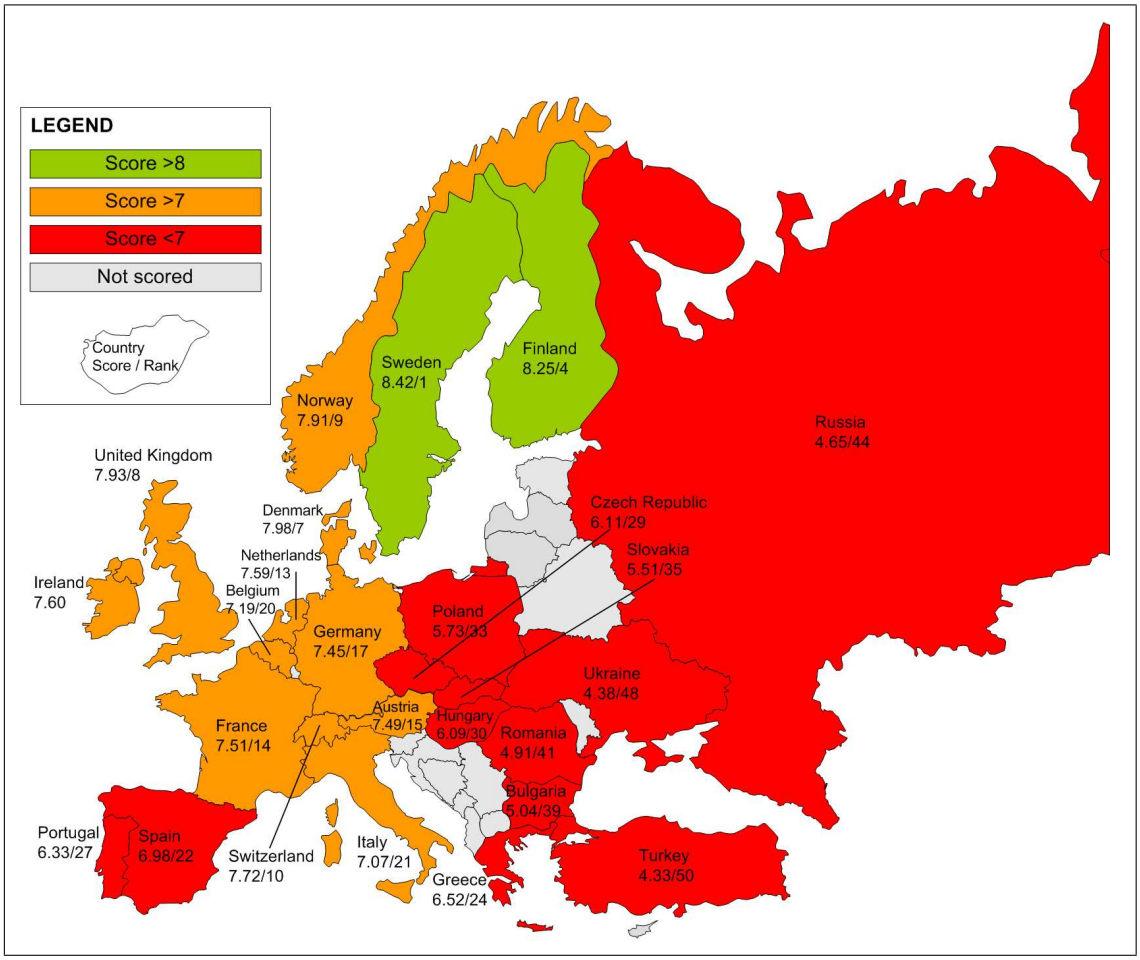


Figure 2.1: e-learning readiness in Europe in 2003 - scores and ranking for each state, [4].

The ranking criteria and score calculation can be found in detail in the white paper "The 2003 e-learning readiness rankings", [4]. Basically there are four categories of criteria, the quality and expansion of the Internet (connectivity), the ability to deliver and consume e-larning (capability), the quality of online learning material (content) and the support of official committees and government (culture).

There are big regional differences in the score (and ranking) in Europe (see Figure 2.1). It is remarkable that these regional differences corresponds with historical and political

Country	Score (of 10)	Rank(of 60)	Country	Score (of 10)	Rank(of 60)
Canada	8.40	2	**Mexico**	5.96	31
US	8.37	3	**Brazil**	5.63	34
South Korea	8.24	5	**South Africa**	4.96	40
Singapore	8.00	6	**Saudi Arabia**	4.50	47
Australia	7.71	11	**Egypt**	3.98	51
Taiwan	7.47	16	**Vietnam**	3.32	57
Japan	6.53	23	**Pakistan**	3.22	58
Malaysia	6.48	25	**Iran**	3.06	59
Chile	6.13	28	**Nigeria**	2.82	60

Table 2.1: e-learning readiness worldwide in 2003, [4].

circumstances in each country or area. The so-called ”rich” countries are mostly scored higher than 7. These countries have an economic background which enables a high degree of IT diffusion, a strong education system and a free market. Thus the score of Germany, France, UK, Switzerland or Denmark can be explained. What is striking, is the fact, that all Northern European countries, like Norway, Sweden, Finland and Denmark has a significant higher score than the others. This circumstance is explained in the paper [4] as follows:

- An unequaled IT infrastructure,
- a citizenship that is eager to integrate the Internet into all facets of daily life,
- a top-notch education system,
- the fact, that government has helped set the stage,
- and a region which is saturated with mobile phones and broadband connections.

On the other hand, Southern Europe is much less enthusiastic in using information technology, while the backlog demand of Eastern Europe can be explained through its historical and political background. On the turn of the millennium the whole eastern region was in a state of massive political change after the collapse of the communist system in 1991. In Table 2.1 some worldwide countries and its values are listed.

Chapter 3

EuQoS - End-to-end Quality of Service Support over Heterogeneous Networks

Within the scope of the Diploma thesis, a module in the e-learning environment of the End-to-end Quality of Service Support over Heterogeneous Networks (EuQoS project [5]) was developed and implemented. In a second step, the Visualization and Animation Tool for Network Simulations (VAT4Net), a part of the Internet Remote Network Simulation Infrastructure (IRNSI), was designed as a tool supporting the practical training in line with the EuQoS e-learning environment.

The e-learning modules concentrate on conceptual principles of networking in QoS related areas and describe the main topics with established protocols and applications of today's Internet and present the solutions, concepts, architectures and protocols provided by the EuQoS project. The contents are designed according to certain didactical guidelines (Chapter 4.3.1). The modules will be offered via an e-learning environment consisting of a set of tools and (multimedia) applications (EuQoS deliverables, [6]).

A short summary on each module based on the introductory chapters follows:

1. Implementing Protocols on Network Simulators
In this module students get a theoretical overview of network simulation concepts. The different concepts are explained and classified. Advantages and disadvantages are discussed. The details of the module will be described in Chapter 7.

2. Applications and QoS
This module gives the students the basic knowledge about QoS. The module first explains the parameters which characterize a network connection and which different parts of the network influence them. Then it looks at the different service-level requirements the various applications require. At the end of the module different packet scheduling algorithms and the today available QoS architectures are explained.

3. Network Emulation in General and with a Focus on Satellite and QoS
In this module the student will learn to apply network emulation for research or professional use. First the module gives an general overview of frequently used emulation systems and their concepts. It further focuses on the FreeBSD/Dummynet, which is a free and simple network emulator. Finally an emulation for satellite communications is shown as a case study to harden the learned skills of FreeBSD/Dummynet.

4. Traffic Engineering
This module describes the technical problems of mapping traffic flows onto an existing physical topology, following certain policies. It describes MultiProtocol Label Switching (MPLS) as a solution for traffic engineering. The module focuses on the description of information distribution, path calculation and the resource reservation in MPLS networks.

5. Signaling (SIP, NSIS, COPS)
This module focuses on the signaling protocols used to establish sessions between several parties and to provide the required QoS for them. First the Session Initiation Protocol (SIP) is explained. Then the work of the IETF Next Step In Signalling (NSIS) working group is explained. Their first use case is to standardizing an IP QoS signalling protocol. At the end of the module the Common Open Policy Service (COPS) Protocol for exchanging policy and configuration information between a network policy server and a set of clients is explained.

6. Enhanced Transport Protocols (API and Integration)
This course module allows students to understand that traditional and new generations of transport protocols have not been specifically designed to offer QoS oriented services, but that a transport protocol is able to provide a QoS oriented end-to-end service, resulting from the optimization of the Best-Effort service. Finally the module gives a closer look at the new ETP (Enhanced Transport Protocol), and the different QoS-aware transport protocol mechanisms used.

7. Monitoring and measurement Systems in IP-based Networks
The module first provides a brief overview of the role of monitoring and measurement systems (MMS) in modern IP networks. Then it looks closer to what could be measured and how this could be actually done. Then it explains where and when this measurements are performed. Finally the module describes several tools, that are aimed to measuring the most important QoS metrics.

8. EuQoS Overview
The module provides an overview on the EuQoS project by presenting the global EuQoS architecture and its corresponding high-level behavior. It will show how the architecture is able to handle all main present network technologies and is able to provide a global coherent architecture, which homogeneously integrates all network layers and their related QoS protocols.

9. EuQoS Intra-domain Resource Management
This modules describes the problems and the concepts of resource management in heterogenous networks. The module focuses mainly on the connection admission control (CAC), which is a key resource management problem. The module describes how CAC works for inter- and intra domains to provide QoS.

10. EuQoS Multicast Middleware
This module explains the theoretical basics of multicast technologies first. Furthermore, a brief overview of P2P (Peer-to-Peer) systems is shown. It describes the application layer multicast paradigm and compares it to the native multicast of the network layer. Finally, a special implementation of application layer multicast is described, which acts transparently for native multicast applications.

11. QoS Provisioning Process
This module describes the provisioning process in EuQoS to build end-to-end QoS paths (EQ-paths) across multiple Autonomous Systems (AS). First, it describes the Traffic Engineering and Resource Optimization (TERO) module, which is responsible to build the EQ-paths in the best possible way. Then, it describes the inter-domain QoS routing protocol, named Enhanced QoS Border Gateway Protocol (EQ-BGP) and how TERO interacts with it.

12. EuQoS Signaling
This module provides an overview of the signalling architecture, mechanisms and protocols used in the scope of the EuQoS system. The module gives overview on how they can be combined in order to negotiate and to provide the required QoS levels to applications in inter-domain, heterogeneous scenarios.

Chapter 4

The E-Learning Course System

4.1 Introduction

The e-learning course system is a distributed system with different entities (see Figure 4.6) playing important roles in various areas with different assignments. The system is based on the standard e-learning infrastructure used within different e-learning portals as OSLab [7] or VITELS [8] which is adapted for the EuQoS course system.

On one hand, the authentication and authorization is provideds by the Authentication and Authorization Infrastructure (AAI), a infrastructure originated by the Swiss Academic and Research Network (SWITCH, [9]). The AAI portal is the front end for AAI single sign login (Figure 4.6, Part 3). On the other hand, the course is provided either by the course portal WebCT (Figure 4.6, Part 2) for content deployment or the laboratory portal (Figure 4.6, Part 4 and 5) for practical sessions, whereas each module can have its own laboratory portal. The details on the different entities are described in the next sections. Especially, the laboratory is explained in Chapter 6, because of its special impact for this Diploma thesis.

4.2 Technology

4.2.1 WebCT - Web Course Tool

The course platform provides the learning content, the quizzes, the logbook for the student's notes, and the discussion board of the EuQoS courses. For this purpose, the courses operate on the commercial course platform WebCT CE and in the future on WebCT Vista.

WebCT works similar to a Content Management System (CMS), which enables storing, controlling, versioning and publishing content on a web site. As an e-learning course platform WebCT contains some additional features, like quiz generating, student account management, tutoring functionality such as chat and discussion application and assessment.

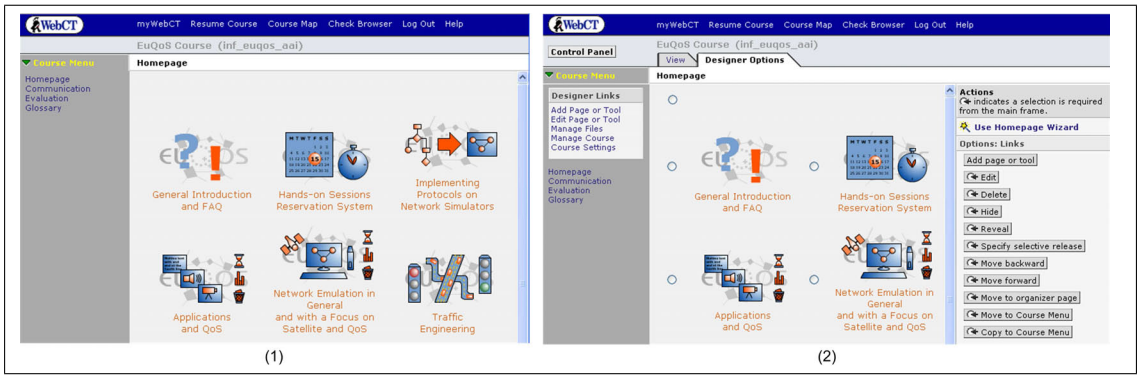


Figure 4.1: WebCT - (1) EuQoS entry page student's point of view, (2) Designer's view with editing options

4.2.2 AAI - Authentication and Authorization Infrastructure

The authentication and authorization process for the e-learning portal is deployed by SWITCH's Authentication and Authorization Infrastructure (AAI). The main objectives of AAI is to simplify inter-organizational access to web resources. Most of Swiss universities use the AAI infrastructure to allow simple access to different entities, such as web mail, e-learning, research databases or student admin pages, with a single sign on. For the EuQoS project a new virtual home organization was built up in order to include users of the EuQoS e-learning portal into the SWITCH federation.

The AAI is a framework based on Shibboleth [10], an Internet2 middle-ware. The web-based AAI Portal (see Fig. 4.6, 3) is the front end implementation of AAI. The learner can subscribe to the course after login with its AAI identification. After the tutor has granted access to the selected course, the participant is then able to pass through the AAI portal into the course system or the laboratory computers (see status column in Figure 4.2). The AAI portal therefore forwards the user to the course system and further creates and updates the user in the resource management system by writing to a web service.

4.2.3 The Resource Management Server

The resource management server implemented for the system acts as a booking engine for time-slots on the available laboratory "seats" and access is granted by AAI single sign login for the students. The resource management server is not only used for hands-on sessions in the IRNSI environment but for all time shared laboratory setups.

In Figure 4.3 a typical view of the resource management server is depicted. The laboratory for the module "Implementing Protocols on Network Simulators" has two seats available and could be booked 24 hours a day for time-slots of one hour.

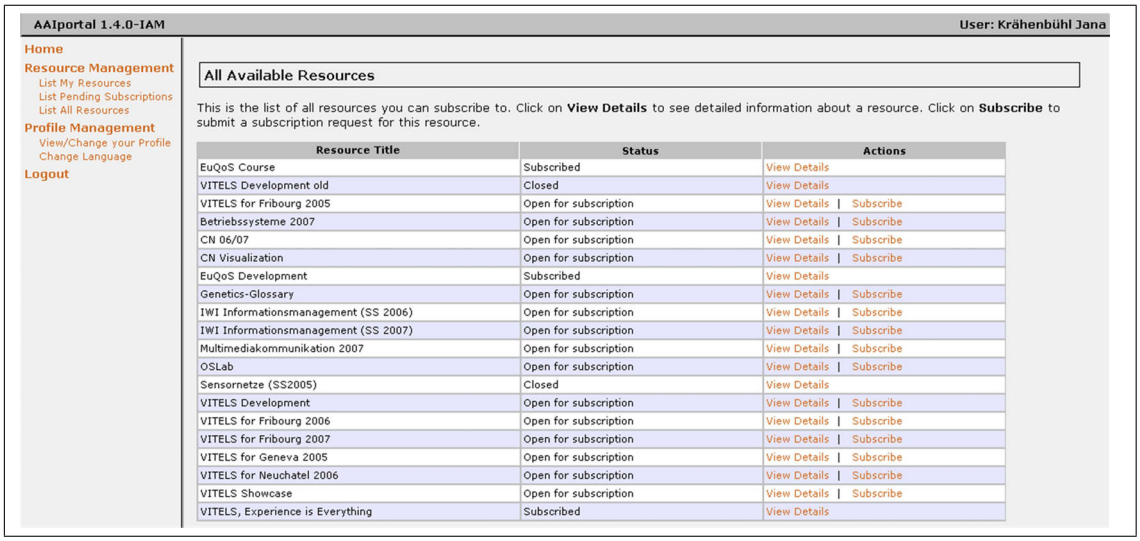


Figure 4.2: AAI portal overview of registered and available courses (Example is taken from the AAI Portal of the University of Bern)

From administrator's point of view, there is surely extended managing functionality available like user, module or lab-portal management.

4.2.4 The Laboratory Portal Server

The laboratory portal server manages the access to the laboratory seats by verifying user's authorization and authentication with AAI and checking up on the resource management server, if a time-slot is reserved by this user. If the reservation is missing, the user is redirected automatically to the resource management server's front end.

From the entry page of the laboratory portal server (see Figure 4.4) all required applications are accessible. In each user's time-slot the state of the laboratory remains the same until the user is deleted from the system or he resets the system. The reset function restores the initial state of the laboratory. All data is lost, but this is a way out of an impasse situation if the user is totally lost. Further, Figure 4.4 shows the module specific functions. In this example, on the left side you can find the entry to the network simulator ns-2 and on the right side the "Visualization and Animation Tool for Network Simulations" (VAT4Net, see Chapter 5) can be start up as an applet application. The details on the exact laboratory functions follow in the next section.

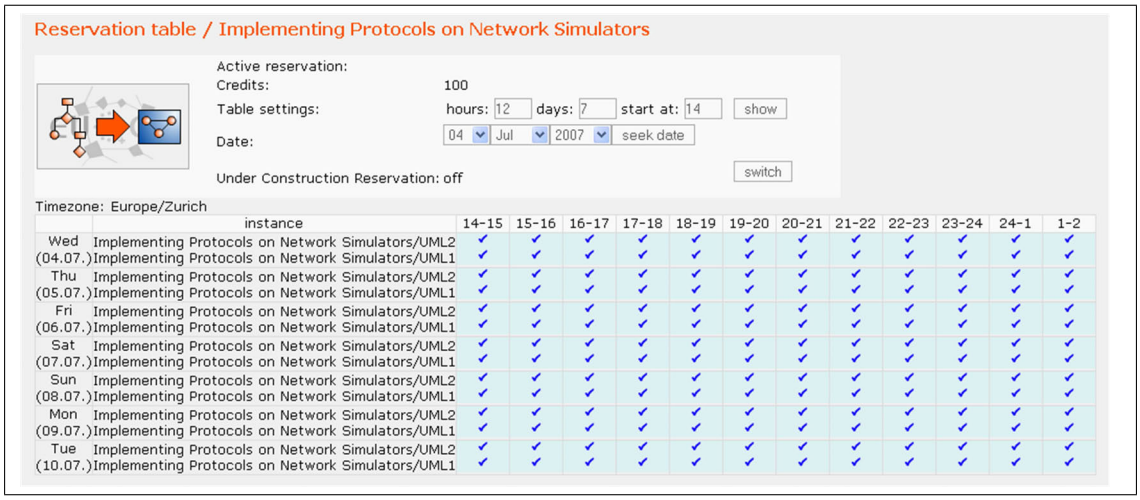


Figure 4.3: The Resource Management Server - Booking front end where students can book laboratory seats for preferred time-slots.

4.3 Didactics

4.3.1 The EuQoS Didactics and Design Guide

The Didactics and Design Guide [11] acts as a cookbook-like documentation of the full didactic and design concept. It is a handbook which affords the various authors of modules within the e-learning environment of the EuQoS project to develop a module with:

- comparable build and intelligent chain of chapters in each module,
- well formatted and consistent appearance,
- assigned time-frame and expected time spent by students for each module,
- forced knowledge application and exploration in hands-on session,
- adequate grading methods of work performed by students.

To facilitate co-operation within module development, a small utility, the file framework generator and formatter (ffgf) is provided and described in the next section.

4.3.2 ffgf - Automatic Content Generation for WebCT

The file framework generator and formatter (ffgf) [12], a Perl based tool, simplifies the content generation and standardizes the look and feel of content supplied by different developers. It supports the module producer in generating the content according to the Didactics and Design Guide [11]. It further decouples the module content from a specific e-learning platform.

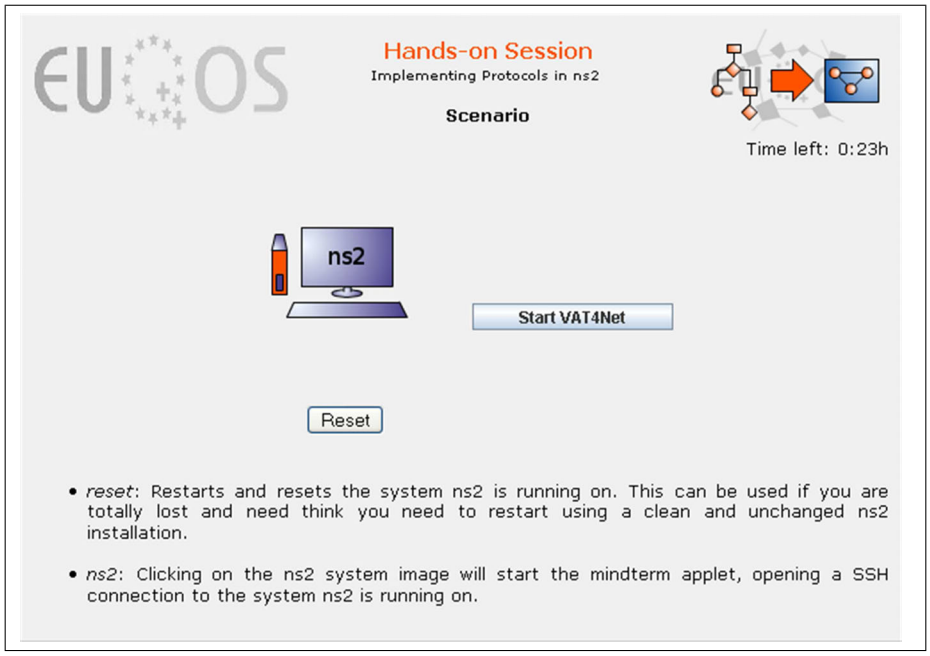


Figure 4.4: The Laboratory Portal Server - Entry Page to the Laboratory, example taken from the module "Implementing Protocols on Network Simulators".

The module producer starts with a table of contents (see Figure 4.5, step 1). ffgf translates this text file into a numbered and formatted set of files (see Fig. 4.5, step 2). ffgf offers a possibility to generate and edit the module's content (Figure 4.5, step 3) without any hard knowledge of HTML and layout concerning details of the e-learning platform. All predefined pages, such as table of contents, welcome pages, schedules or quiz introductions are generated automatically and filled with predefined content (Figure 4.5, step 2) which is dedicated from the EuQoS Design Guide and built into ffgf. Each module's writer has only to insert his own module specific text in basic HTML format, which can be done with an "What You See Is What You Get" (WYSIWYG) editor (Figure 4.5, step 3). In step 4 ffgf generates specific html files for different e-learning platforms (WebCT CE, WebCT Vista, etc.) out of the user edited source files from step 3. These files have to be deployed in the course system, in our case WebCT CE (Figure 4.5, step 5).

4.3.3 Requirements and Handling from the Student's Point of View

In Figure 4.6 student activities are shown in cooperation with the course system. In a first step, the student subscribes to a course on the AAI Portal via a web browser (Figure 4.6, part 1 and 3 are involved). The owner of the course then grants or denies access to the course (Figure 4.6, part 2-4). In case of granted access, the system registers the user to the course platform and resource management server. The user may now access the course platform or making a

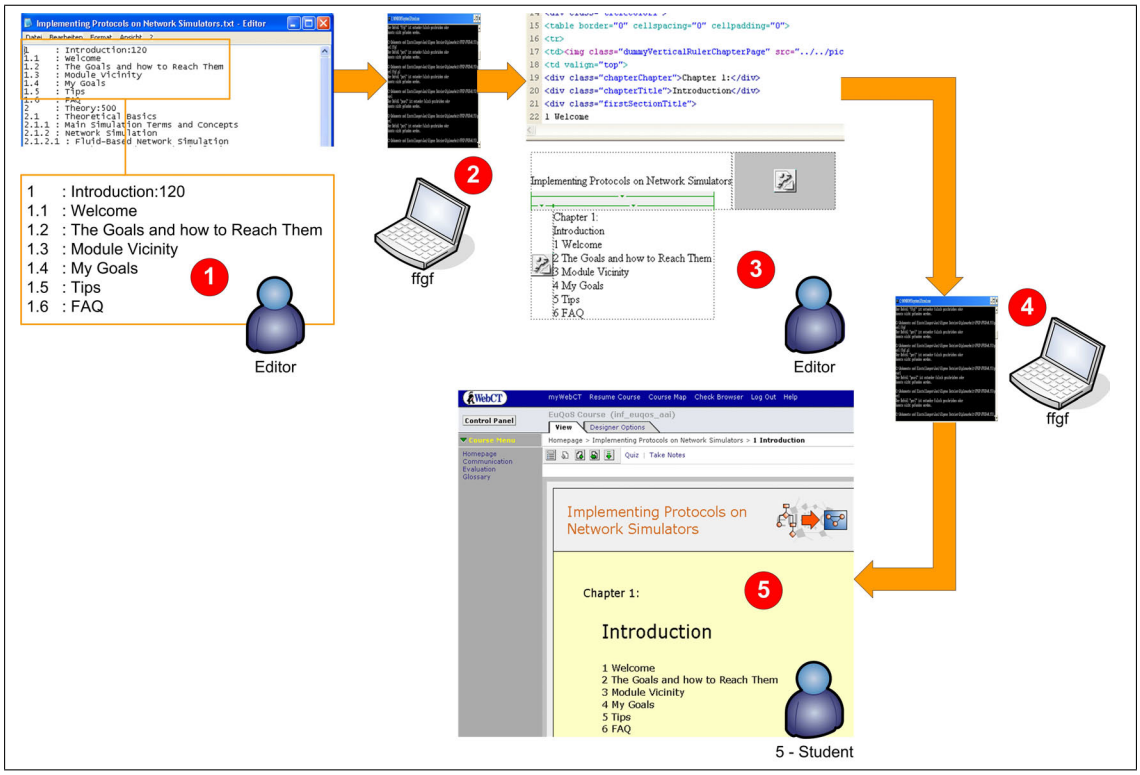
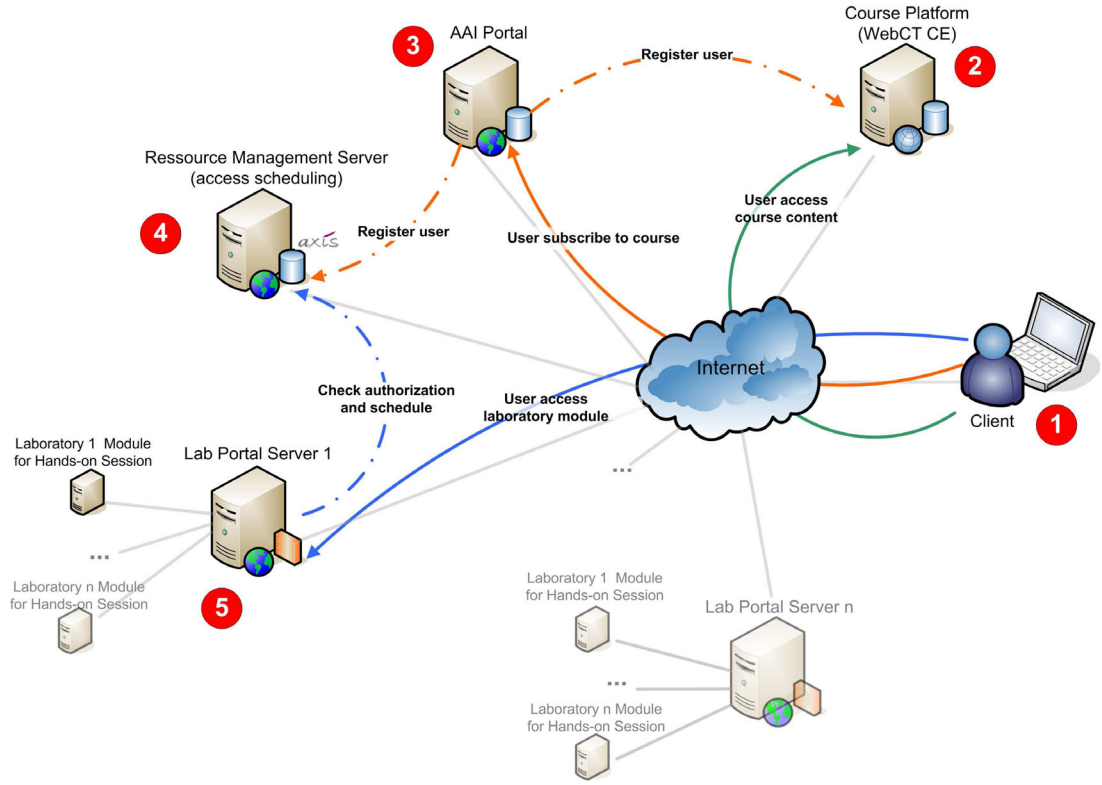


Figure 4.5: Content generation with ffgf - steps from the table of contents up to the completed content of the module.

reservation for a laboratory system (Figure 4.6, part 1,2,4).

He can read content, pass quizzes from the course platform or make hands-on sessions. Whereas the course platform is available 24 hours a day, the laboratory (Figure 4.6, part 5) system may be a shared resource and the user has to make a reservation first. In order to access the course content, a web-browser and in some cases a pdf reader is required. When working on the laboratory there are some further requirements an accessing computer has to fulfill. Requirements for accessing the complete course content are: web browser, pdf reader, flash player and support for Java applets(JVM).



Accessing the e-learning course platform

1. The **Client** either likes to work on theoretical content of the e-learning course or he likes to access the laboratory of the module.
2. The **Course Portal** (WebCT CE) provides several e-learning tools such as content, exercises, forum, etc.
3. The **AAI Portal** is the front end implementation of AAI, the Authentication and Authorization Infrastructure.
4. The **Resource Management Server** acts as a booking engine for the laboratory seats.
5. The **Laboratory Portal Server** manages the access to the laboratory sats and provides different communication methods.

Figure 4.6: System architecture of the e-learning portal with different entities and its collaboration.

Chapter 5

VAT4Net - a Visualization and Animation Tool for Network Simulations

5.1 Introduction

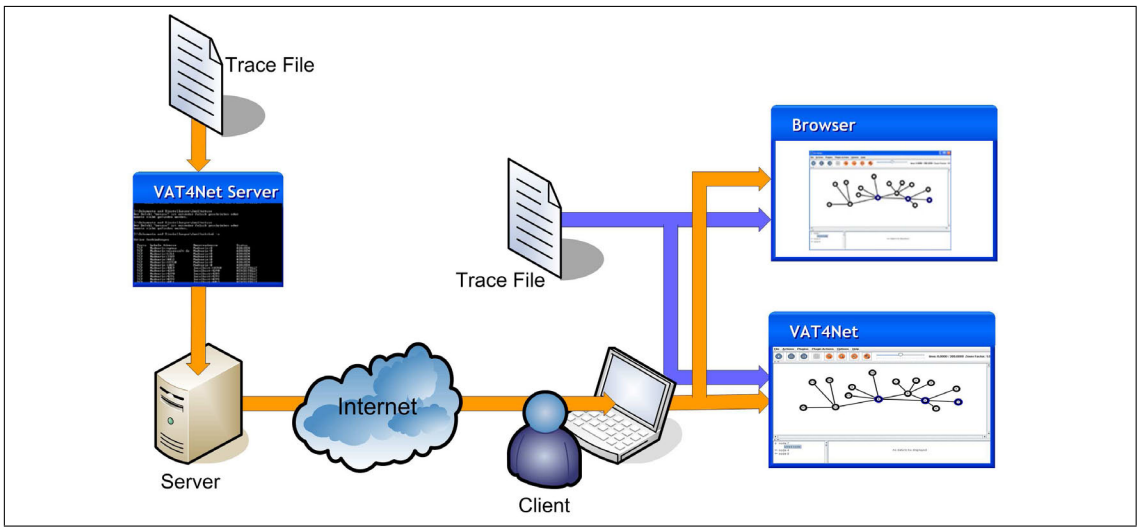


Figure 5.1: VAT4Net System Architecture

The Visualization and Animation Tool for Network Simulations (VAT4Net) has been developed during this thesis to support the analysis step in a network simulation process. It enables the user to visualize gathered trace files from network simulations. While animating a simulation, some problems and behaviors of the simulated environment are visualized and could be recognized, such as bottlenecks, packet drops, congestions, network structure, movement behavior of wireless nodes, etc. A second focus of VAT4Net is the processing of statistical (data and its graphics) representation, e.g. end-to-end delay, packet loss rate and other Quality of Service characteristics.

For the module ”Implementing Protocols on Network Simulators” inside the EuQoS course portal a tool for visualizing and analyzing simulations is required. The tool must fulfil the system requirements of the EuQoS course. While the Network Animator (NAM) [13] is platform dependent as ns-2 and is not applicable into the course infrastructure, VAT4Net is implemented in Java and is available as a Java Applet for the web-based infrastructure. However, outside of the course portal VAT4Net is available as a stand-alone software. In this context, VAT4Net combines two different architectures - the client-server (Fig. 5.1 - orange arrow) and the monolithic (Fig. 5.1 - blue arrow) design.

The basic data of all visualization and animation processes is provided by trace files generated from a simulation run in ns-2. The current version of VAT4Net only focuses on trace files in NAM trace format (see Section 7.4.3), but can be extended easily for other simulation trace file formats.

5.2 Architecture, Data Preparation and Transfer

This chapter describes the architecture, the data preparation and data transfer, while the Chapter 5.3 shows the details on animation and visualization and its implementation from the user perspective. Certain background processes are necessary in VAT4Net to animate and visualize the simulated scenario. More details are given in the next sections.

In Section 5.2.1 the different system architectures used by VAT4Net are described. When analyzing a trace file for the first time the preprocessor (Section 5.2.2) handles the file in the initializing process. As soon as the preprocessed trace file is available and either submitted (Section 5.3.1) or loaded (Section 5.2.3), the animation (Section 5.2.4) and analysis and plugin (Section 5.3.2) control engine is executed.

5.2.1 System Architecture

Applet or Stand-alone Application?

VAT4Net runs in either stand-alone or applet mode. An applet is an application written in Java that can be hosted on a website. When viewing a page containing an applet, the Java technology-enabled browser downloads the applet’s bytecode to the client system an executes it by the browser’s Java Virtual Machine (JVM) [14]. In general, some functions are restricted as reading and writing files, setup network connections, reading system properties, etc., while the JVM is running the applet under different security system. With a digital security certificate it is possible to sign the applet and to enable the necessary functions. When deploying the applet to the webserver, the codebase of the applet has to be packed by applying a digital certificate.

In stand-alone mode VAT4Net is started up from a single Java Archive (JAR) file [14] and interprets the compiled source code in the JVM. There are less restrictions when VAT4Net is started from the local computer than when using it as an applet.

If the applet is signed and the user has granted additional rights, the difference of a signed applet and the stand-alone application is small.

Implementation Details

net.sf.vat4net.Main: In order to start the application in applet mode the *init* method is invoked and applet parameters are read from the website where VAT4Net is hosted. Otherwise the application is started over the standard *main* method.

Server-Client or Monolithic Architecture?

When opening a trace file from a local file system or from a remote server, the appropriate operation mode is started. If the trace file exists on the local computer and is opened by the application, VAT4Net is running as a monolithic software. All processes, threads and operations are performed on the local computer system.

On the other hand, when selecting a trace file from a remote source, the VAT4Net client first tries to connect to the VAT4Net server component on the remote machine. Then the preprocessing (see 5.2.2) and statistic calculation tasks are performed on the remote computer and only the resulting data is submitted to the client over the network (Internet, LAN).

Implementation Details

The different ways of executing VAT4Net are selected when opening a trace file either from the server or the local file system via the menu command ”Load Trace File...” or ”Open Trace File...” (see Figure 5.7). The detailed processes are described in the further chapters.

5.2.2 Preprocessing Trace Files

The fact that event-based network simulations trace events step-by-step requires preprocessing of trace files generated in NAM trace format (see 7.4.3). In consequence of this step-by-step tracing, events of one packet are not grouped together, but are distributed over the trace file (see 5.2, trace lines 585-587,618 of ns-2 trace file) and are merged with events of other packets. In Figure 5.2 an extract of the input file of a typical simulation in the NAM trace format is shown. The unequally distributed events in the example make animation of network simulation difficult. In order to animate a packet on a link the hop event (see Figure 5.2 - line 587) and the receive event (see Figure 5.2 - line 618) or even a drop event should be known. Therefore, it is essential to be aware of line 618 when starting to animate the event in line 587. A pre-parsing of the file is required and under certain conditions it affects the whole trace file. In this context, a look-ahead in the trace file instead of preprocessing is proposed by [15]. A look-ahead implementation was not applicable to VAT4Net as of the implemented streaming interface is limited by the internet

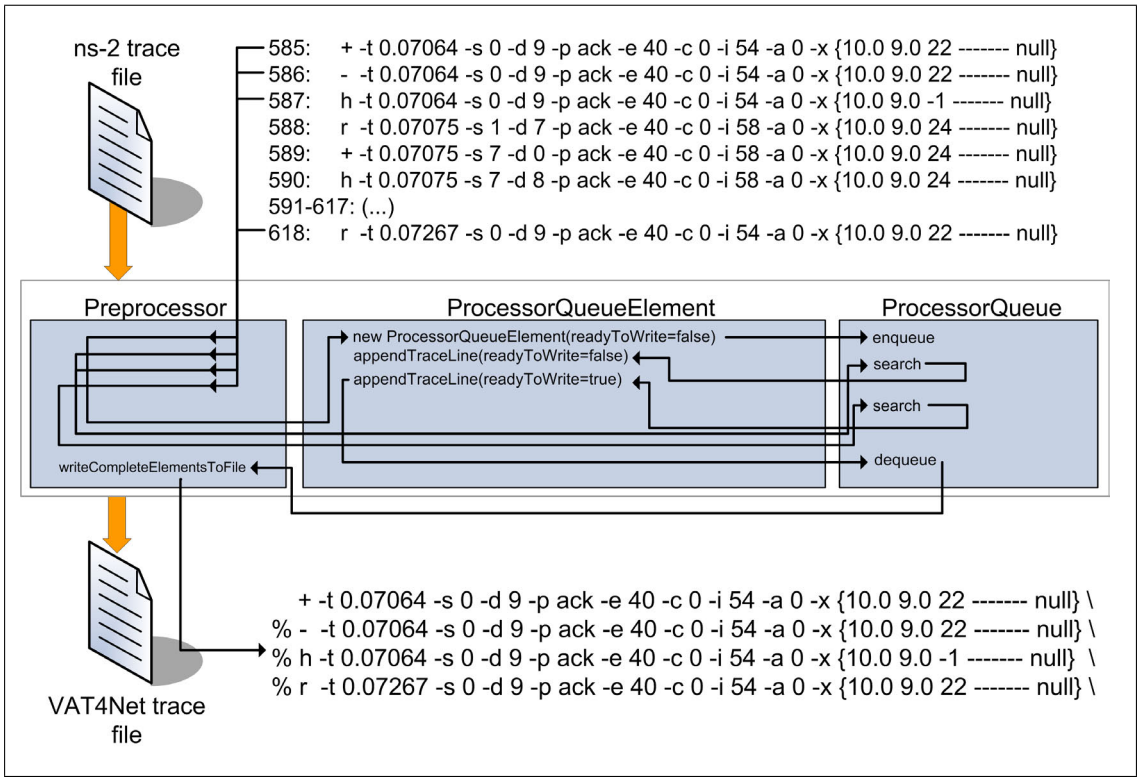


Figure 5.2: VAT4Net Preprocessor

connection bandwidth and excessive memory consumption of the Java application when holding big parts of the trace file in the heap. The main idea of preprocessing trace files is to sort matching events. In case of the example, it would be lines 585,586,587,618.

A new trace file format is introduced at this point:

***.v4n - VAT4Net trace file format - preprocessed and optimized for animation purposes**

The VAT4Net trace file format is adapted from the NAM trace file format. Matching events are combined in one line. Figure 5.2 shows one line of the output file in VAT4Net trace file format. It marks one node-to-node hop event.

The new VAT4Net trace file format enables streaming of data for animation of network simulations. For animating packet flows over nodes, queues and links there is no need of additional lines in succeeding parts of the trace file. At a specified time all necessary data is available for animation in the newly generated trace file.

Implementation Details

Either VAT4Net server or the client takes on the preprocessing, depends on the location of the original trace file. As output the preprocessor generates the new, streaming enabled VAT4Net trace file usable by VAT4Net itself.
The task of preprocessing is implemented using three classes described below:

net.sf.vat4net.io.Preprocessor[A.21]: The preprocessor gets executed as a separate thread and implements the main preprocessing algorithm. In a first step the whole basic NAM trace file is parsed and a new net.sf.vat4net.io.ProcessorQueue is filled (Listing 5.1). As soon as a trace line is completed (Figure 5.2), it can be dequeued from the ProcessorQueue and written out into a temporary file. In a second run the not-processed, still available elements in the ProcessorQueue from the first run are preprocessed (see A.21 for complete file). The temporary file from the first step and the processed lines from the second step are merged and compose to the new trace file in the VAT4Net trace file format.

```
47  public void processLine(String line) throws IOException{
49      parseLine(line,1,queue);
50      (...)
51      while(!queue.isEmpty()){
52          if(queue.hasReadyElement())
53          {
54              q = queue.dequeue(); //dequeue an element
55              //Write all Elements which are in state "ready" to the
                    new file
56              (...)
57          }
58          else
59          {
60              //There is a first element which cannot be dequeued - if
                    queue is full, manage the queue otherwise take a next
                     line from the trace file
61              if(queue.isFull())
62              {
63                  //Write elements which are not yet ready and are
                        blocking the queue out to a temporary list. Go on
                        with elements that are handable
64                  (...)
65              }
66              break;
67          }
68      }
70  }
```

Listing 5.1: net.sf.vat4net.io.Preprocessor, Part of A.21

net.sf.vat4net.io.ProcessorQueue [A.22]: The ProcessorQueue class implements the abstract data structure queue in the First-In-First-Out principle with the main methods enqueue and dequeue. The entities to be hold in this queue are the ProcessorQueueElement instances. A further functionality of this class is the implementation of a search method , which let the preprocessor find the adequate queue elements to complete them subsequently.

net.sf.vat4net.io.ProcessorQueueElement[A.23]: This class is responsible for each individual simulation event. Events not related to packet events are treated line-by-line from the original trace file and are marked as ready ProcessorQueueElement. Events related to packet events are collected until all information is available for animation purposes and afterwards are marked as ready ProcessorQueueElements.

5.2.3 Loading Data from a Local Source

When VAT4Net runs as a monolithic system on a local computer, the data required by animation and visualization operations is loaded in a separate thread from the newly generated or already existing VAT4Net trace file accessing the local file system with the standard package java.io.*. The read-in operation is done line by line. This is possible due to the fact that each line marks one animation event. The read-in operation is used for handling either animation or statistic data.

Handling Animation Data

Each read-in line is put to a buffer, until the buffer is full. The net.sf.vat4net.io.Buffer is implemented as a java.util.LinkedList and has put and get methods (Listing A.24) to handle the animation events.

Once there is content in the buffer, the animation control engine can take up its work (see Section 5.3.1). As soon as trace events from the buffer are consumed by the engine, further trace events are read in until the last line of the simulation trace file is available in the buffer.

The **size of the buffer** is an important part concerning memory consumption and time complexity relating problems. In Chapter 5.4 this aspect will be discussed in detail.

Implementation Detail

net.sf.vat4net.io.FileBufferLoader: Inherits from the class BufferLoader the new buffer instance. The newly started thread reads line-per-line into the buffer instance with the help of the TraceFileReader. When the buffer is full, the thread is waiting until the buffer has more space left and continues then to fill trace lines into the Buffer instance. This is going on until the whole file is processed.

net.sf.vat4net.gui.io.TraceFileReader: Uses standard java.io.BufferedReader to read in the trace file line-by-line. By providing the method *readLine()*, the functionality of FileBufferLoader is enabled.

net.sf.vat4net.io.BufferLoader: Runnable superclass which instantiates a new net.sf.vat4net.io.Buffer.

Handling Statistic and Visualizing Data

Accessing data for statistic purposes works equally as when handling animation data. For statistic data handling the trace file is the basis of all calculation processes. The data is loaded by accessing the local file system with the standard package java.io.*. It is possible to do statistical data extractions with the trace file in VAT4Net or NAM format, when the NAM trace file is still available. The details are explained in Chapter 5.3.2.

5.2.4 Loading Data from a Remote Source

The difficulty in implementing both architectures - stand-alone and client-server - is dividing the client and the server parts. A further problem is the required amount of data, which is transported over the network from server to client.

VAT4Net Server

VAT4Net provides the server part which delivers parsed trace files to the client to animate and visualize them. If necessary the trace file will be preprocessed first on the server and transported afterwards. Furthermore, the server part is also able to start plugins and delivering precalculated data to the client. This part is also included in the the server part because of the location of the trace file and the matter of fact, that all plugins require the trace file as basis for analyzing and visualizing data.

Implementation Detail

net.sf.vat4net.io.ParsingServer [A.25]: Starting the server with the following command:

java -cp VAT4Net.jar net.sf.vat4net.io.ParsingServer [[host:]port] [directory]

The command starts the server which will listen on the socket defined by host and port (Listing 5.2). Trace files are searched in the file directory which is assigned. The server is on standby until a connection request is incoming. The *ServerSocketChannel.accept()* method is blocking indefinitely until a new connection is available or an I/O error occurs (Listing 5.3). Further details follow in the according subsections.

```
25        ssc = ServerSocketChannel.open();
26        InetSocketAddress isa = new InetSocketAddress(InetAddress.
              getByName(hostname), port);
27        ssc.socket().bind(isa);
```

Listing 5.2: net.sf.vat4net.io.ParsingServer, Part of A.25

```
38        SocketChannel sc = ssc.accept();
```

Listing 5.3: net.sf.vat4net.io.ParsingServer, Part of A.25

VAT4Net Client

The client part of the server-client implementation is responsible for receiving data from the server and communicate it to the VAT4Net client application, which is showing the resulting data as animation, visualization and plugin-defined view in the Graphical User Interface (GUI). The client part is the interface which integrates the server-client architecture into the existing stand-alone and applet application.

Implementation Detail

net.sf.vat4net.io.ParserStub [A.26]: When the VAT4Net client part tries to load a trace file from the remote server it registers a SocketChannel and opens a connection to the server (Listing 5.4). The connection will be setup via a SSH port forwarding. At this moment the server and the client are connected together. When the server accepts the connection (Listing 5.3), the server starts serving the desired data.

```
3     public boolean connect(String host, int port) {
4         (...)
5         sc = SocketChannel.open();
6         InetSocketAddress isa = new InetSocketAddress(InetAddress.
              getByName(host), port);
7         sc.connect(isa);
8         successful = true;
9         (...)
10        return successful;
11    }
```

Listing 5.4: net.sf.vat4net.io.ParserStub, Part of A.26

Connecting the Server with the Client

The connection between the client and the server is achieved by creating a connecting network socket (see Figure 5.3) as mentioned above. Both the server and the client have a socket bound to a specific endpoint. An endpoint is a combination of an IP address and a port number. Every

TCP connection can be uniquely identified by its two endpoints. The server just waits, listening to the socket for a client to make a connection request.

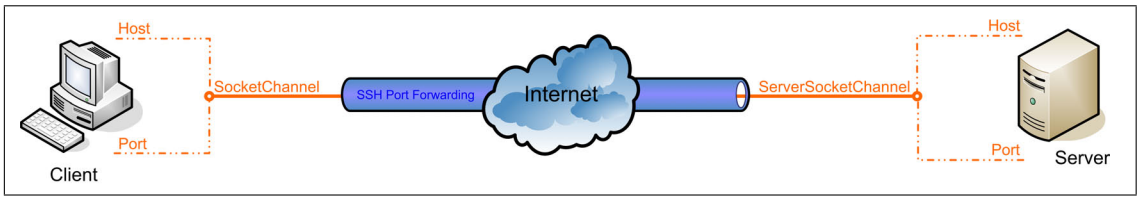


Figure 5.3: VAT4Net Server-Client Connection on a Network Socket

SSH and Port Forwarding

As already mentioned, the connection is setup using SSH port forwarding. Port forwarding, or tunneling, is the use of Secure Shell (SSH) to securely and transparently redirect TCP/IP traffic from a client application to an application server. SSH enables to connect to a server which is behind a firewall and where authentication and authorization is required.

Implementation Detail

net.sf.vat4net.gui.menu.MnuLoadActionListener [A.27]: When the menu command ”Load Trace File...” is selected, the MnuLoadActionListener is invoked and prepares the necessary steps to successfully connect server and client and transport data over this connection. In line 13 of Listing 5.5 the method *setupSSHTunnel()* (Listing A.27, lines 42-62) is called. This method installs a SSH tunnel from the client to the server. The SSH connection is set up with the help of an extern library Ganymed SSH-2 [16]. After the SSH connection is set up, a new instance of the class ParserStub (5.6, line 28) and the connection to the animation and plugin engine is installed (5.6, lines 29-34).

```
13          int tmpPort = setupSshTunnel(-1);
```

Listing 5.5: net.sf.vat4net.gui.menu.MnuLoadActionListener, Part of A.27

```
28          this.parser = new ParserStub();
29          frmRoot.setDataSource(parser);
30          parser.connect(addr,port);
31          parser.startLoadingData(this.buffer);
32          V4N_Parser V4Nparser = new V4N_Parser();
33          TimeController tc = new TimeController(this.buffer,V4Nparser
                ,frmRoot);
34          tc.startAnimation();
```

Listing 5.6: net.sf.vat4net.gui.menu.MnuLoadActionListener, Part of A.27

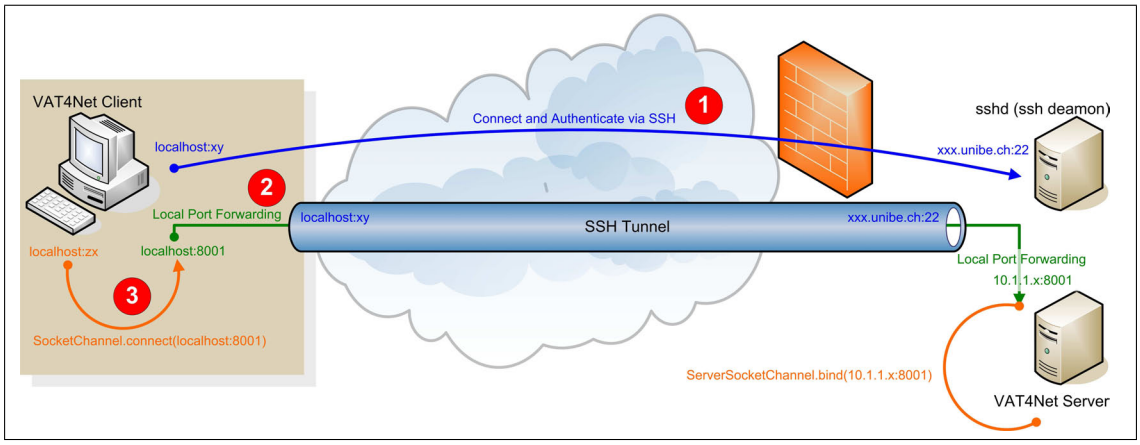


Figure 5.4: VAT4Net Connection setup with SSH Port Forwarding

The example in Figure 5.4 depicts the details of the connection setup from VAT4Net client to a VAT4Net server process. In a first phase, the client connects via SSH to the SSH Daemon (sshd) on a remote machine behind the firewall (Listing A.27, lines 51-54), authenticates and authorizes the client user. In a second step a local port forwarding (Listing A.27, line 56) is installed from localhost:8001 to 10.1.1.x:8001, on the computer where the VAT4Net Server application is running (see Figure 5.4-green arrow). In the last step the client socket will be connected to the start point localhost:8001 of the local port forwarding (Listing A.26, lines 3-11). The server socket is listening on 10.1.1.x:8001 once the server process has been started. The connection between client and server is ready now to transfer data. When the client sends out data, it will be automatically transported via localhost:8001 through the SSH Tunnel to 10.1.1.x:8001 and vice-versa (see Figure 5.4-orange arrow).

Server-Client/Client-Server Data Transfer

As soon as the client and the server have successfully established a connection, the server can start to send the required data. Without any further command the server will only deliver trace lines to the client. But the server is always listening for incoming commands from the client. A possible request of the client is to get special plugin data objects. The delivery of the trace file is halted and the plugin data is served. Once the data is received by the client, the delivery of the trace file data can be resumed.

Implementation Detail

The server is sending data via ObjectOutputStream enclosed in a SocketOutputStream to the client (Listing 5.7, lines 57, 68-70). The client receives this data on ObjectInputStream enclosed in a SocketInputStream (Listing 5.8, lines 21,23-41). The trace file data is sent and received as a String object.

```
57          ObjectOutputStream out = new ObjectOutputStream(sc.socket().
                getOutputStream());
58          (...)
59          FileReader in = new FileReader(f);
60          while(this.proc!=null && !this.proc.finished()){
61              (...) // Inform client, that file is on preprocessing
                    state
62          }
63          while((line = bIn.readLine()) != null){
64              commandAvailable(sc,out);
65              if(connectionClosedByClient){
66                  break;
67              }
68              out.writeObject(line);
69              out.flush();
70              out.reset();
71          }
```

Listing 5.7: net.sf.vat4net.io.ParsingServer, Part of A.25

```
21      ObjectInputStream ois = new ObjectInputStream(sc.socket().
            getInputStream());
22      Object streamObject;
23      while((streamObject = ois.readObject())!=null){
24          if(streamObject instanceof String){
25              if(streamObject.equals("V -t * preprocessing")){
26                  //preprocessing debug message
27              }
28              else if(streamObject.equals("NOD")){
29                  //no object debug message
30              }
31              else if(streamObject.equals("stats")){
32                  //statistic object debug message
33              }
34              else if(streamObject.equals("EOF")){
35                  (...)
36                  buffer.put("EOF");
37              }
38              else{
39                  buffer.put((String)streamObject);
40              }
41          }
42          else if(Class.forName(this.statPlugin.getReturnType()).
                isInstance(streamObject)){
43              this.statPlugin.showStat(streamObject);
44          }
45          streamObject =null;
46      }
```

Listing 5.8: net.sf.vat4net.io.ParserStub, Part of A.26

Further hand the server is listening on a SocketInputStream for receiving other commands (Listing 5.9, line 91) from the client and reacts whenever the command is known by the server. After receiving the command, the server sends the requested data to the client. This is important to handle the different plugins. When the client is receiving data other than string objects, it passes it to the appropriate plugin (Listing 5.8, lines 42-44).

```
91  if (sc.socket().getInputStream().available() > 0) {
```

Listing 5.9: net.sf.vat4net.io.ParsingServer, Part of A.25

5.3 Animation, Visualization and Analysis

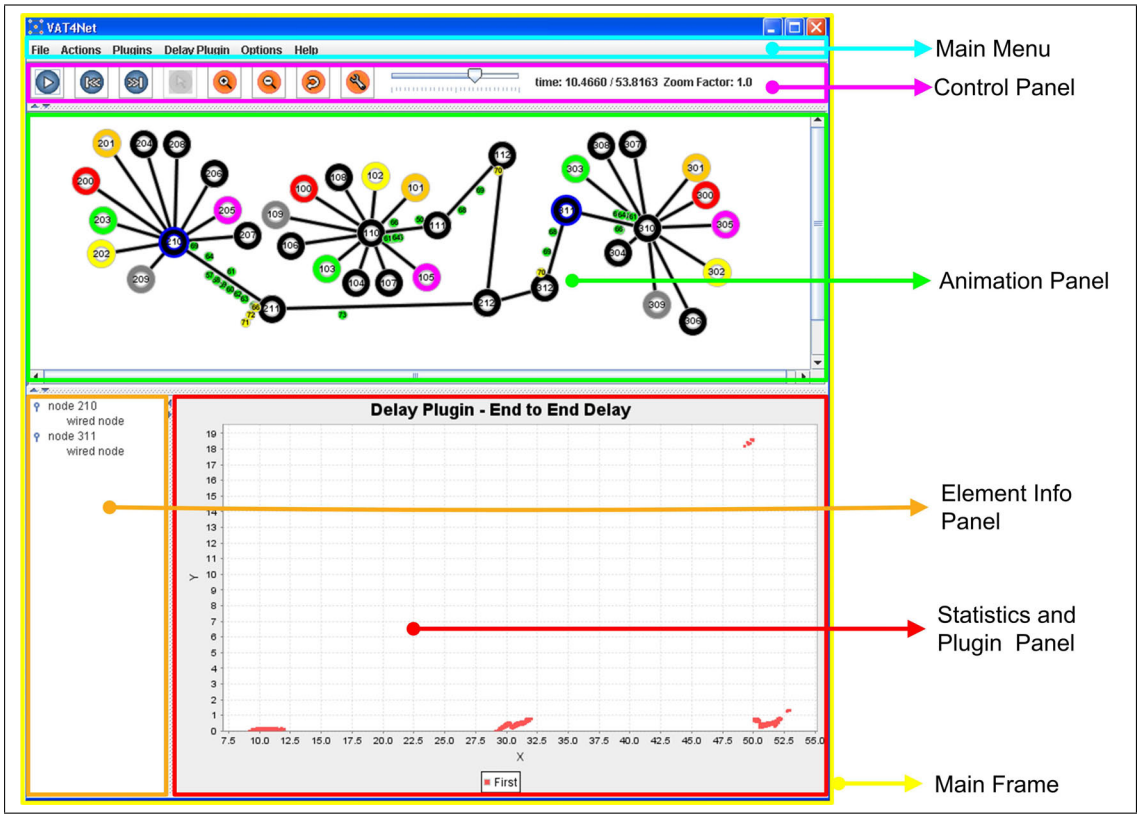


Figure 5.5: Overview of the application of GUI VAT4Net

The VAT4Net GUI (Main Frame) is divided into five parts (see Figure 5.5):

- **Main Menu**: The Main Menu contains all important commands for the whole application (see Section 5.3.1 for further details).

- **Control Panel**: The Control Panel gives access to all animation concerning commands (see Section 5.3.1 for further details).
- **Animation Panel**: The animation process takes place in the Animation Panel (see Section 5.3.1).
- **Element Info Panel**: The Element Info Panel displays properties of nodes and links in a tree based view. It is usually used to display information about selected items (nodes or links).
- **Statistics and Plugin Panel**: All plugin or statistic related views are shown in the Statistics and Plugin Panel (see Section 5.3.2).

The VAT4Net GUI running in stand-alone or applet mode is independent from the application . The following section describes in detail the functionality and implementation of these parts.

5.3.1 The Animation Engine

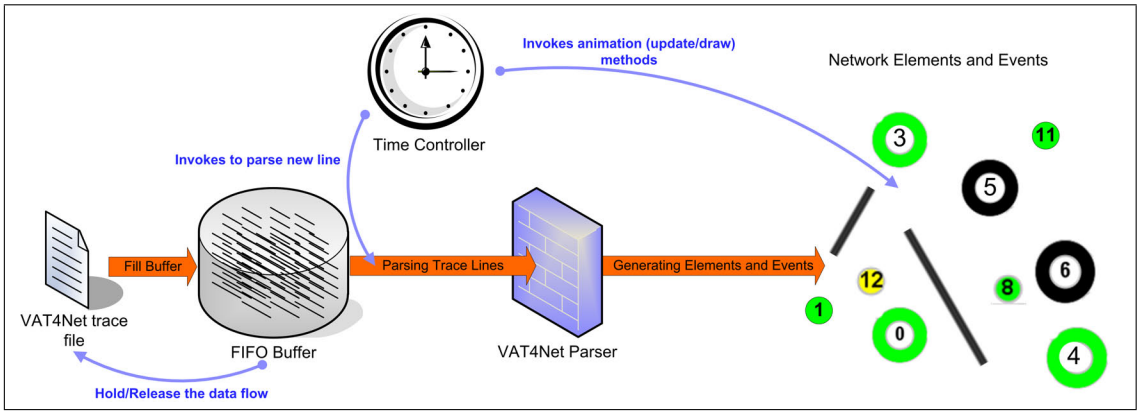


Figure 5.6: VAT4Net Animation Engine Workflow

The main feature of VAT4Net is providing animation and visualization of trace files. After a part of the trace file is available in the buffer either from a local or a remote source, the data will be consumed and the animation can be initialized and started.

There are four main entities involved into the animation process: the Buffer, the Time Controller, the VAT4Net Parser and all elements and events (Figure 5.6). The Time Controller acts as a supervisor over the animation. The VAT4Net Parser converts trace lines holding in the buffer into elements and events.

The Time Controller

”An animation is defined as a rapid display of pictures in order to create an illusion of movement” [wikipedia.org]. The rapid display of pictures is measured by the unit Frames per

Second (FpS), where frames can be exchanged by pictures. The other time unit used in the animation of simulations is the time step. The time step specifies the amount of time that passes from one frame to the other frame displayed. As an example, the following calculation will help to understand the time step and FpS:
When animating with a constant rate of 10 FpS, each 100 ms a picture or frame should be displayed. Therefore each step including parsing, update and repaint should not have a duration longer than 100ms. Otherwise, there would be a delay causing the flow of the animation to halt. To go on with the example, when a time step of 0.01s is selected, resulting in a real time to animation time ratio of 10:1. That means that the animation plays 10 times slower than the simulation in real time.

The Time Controller further act as a timer for the animation and invokes at the right time the necessary methods as parsing new lines from the buffer, updating and drawing simulation elements and events.

Implementation Details

net.sf.vat4net.logic.TimeController [A.28]: From lines 3-14 of the Listing A.28 the animation is prepared. Required constants as *time step* and *period* are set and the animation thread is started. The *run()* (Listing A.28, lines 15-78) method is rather complex and will be described step-by-step: In a first part (Listing 5.10) the animation is initialized and repaint after all required trace lines are parsed by the V4N_Parser (*parseLine(String line)* ,5.10, line 24) to animated the initial state of the animation at time 0.0s.

```
20      while(!networkIsInitialized)
21      {
22          line = (String)buffer.get();
23          (...)
24          parser.parseLine(line);
25          if(!networkIsInitialized && parserTimeStamp>this.
                animationStartTime){
26              networkIsInitialized = true;
27              netPanel.setResetStatus(true);
28              netPanel.setNetwork(network);
29              network.update(this.time);
30              network.connectLinks();
31              netPanel.initialiseNetworkPanel();
32              netPanel.updateTimeDisplay();
33              netPanel.setNetworkInitStatus(true);
34              netPanel.repaint();
35          }
36      }
```

Listing 5.10: net.sf.vat4net.logic.TimeController, Part of A.28

Once the animation has started playing, the second part of the *run()* method is invoked until the animation is finished. In each animation step (equals the chosen time step), first some lines were parsed from the Buffer, the whole network elements and events are updated (Listing 5.11, line 58) and the network will be repainted (Listing 5.11, line 61). It is decided if the calculation of one animation frame causes a delay or the animation thread could sleep for some milliseconds (5.11, lines 62-70).

```
58          network.update(this.time);
59          netPanel.updateTimeDisplay();
60          if(!this.jumpInSimulation){
61              netPanel.repaint();
62              afterTime = System.currentTimeMillis();
63              timeDiff = afterTime - beforeTime;
64              sleepTime = (period - timeDiff);
65              //let thread sleep for the remaining time (1000/fps not
                    used)
66              if (sleepTime > 0) {
67                  (...) // sleep
68              }
69              (...)
70              beforeTime = System.currentTimeMillis();
71          }
```

Listing 5.11: net.sf.vat4net.logic.TimeController, Part of A.28

net.sf.vat4net.gui.animationpanel.NetworkPanel [A.29]: This class is containing the JPanel implementation for the animation part of the Main Frame. The required method *paintComponent(Graphics g)* (Listing A.29, lines 16-27) is always invoked, when calling the method *NetworkPanel.repaint()*. Further the method *initialiseNetworkPanel()* (Listing A.29, line 3-17) does its work in the initialization process of the animation (node placing, wireless grid setup).

The VAT4Net Parser

The VAT4Net Parser translates trace lines into elements and events. This elements and events can be animated - updated and drawn.

Implementation Details

net.sf.vat4net.io.V4N_Parser [A.30]: When parsing a line, first there will be a categorizing selection (Listing A.30, lines 4-22) on each trace line into the different elements and events. When having categorized the trace line, there is the appropriate setup method chosen. In a setup method the appropriate element and its event will be generated. As example the nodeSetup() method generates an element of the type Node (Listing 5.12, lines 26-36) and events of the type NodeUpdateElement (Listing 5.12, lines 37-44).

```
26        Node node;
27        int nodeID = Integer.parseInt(nodeTable.get("-s").toString());
28        //check if node already exists, when yes, try to make an update
              element
29        if(tc.getNetwork().existNode(nodeID)){
30            node = tc.getNetwork().getNode(new Integer(nodeID));
31        }
32        else
33        {
34            node = new Node(nodeID, netPanel);
35            tc.getNetwork().addNode(node);
36        }
37        String time = nodeTable.get("-t").toString();
38        NodeUpdateElement element = new NodeUpdateElement(time);
39        (...)
40        element.setXCoord(new Double(Double.parseDouble(nodeTable.get("
              -x").toString())));
41        element.setYCoord(new Double(Double.parseDouble(nodeTable.get("
              -y").toString())));
42        element.setMovementDuration(new Double(Double.parseDouble(
              nodeTable.get("-T").toString())));
43        (...)
44        node.addUpdateElement(element);
```

Listing 5.12: net.sf.vat4net.io.V4N_Parser, Part of A.30

Animation Control

In Figure 5.7 the VAT4Net menu and the main control panel is shown and described. This control structure enables the access to all required functions provided by VAT4Net.

Implementation Details

net.sf.vat4net.gui.menu.*: The package contains all menu relating classes and the according java.awt.event.ActionListener for each menu point.

net.sf.vat4net.gui.animationpanel.ControlPanel: The ControlPanel implements animation relating control buttons and its functionality as described in Figure 5.7.

Animated Simulation Elements and Events

The animation engine visualizes and animates the network topology with all its elements and events occurring in the simulation. Wired nodes, links, queues and wireless network grids are more or less static elements while wireless nodes and packets are moving objects. Each of those

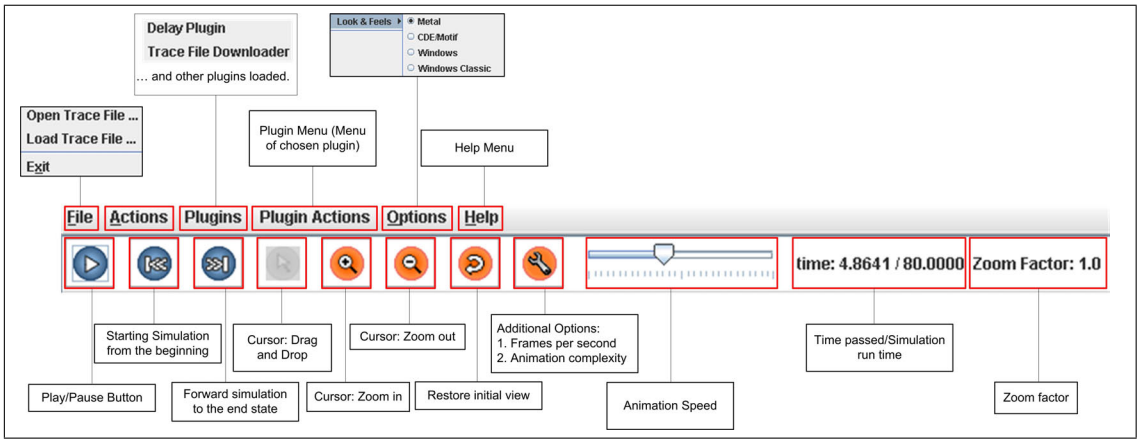


Figure 5.7: VAT4Net Menu and Animation Control

elements are described in detail in the further sections.

Network Element

The Network element is the main element in the animation process. It is aware of all other elements and events in the animation. All updating and drawing calls are controlled from it. When animating wired networks the Network element is shape-less, because there is no defined network area available where the simulation is positioned. It is depicted as a white plane. Otherwise when animating wireless simulations, the network grid is shown (see Figure 5.8).

Implementation Details

The Network is the main part of the animation element tree. The animation engine basically communicates over this class.

net.sf.vat4net.logic.Network [A.31]: In a first part the class Network is responsible to do some initialization actions when loading a simulation into VAT4Net. When loading the trace file nodes and links are not connected to each other. The method *connectLinks()* makes this connections. Further wired nodes has no position information that can be extracted from the trace file. The method *nodePlacing2(int width, int height)* uses an algorithm to spread the nodes over the panel. As a main part the class Network holds all elements and events in Lists. From the Network instance the *update()* (line (Listing 5.13, 18-31) and *draw()* (Listing 5.13, lines 6-17) methods on all this elements and events are invoked.

net.sf.vat4net.gui.animationpanel.NetworkShape: This class is responsible to animate the wireless network grid.

```
6     /*Draw the Network onto the panel */
7     public synchronized void drawNetwork(Graphics2D g)
8     {
9        //draw Network environment
10       networkshape.draw(g);
11       // draw nodes
12       for (Iterator it = nodes.values().iterator(); it.hasNext();) {
13          ((Node) it.next()).draw(g);
14       }
15       //draw all other network elements
16       (...)
17    }
18    /*Updates all object in the network*/
19    public synchronized void update(double time)
20    {
21       /** Update Nodes first*/
22       Node node;
23       for (Iterator it = nodes.values().iterator(); it.hasNext();) {
24          node = (Node) it.next();
25          if(node.doUpdate()){
26             node.update(time);
27          }
28       }
29       //updating all other network elements
30       (...)
31    }
```

Listing 5.13: net.sf.vat4net.logic.Network, Part of A.31

Wired and Wireless Nodes

The node is a basic structure of ns-2 and stands for entities like routers, terminals, servers or mobile devices. In the simulation itself, these nodes are not distinguishable from the different types. In the animation, this fact leads to the point, that each node has the same shape. If defined in the simulation, its is possible to color these nodes different. Further there will be differentiated between wireless and wired node. Wired nodes are always connected through links to each other while wireless nodes are ”connection-less”. Some different node pictures are arranged in Figure 5.9.

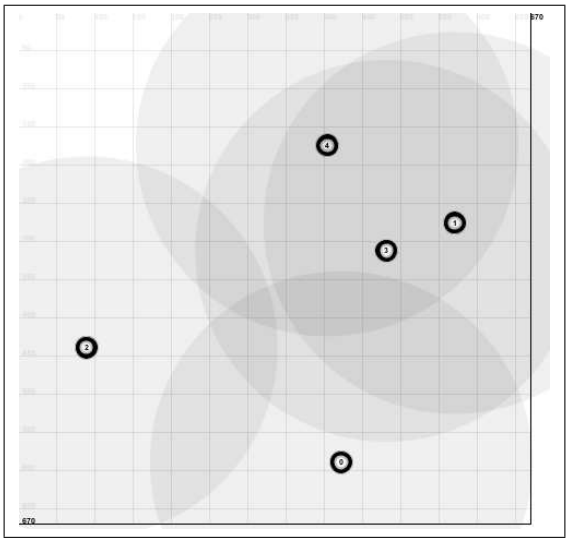
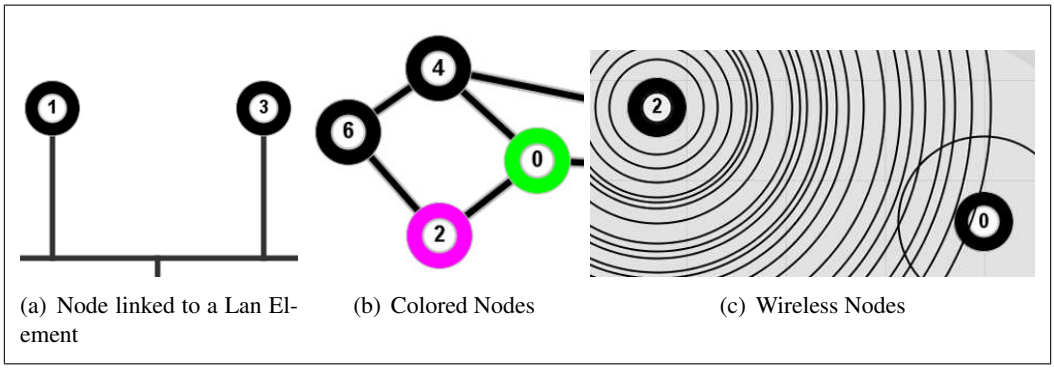


Figure 5.8: Wireless Network in VAT4Net



(a) Node linked to a Lan Element (b) Colored Nodes (c) Wireless Nodes

Figure 5.9: Nodes

Implementation Details

Each node has a class holding data and a referring shape class with position and design concerning data. If the node changes its appearance or position during animation there are NodeUpdateElements available. Further a node has a NodeMouseAdapter for moving and marking actions. Wired nodes are movable while wireless nodes cannot be moved, because of their own position information and defined movements.

net.sf.vat4net.logic.Node [A.32]: When generating a new node the initial data is set (Listing 5.14). The method *update(double time)* is called each time the whole network is updated. After updating the node the animation engine will call the *draw(Graphics2D g)* method, which calls the draw method of the NodeShape class. When setting up a wireless node, a net.sf.vat4net.logic.Queue instance is generated and linked to this node. Normally a Queue is connected to a link, but while wireless nodes do not have links, the node has to takeover this part. Details on the implementation of a Queue are described later on.

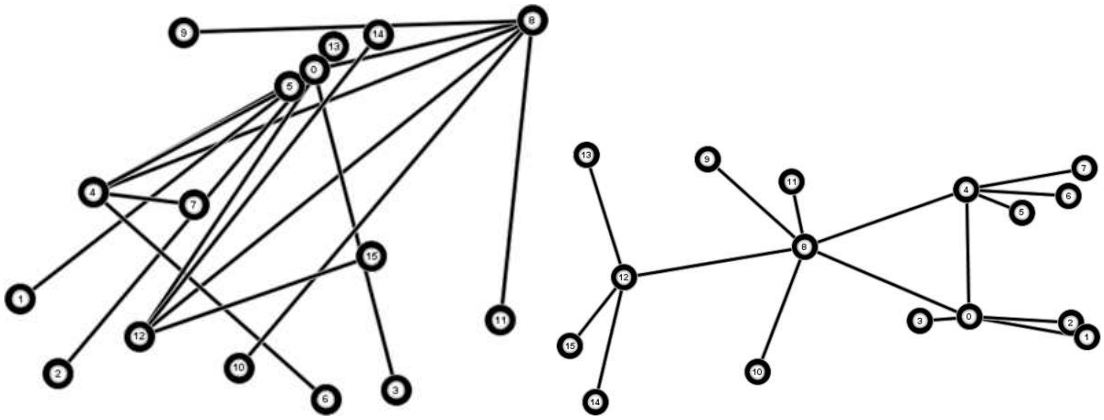
```
3     public Node(int nodeID,NetworkPanel panel){
4         (...)
5         this.nodeID = nodeID;
6         this.updateEvents = new Vector();
7         (...)
8         //Node Shape
9         nodeshape = new NodeShape(this);
10        //Event Listener (Mouse)
11        panel.addMouseListener(new NodeMouseAdapter(nodeshape,panel,
              mousePoint));
12         panel.addMouseMotionListener(new NodeMouseMotionAdapter(
               nodeshape,panel,mousePoint));
13    }
```

Listing 5.14: net.sf.vat4net.logic.Node, Part of A.32

net.sf.vat4net.logic.NodeUpdateElement: Each change to a node is held in a NodeUpdateElement, where position information, shape or state change, etc. are stored. As a main variable in this class a time stamp is set. The time stamp defines when the update takes place.

net.sf.vat4net.logic.NodeShape [A.33]: As already mentioned the NodeShape class is holding all information needed to draw the node to the animation panel of VAT4Net. In each repaint process of the whole animation the *draw(Graphics2D g)* takes place.

Placing the Nodes over the Animation Panel



(a) Algorithm based on NAM (b) Spring Model Algorithm

Figure 5.10: Node Placing Algorithms

Placing wired network nodes over the animation panel is a necessary task in the initialization

process of the network animation. From almost all wired network simulations in contrary to wireless networks, there is no location information available in the resulting trace files.

In a first approach, the graph drawing algorithm of NAM was implemented in VAT4Net by translating the C++ code into Java. The resulting networks drawn by this algorithm were unstructured and badly distributed over the panel. When displaying more than 10 wired nodes, the resulting graph has a lot of overlapping nodes and links (see Figure 5.10(a)). While the algorithm from NAM tries to place the nodes randomly over the animation panel without evaluating the network structure, the newly chosen algorithm implements the spring model, where the calculation is based on attractive and repulsive forces of nodes.

The new algorithm, "a modification of the spring-embedder model of Eades, should:

1. Distribute the vertices evenly in the frame.
2. Minimize edge crossings.
3. Make edge lengths uniform.
4. Reflect inherent symmetry.
5. Conform to the frame." (Fruchterman and Reingold 1129 [17])

The implemented algorithm does not explicitly strive for all these goals above but for the needs of VAT4Net, the implementation based on the Pseudocode listing A.34 is almost optimal. Compared to the old algorithm the new method is much less memory consumptive and significant faster and gives useful results of node placing (see Figure 5.10(b)).

Implementation Details

The implementation of the node placing algorithm takes its place in the initialization process of the animation. When each node is generated the method *nodePlacing2(int height,int width)* available from the class net.sf.vat4net.logic.Network A.31 is invoked.

Wireless nodes are not affected by the node placing problem. The position information is available from the trace file. In the next three lines of a VAT4Net trace file, the position is set by x an y coordinates:

```
n -t * -s 0 -x 83.364418416244007 -y 239.43800983126101 (...)
n -t * -s 1 -x 257.04629832315698 -y 345.35773177920402 (...)
n -t * -s 2 -x 591.25656009383295 -y 199.37330681680399 (...)
```

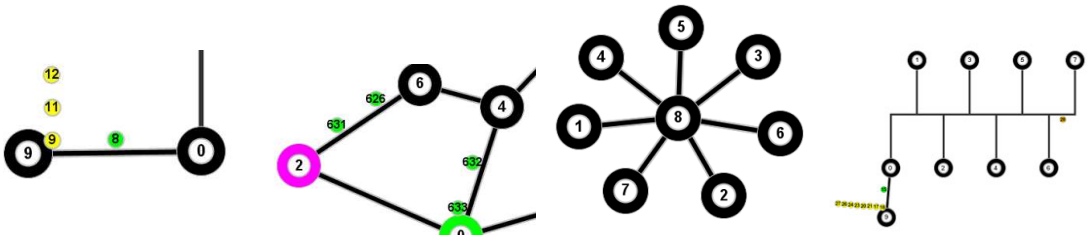
Links and Queues

A link is a base structure in ns-2 simulations and connects wired nodes to each other. In the animation there is no difference between duplex and simplex links. Each link is designed as

a line between two nodes. In Figure 5.11(b) and 5.11(c) two different network topologies are shown.

Queues represent locations where packets may be held (or dropped) in the ns-2 simulation. Queues are connected to links. If there is no link between nodes in case of a wireless simulation, the queue is connected to a node. In Figure 5.11(a) the yellow packets are queued. There is no shape to depict a queue. The queued packets are different in color and are stringed together vertically to the link.

A further element similar to the Link element, is the LAN Element. It is implemented according to the Link element. A LAN Element is depicted in Figure 5.11(d).



(a) Link with Queued Packets (b) Links - Ring Topology (c) Links - Star Topology (d) LAN Element

Figure 5.11: Links

Implementation Details

While the Link class is very similar to the Node implementation described above, the Queue class requires further detailed description.

net.sf.vat4net.logic.Link: The Link class is similarly implemented as the Node class. There is a *update(double time)* and a *draw(Graphics2D g)* method available invoked from the Network instance. When setting up links connected to wired nodes, a Queue instance is generated and connected to this link. There is also a class LinkUpdateElement available. There are only changes in the link color possible.

net.sf.vat4net.gui.animationpanel.LinkShape: Parallel to the NodeShape class, this class is implemented with the details to animate a link in the animation. In Figure 5.12 the packet flow direction is depicted. While each link is simplex or duplex with the same shape it is important, that each link has the same packet flow direction. A help class of LinkShape is net.sf.vat4net.gui.animationpanel.LinkPath. It calculates the adequate position on the link instance for a packet, which has passed x percent of the way over the link.

net.sf.vat4net.logic.Queue [A.35]: The class queue has two different constructors; one for queues connected to links and one for queues connected to wireless nodes. Because a graphic view of the class Queue is missing, the class QueueShape is only a helper class. Due to this fact a draw and update method are not implemented. More important are the methods *enqueue(PacketUpdateElement packetElement)* (Listing 5.15, lines 12-16) and *dequeue(PacketUpdateElement packetElement)* (Listing 5.15, lines 18-22). This two methods handle the amount of packets queued in the queue instance at certain time. The two methods are invoked from within the net.sf.vat4net.logic.Packet class.

```
12      /**Enqueue packet on this queue */
13      public void enqueue(PacketUpdateElement packetElement) {
14          queuedPackets.addElement(packetElement);
15          this.queueshape.addQueueHeight(packetElement.getShape().
                getRadius()*2);
16      }
18      /**Dequeue packet on this queue */
19      public void dequeue(PacketUpdateElement packetElement) {
20          queuedPackets.removeElement(packetElement);
21          this.queueshape.minusQueueHeight(packetElement.getShape().
                getRadius()*2);
22      }
```

Listing 5.15: net.sf.vat4net.logic.Queue, Part of A.35

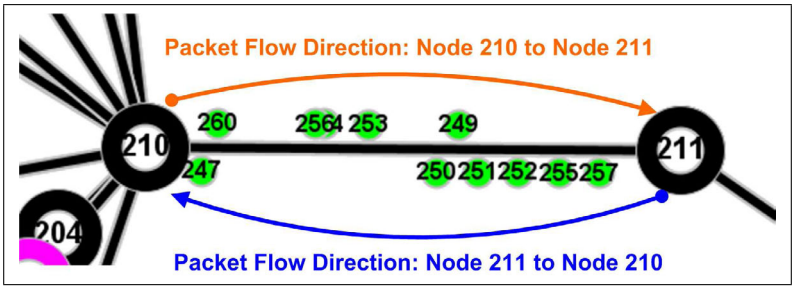


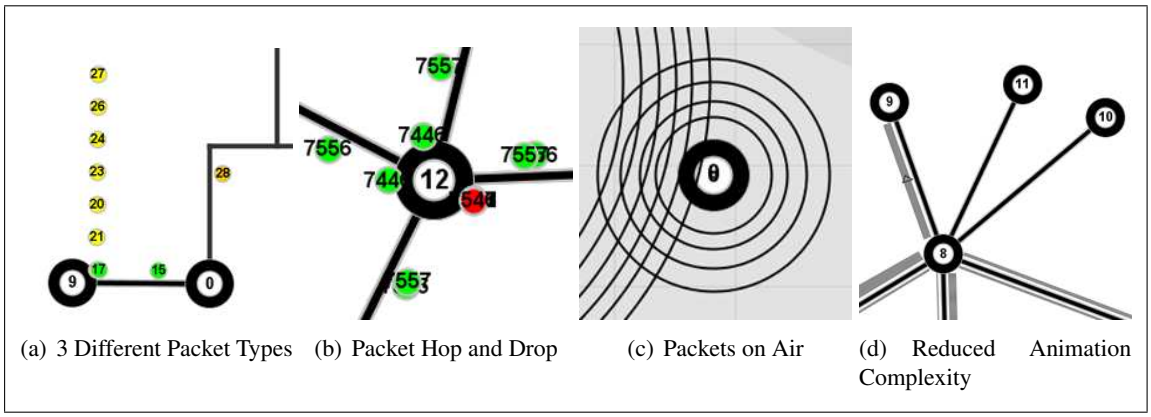
Figure 5.12: Packet flow direction of links in the VAT4Net animation

Packets

Packets are the most important elements of a simulation. Also for the animation packets are the most interesting and the most significant part. There are different looks a packet can have in the animation process (see Figure 5.13). The different packet types are:

- **yellow**: queued packets, Figure 5.13(a)
- **green**: packets on a link, Figure 5.13(a) and Figure 5.13(b)
- **orange**: packets on a LAN Element, Figure 5.13(a)

- **red**: dropped packets, Figure 5.13(b)
- **circle lined**: packets on air, Figure 5.13(c)
- **combined**: packets on a link depicted in reduced animation complexity, Figure 5.13(d)



(a) 3 Different Packet Types (b) Packet Hop and Drop (c) Packets on Air (d) Reduced Animation Complexity

Figure 5.13: Packets

The packet movements can be extracted out of the VAT4Net trace file. One line equals one hop from a node to another node. It further includes data from enqueue a packet on a queue to receive a packet at a node or the drop event. The VAT4Net trace file also delivers the very last action of a packet. This is important concerning memory consumption and time complexity problems. After a packet has done its last action it can be removed from the animation and needs no space in the heap anymore. It shows the chain of a packet. As example we look at

(a) Enqueue, t0 (b) Dequeue/Hop, t1=t2 (c) Receive, t3

Figure 5.14: Packet Movement - Packet with Packetid=3

a normal hop from one node to another (see Figure 5.14). First a packet is enqueued (in the queue) at a time t_0. That can be the first action of a packet or the first action of another hop action. At a time t_1 the packet will be dequeued from the queue. At this time t_1 the packet "enters" the link. This hop event takes place at time t_2. More or less it is t_1=t_2. The packet will be on this link until it is received from the destination node at time t_3. While on the link between the time t_2 and t_3 the packet is assumed to be on the same speed at every moment. So it is possible to calculate the position of the packet for every time when it is on a link.

Implementation Details

Different from the other update elements (from nodes, links etc.) the update event class PacketUpdateElement has more importance. It implements the *update(double time)* method itself. Further there are two different ways to show an animated packet, either as a circle (PacketShape) or combined with other packets in reduced complexity mode (PacketsShape).

net.sf.vat4net.logic.Packet [A.36]: Both, the *draw* (Listing below 5.16, line 10-14) and *update* method are dedicated in the same way to the PacketUpdateElement as it is possible that one Packet instance is seen more than one time on the animation panel at the same time. This behavior occurs for example when doing multicast or broadcast requests.

```
10  public void draw(Graphics2D g){
11      for(int i = 0;i<updateEvents.size();i++){
12          ((PacketUpdateElement)updateEvents.get(i)).draw(g);
13      }
14  }
```

Listing 5.16: net.sf.vat4net.logic.Packet, Part of A.36

net.sf.vat4net.logic.PacketUpdateElement [A.37]: When invoking the *update(double time)* method of a PacketUpdateElement, depending on the actual time there are different ways to handle the packet. For example when the enqueue action is in time, the method *handleEnqueue(double time)* is invoked (see Listing 5.17, lines 24-30 and 5.18, lines 69-71). In this handle methods, the packet state is changed according to the action invoked at a defined time.

```
24      //ENQUEUE
25      if(time >= this.startTime){
26          if(!this.enqueued){
27              handleEnqueue(time);
28          }
30      }
```

Listing 5.17: net.sf.vat4net.logic.PacketUpdateElement, Part of A.37

```
69  private void handleEnqueue(double time){
70      (...) // do the handle actions on a packet state
71  }
```

Listing 5.18: net.sf.vat4net.logic.PacketUpdateElement, Part of A.37

net.sf.vat4net.gui.animationpanel.PacketShape: This class implements the *draw* method which defines the different looks of packet. This class is implemented similarly to the other shapes occurring in the animation.

net.sf.vat4net.gui.animationpanel.PacketsShape: The PacketsShape class is connected to a link. Each link has a instance of this class. When the animation is in reduced animation complexity mode, each packet will be added to the appropriate instance. The PacketsShape instance looks as shown in Figure 5.13(d). The more packets are on a link the wider is the grey ”packet”-bar.

5.3.2 The Add-on Engine

This component of VAT4Net is responsible for all actions and functions in addition to the animation of network simulations. Primarily this part is able to process statistical data, e.g. end-to-end delay, packet loss rate and other Quality of Service characteristics. It is also possible to add further functionality, e.g. a file downloader for remote trace files, a file uploader, a trace file generator, etc., supporting the practical work on the laboratory.

For example when generating trace files on a remote computer there is no way to download this trace file or a part of it with the Mindterm Java applet from the laboratory computer. This could be a nice add-on beside of the statistical plugins.

The Plugin Architecture

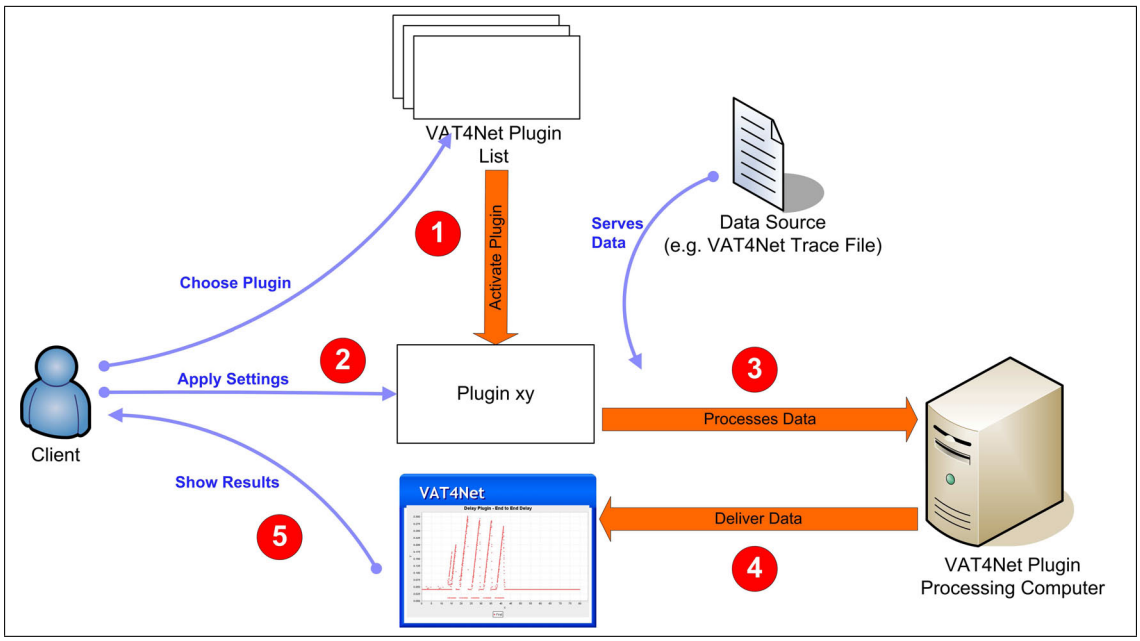


Figure 5.15: VAT4Net Plugin Architecture - workflow of the plugin engine.

The analysis and add-on engine is implemented to support modular implementation of various plugins. The plugins are implemented as modular packages with predefined interfaces to the VAT4Net application.

The workflow of the plugin-based functionality is shown in Figure 5.15. In a first step the user has to select a plugin from the ”Plugin” menu (Figure 5.7). The plugin and the corresponding plugin menu are activated (Figure 5.15, step 1). As soon as the plugin is

activated, the user has to apply settings required by the plugin and to select the preferred action the plugin provides (Figure 5.15, step 2). As soon as all user required inputs are successfully done, the plugin starts calculation and preparation and delivers the data to the user. If the application runs as a monolithic system the plugin's process is executed on the local computer. Otherwise, if VAT4Net is running in server-client mode the plugin calculation process is executed on the server. For statistical plugins the VAT4Net trace file is in most cases the data source for calculating and preparing the output (Figure 5.15, step 3). The details on this process are described by an example plugin following later in this chapter.

As soon as the calculation and preparation process is completed, the plugin delivers the results to the VAT4Net application (Figure 5.15, step 4). The data is visualized in the "Statistic and Plugin Panel" (see Figure 5.5) or the action can be completed by the user (e.g. saving the trace file on the local computer). This last step (step 5) is shown in Figure 5.15.

Implementation Details

When the VAT4Net Application is started, the plugin list is loaded and shown in the plugin menu. When a plugin is selected from the menu the appropriate ActionListener implemented in the plugin itself is invoked (Listing 5.19, line 30-35). The example listings are taken from the Delay Plugin which is described below. All further implementation details are listed in the paragraph "An Example Plugin".

```
26  public JMenu getActionMenu() {
27      String name = DelayPlugin.class.getName();
28      JMenu menAction = new JMenu(name.substring((name.indexOf('.') >
            0)? name.indexOf('.'):0));
30      endToEndDelayItem = new JMenuItem("End-to-end delay");
31      endToEndDelayItem.addActionListener(new ActionListener() {
32          public void actionPerformed(ActionEvent e) {
33              calcDelay(END_TO_END);
34          }
35      });
36      menAction.add(endToEndDelayItem);
37      (...)
38      return menAction;
39  }
```

Listing 5.19: net.sf.vat4net.plugin.DelayPlugin, Part of A.38

net.sf.vat4net.io.PluginLoader [A.39]: The method *getPluginList()* (Listing 5.20) is invoked at the startup of VAT4Net. It loads all plugins listed in the plugins.properties file (e.g. *net.sf.vat4net.plugin.DelayPlugin*).

```
9   public ArrayList getPluginList() {
10     URL listURL = FileLoaderUtil.createURL("plugins.properties");
11     (...)
12     Properties pluginsFile = new Properties();
13     pluginsFile.load(listURL.openStream());
15     for (Enumeration keyEnum = pluginsFile.keys(); keyEnum.
           hasMoreElements();) {
16        String key = (String) keyEnum.nextElement();
17        Class pluginClass = this.getClass().getClassLoader().
              loadClass(key);
18        plugins.add(pluginClass.newInstance());
19     }
20     (...)
21     return plugins;
22  }
```

Listing 5.20: net.sf.vat4net.io.PluginLoader, Part of A.39

An Example Plugin

The implemented Delay plugin which calculates the end-to-end delay between two selected nodes is described in detail. After loading a trace file into the animation engine, this trace file is also available for the Delay plugin. The user selects two nodes in the "Animation Panel" on which he prefers to calculate the end-to-end delay. The calculation of the end-to-end delay is derived as described in the Section 7.4.3. The plugin calculates the statistical values and delivers it to the "Statistic and Plugin Panel" (Figure 5.16). The data is then shown in a chart with the simulation time as x-coordinate and the delay as y-coordinate. The chart is drawn with the help of the additional framework JFreeChart [18]. The framework is distributed under the terms of the GNU Lesser General Public Licence (LGPL). The framework is available for all other plugins too. It is included in the VAT4Net application.

Implementation Details

The implementation of all other plugins is similar to the implementation of the Delay Plugin. The process is described as follows:

net.sf.vat4net.plugin.DelayPlugin [A.38]: The class DelayPlugin is the main class of the Delay Plugin. As soon as the plugin is started from the menu, the method *calcDelay(int type)* is invoked. A command is send out to start the calculation thread on the server (Listing 5.22, line 53). Otherwise the calculation thread is started immediately (Listing 5.22, line 59).

The *calculateEndToEnd(...)* method prepares all data required for the method *showStat(Object statsData)* (Listing 5.21, lines 64-82). The required object for this method is either received from the server or already available from the application.

```
64    public void showStat(Object statsData){
65       if(statsData instanceof List){
66          XYSeriesCollection dataset = new XYSeriesCollection();
67          XYSeries series1 = new XYSeries("First");
68          double data[][] =(double[][])((List)statsData).toArray(new
                double[2][((List)statsData).size()]);
69          for(int i = 0; i< data[0].length;i++){
70             series1.add(data[0][i],data[1][i]);
71          }
72          dataset.addSeries(series1);
73          JFreeChart chart =  ChartFactory.createScatterPlot(
74                //chart settings
75                (...)
76                dataset,
77          );
78          chartPanel = new ChartPanel(chart,false);
79          this.refreshComponent(chartPanel);
80          this.calculateStatisticThread = null;
81       }
82    }
```

Listing 5.21: net.sf.vat4net.plugin.DelayPlugin, Part of A.38

```
46       if(mainFrame.getDataSource() instanceof ParserStub){
47          if(type == END_TO_END){
48             this.title = "End to End Delay";
49             List.class.cast(this.returnType);
50          }
51          (...)
53          ((ParserStub)mainFrame.getDataSource()).sendStatisticRequest
                (this,String.valueOf(type),((Node)selectedNodes.get(0)).
                getNodeID()+","+((Node)selectedNodes.get(1)).getNodeID()
                );
54       }
55       //The calculation is done one the local computer
56       else{
57          if(type == END_TO_END){
58             this.title = "End to End Delay";
59             this.calculate((File)mainFrame.getDataSource(),type,((
                   Node)selectedNodes.get(0)).getNodeID()+","+((Node)
                   selectedNodes.get(1)).getNodeID());
60          }
61          (...)
62       }
```

Listing 5.22: net.sf.vat4net.plugin.DelayPlugin, Part of A.38

In the last step the "Statistic and Plugin" panel is refreshed (Listing 5.21, lines 78-79) and the user receives the resulting plugin output.

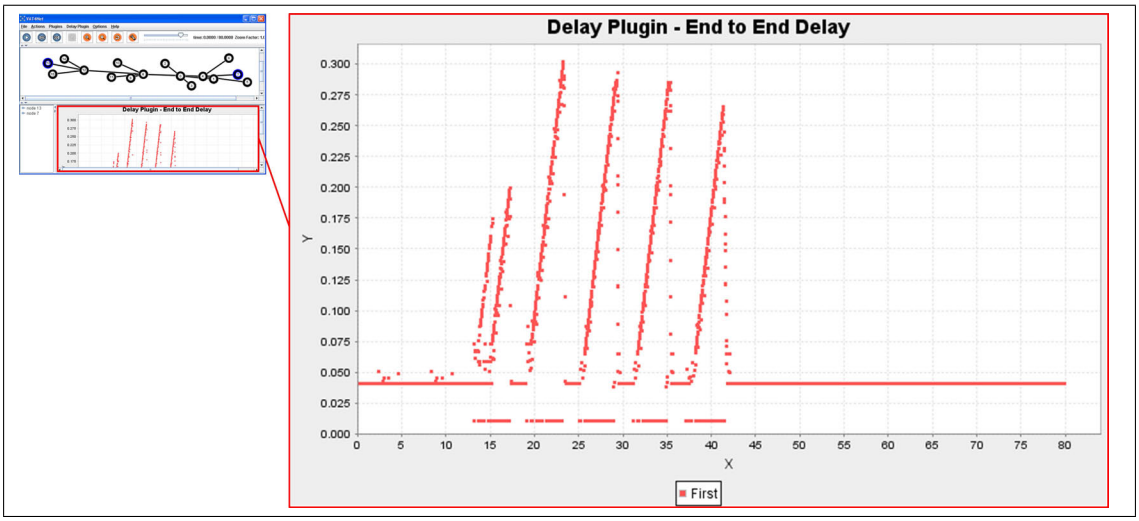


Figure 5.16: VAT4Net Delay Plugin Example - End-to-End Delay

5.4 Performance Evaluation

The first prototype version of VAT4Net suffered from high memory consumption. The first problem was the loading process of a whole simulation. Loading a trace file completely requires a lot of memory depending on the duration and level of detail of the chosen simulation. Trace files bigger than 64MB could not be animated in VAT4Net. In a new development phase the process of loading was changed from loading all at once into a streaming like process with buffering required and releasing old data. This reduces the memory consumption and solves the problem.

Test Scenario - Changing the Buffer Size

In order to test the effects of the buffer size on the whole animation process the sum of delays to calculate one time step were measured. In detail the test has been executed as follows:

- **Test computer**: Intel Pentium M 1.6 GHz, 512 MB RAM, Windows XP
- **Test file**: ∼69MB, 172'018 lines, VAT4Net trace file format, simulation with 16 wired nodes
- **Frames per Second (FpS)**: constant, 10 FpS
- **Time Steps**: variable; 0.1,0.5,1.0 seconds
- **buffer size**: variable; 1'000, 15'000, 20'000, 37'500, 50'000, 75000
- **Delay (of animation steps)**: If the time used to calculate and to repaint one animation step is greater than the specified period of time (1000/FpS), the step is counted as delayed.

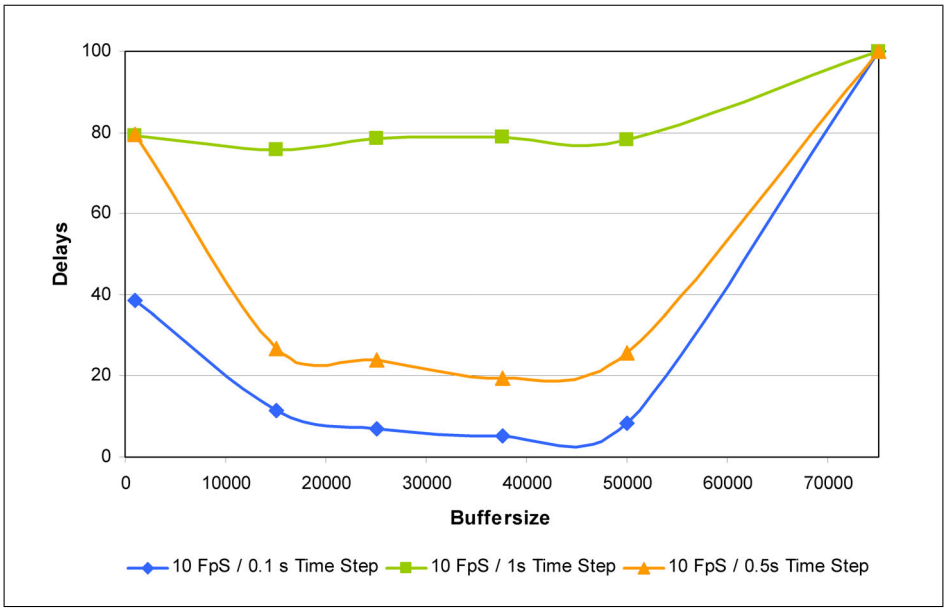


Figure 5.17: VAT4Net - Delay in subject to the buffer size

- **Delays**: Amount of delays (of animation steps) occurring at an animation of a simulation

The resulting delays of each test are shown in Figure 5.17. When starting the animation with the buffer size of 75'000 there was a java.lang.OutOfMemoryError in the Java heap space (when a OutOfMemoryError occurred in the test case, the amount of delays was set to 100). Further the test scenario with a time step of 1 second (green line) shows, that big time steps cause a high amount of delays. When doing big time steps there is a lot of simulated data to calculate between two steps, independent of the size of the buffer. Big time steps require more computational time than smaller steps.

Otherwise when doing small animation steps (0.1 second - blue line), the buffer size plays an important role when talking about the animation flow. The less delays an animation process has, the better the animation flow is. Therefore, a buffer size of 37'000 would be the best matching for the test scenario.

Chapter 6

Virtual Network Simulation

6.1 IRNSI - Internet Remote Network Simulator Infrastructure

The laboratory part of the e-learning infrastructure provided by EuQoS consists of three main components, the laboratory computers itself (see Figure 6.2) with ns-2 and VAT4Net as the main applications, the resource management server and the laboratory portal server. Generally it is not dictated by the e-learning system how this three parts have to look like. In the scope of the project, the Computer Networks and Distributed Systems research group (RVS) from the University of Bern has proposed a possible solution in the form of the Internet Remote Network Simulator Infrastructure (IRNSI) [6].

IRNSI provides a fully web accessible laboratory for network simulations including reservation system, laboratory portal server and laboratory computers. It offers a way to implement and integrate hands-on sessions based on network simulations into an existing e-learning course platform. There are several EuQoS modules using IRNSI for the hands-on sessions. There is nearly no constriction in extending IRNSI with other laboratory setups. It would be easy to substitute the Network Simulator (ns-2) with another network simulation software.

6.2 ns-2 Platform

The actual laboratory resides on a system (Fig. 6.2, Part 3), where multiple instances of an operating system can be run under User-Mode Linux (Fig. 6.2, Part 4). Each of this instances acts as a laboratory seat used by one user, who has a time-slot reservation. This setup facilitates a completely separated, stand-alone working environment for each user with the possibility to resume work in a later time-slot or after all to reset the laboratory to initial state.

User-Mode Linux (UML) allows multiple sand-boxed virtual instances of Linux to run as a stand-alone application on the Linux host system. When accessing the hands-on session for the first time, a standard predefined image of a User-Mode Linux instance is loaded and the client gets a blank and clean laboratory seat.

For the module ”Implementing Protocols on Network Simulators”, the standard Linux image contains the following parts:

- Standard Linux software (vi)
- Pre-compiled ns-2 Version 2.29 inclusive examples and source code delivered with ns-2
- C++ Compiler (gcc version 3.3.5)
- Template files for hands-on sessions

With some basic Unix/Linux and enhanced C++ knowledge and the theory part of the module, the learner should successfully pass the hands-on session of the module.

To allow a resuming of the work done on the laboratory during earlier sessions, a so-called Copy-On-Write (COW) file is stored for each user. The COW file contains all changes done by the user, such as newly generated files, system setting and changes to software, for example a recompiled ns-2 instance as it is needed in the module. The COW file in association with the write protected predefined UML image forms the current state of one user's laboratory work. Each user can reset his own laboratory seat into its initial state, but he is not able to step back or choose a mid session's state.

Interaction with the ns-2 Platform

IRNSI offers different accessing and communication methods (see Fig. 6.1) to interact with the laboratory. Each of these methods allows the user to interact with the laboratory without any special software requirements. In Table 6.1 the requirements of the main communication methods used in the module ”Implementing Protocols on Network Simulators” are described. The requirements have to fulfill the requirements defined in the Design and Didactics Guide [11].

Application	**Connection Technology**	**Requirements**
Laboratory Portal(Figure 6.1, Part 1)	Authentication and Authorization via AAI Secure Sockets Layer (SSL)	Web Browser with Java Plugin
Mindterm Java Applet (Figure 6.1, Part 2)	SSH	Java Virtual Machine (JVM)
VAT4Net Java Applet (Figure 6.1, Part 3)	SSH (Remote Port Forwarding, SSH Tunnel)	JVM

Table 6.1: IRNSI Communication Methods - Connection Technology and Requirements

The Mindterm Java applet [19] is a widely used client that implements the SSH1 and SSH2 protocols written in pure Java. The SSH applet is used to connect the user with the laboratory ”seat”, that means in fact to the command line of the UML instance the user is assigned to. In detail, the user logs to the Laboratory Portal Server with the SSH applet, where

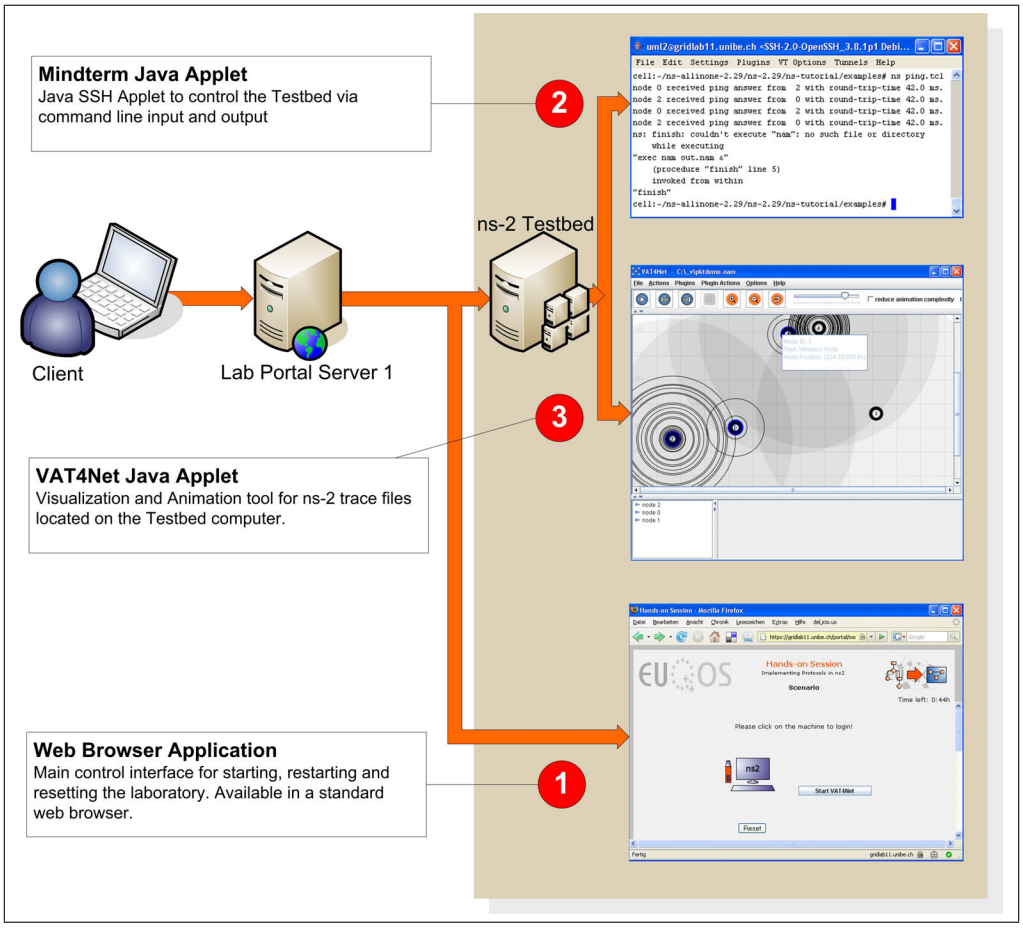
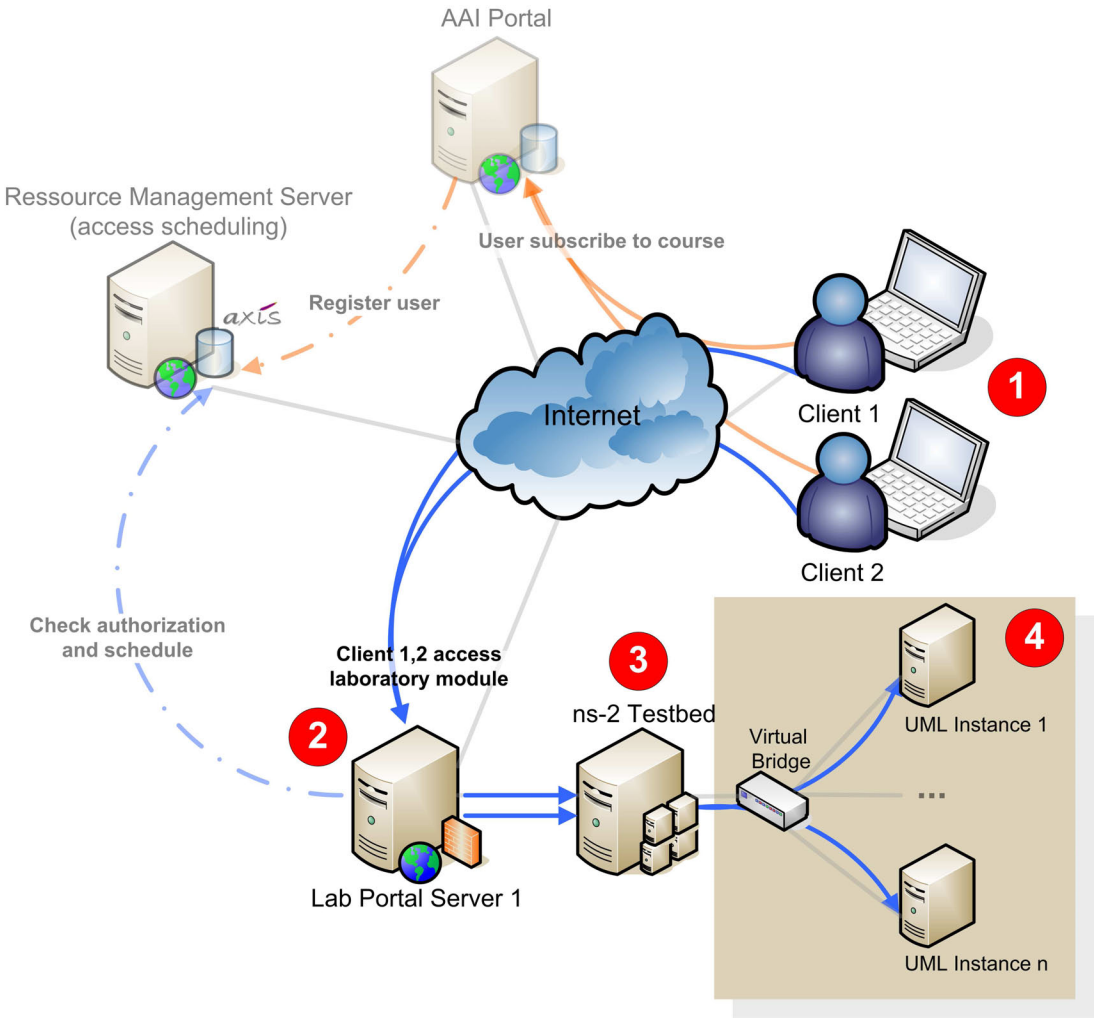


Figure 6.1: ns-2 Platform - showing the different interaction possibilities with the Platform.

he is automatically forwarded to the command line of the assigned UML instance and logged in there as root by automatic SSH key exchange. The login process onto the Laboratory Portal is accomplished by the applet parameters of the Mindterm applet and the fact that the whole system works with AAI single sign login. There is no login action required from the student.

The application VAT4Net and its connection possibilities with the ns-2 Platform are described in detail in Chapter 5, while the details on the Laboratory Portal implementation will be published in another Diploma thesis.



Accessing the ns-2 Platform Step-by-Step

1. Different **Clients** (Students, Learners) want to work on their hands-on sessions.
2. After Clients have the access rights to the **Lab Portal Server**, they are able to start with their work on the laboratory.
3. On the **Testbed** computer the laboratory is running. Imagine it as a classic laboratory with several seats.
4. The **UML Instances** as the client's own laboratory seat is playing the role of a stand-alone computer.

Figure 6.2: ns-2 Platform - System Architecture of the platform with different entities (multi-user access) and its collaboration.

Chapter 7

Module - Implementing Protocols on Network Simulators

7.1 Introduction

In the module ”Implementing Protocols on Network Simulators”, students get a theoretical overview of network simulation concepts. The different concepts are explained and classified. Advantages and disadvantages are discussed. The most common network simulators and details

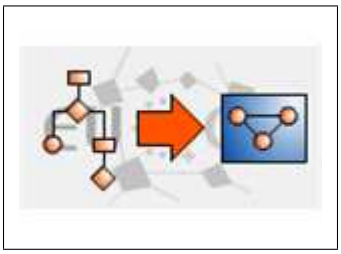


Figure 7.1: Module Logo

of implementing simulations (especially protocols) in Network Simulator (ns-2) are introduced to the student in the theoretical part. The development of a protocol in ns-2 is described step-by-step. In the theory chapters knowledge to successfully pass the hands-on exercises is acquired.

In order to successfully pass the module some basic knowledge is required:

- Basic knowledge of the script language ”Tool command language” (Tcl) (knowledge of variable use, arrays and control structures, handle classes and objects).
- Basic knowledge of C++ (object-oriented concept knowledge, awareness of data types, pointers, structures, functions and control structures).
- Basic knowledge of Linux/Unix environments for the hands-on exercises (handle simple text editor, e.g., vi Editor, main shell usage and basic Makefile editing).

7.2 Structure of the Module

The module is divided into four main parts beginning with the introduction, going on with the theory part, the practical part and at the end the examination part. The chapters are composed as follows:

```
1 Introduction
    1.1 Welcome
    1.2 The Goals and how to Reach Them
    1.3 Module Vicinity
    1.4 My Goals
    1.5 Tips
    1.6 FAQ
2 Theory
    2.1 Theoretical Basics
    2.2 Readings
    2.3 Personal Synthesis
    2.4 Self Test
    2.5 Quiz
3 Knowledge Application/Exploration
    3.1 Introduction
    3.2 Hands-on Session
4 Prove Your Knowledge and Skills
    4.1 Personal Synthesis
    4.2 Final Quiz
    4.3 Survey
```

Listing 7.1: Module Chapter Overview

The content and design of each module chapter is defined in the ”EuQoS Didactics and Design Guide”, [11].

7.3 Module Chapter - Introduction

7.3.1 Structure

The *Introduction* chapter introduces the topic of the module and provides some useful stuff as the Frequently Asked Questions (FAQ) section, tips and tricks, the contact address of the tutor, position of the module within the other modules and a first survey about the learner's expectations (see Listing 7.1).

7.3.2 Content

Motivation

”Implementing and setting up proper network simulation environments eliminates the need of speculation in network planning and developing”, [20].

Rapidly growing networks, for example the Internet, have a great demand on newly developed technologies and on the enhancement of the older ones, but the implementation of new protocols and algorithms without accurate testing scenarios is potentially dangerous and not recommended. Real-world test beds for large topologies are cost-intensive. Therefore network simulations become the preferred tool for system administrators and scientists.

Goals

The section ”Goals” describe the learning goals for the student. They are part of the didactics framework of the EuQoS course. After working through the module ”Implementing Protocols on Network Simulators”, each participant

- can provide a short explanation of the benefit of using simulators in the network research area,
- will be able to set up a network simulation environment to fulfil the particular needs in research activities,
- will know the basics of the most common open source and commercial network simulators and their different concepts and will be able to differentiate between the advantages and disadvantages of a network simulator and choose the right option for different issues,
- will know how to set up an ns-2 simulation scenario and how to run a simulation,
- is able to implement a routing protocol in ns-2,
- and will be competent in analyzing the simulation results and will be able to make of the resulting conclusions.

7.4 Module Chapter - Theory

7.4.1 Structure

The theory part corresponds to Section 2 of the module and is divided into 5 subsections: Theoretical Basics, Readings, Personal Synthesis, Self Test and Quiz. The structure and content of subsection 2.1 - 2.5 (see Listing 7.1) are implemented according to the EuQoS Didactics and Design Guide ([11], page 64 ff.). After working through the theory part each student should be able to pass the practical exercises in Section 3 and 4.

7.4.2 Structure of the Section ”Theoretical Basics”

”Everything that is essential for the laboratory must be mentioned in this section. It can be assumed, that the participants have a strong knowledge in computer science or have a similar technical background”, [11].

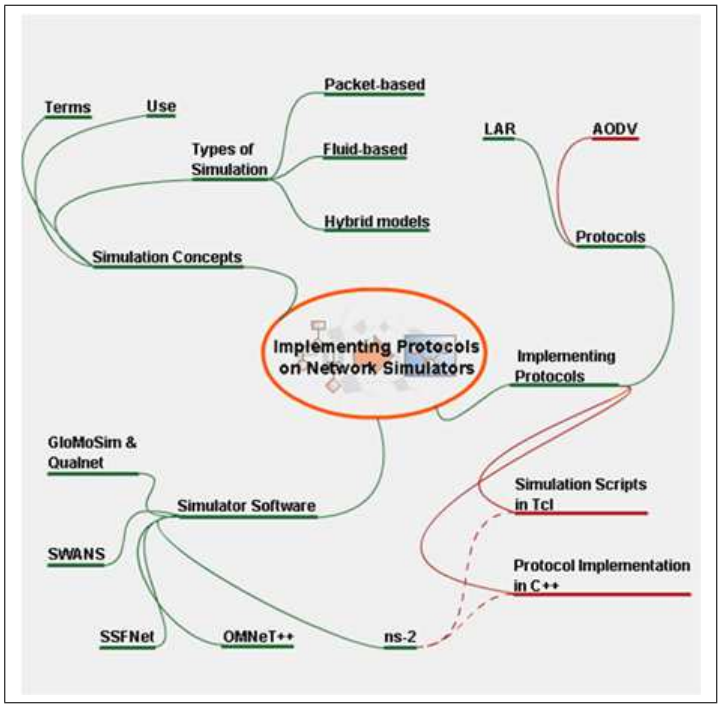


Figure 7.2: Module Content Overview

The theory chapter (Section 2.1 in the module) is divided into seven subsections:

```
2.1 Theoretical Basics
    2.1.1 Main Simulation Terms and Concepts
    2.1.2 Network Simulation
    2.1.3 Use of Network Simulation in Research
    2.1.4 Common Network Simulator Software
    2.1.5 ns-2
    2.1.6 Implementing Protocols with ns-2
    2.1.7 Analysing and Visualising ns-2 Network Simulations
```

Listing 7.2: Module Chapter Overview

7.4.3 Content of the Section "Theoretical Basics"

Main Simulation Terms and Concepts

Computer simulation is an approach to design a model of an actual or theoretical system by substituting real objects, executing the model on a computer and analyzing its output. The most important aspect is the possibility to modify and run a simulation in different ways, keeping the costs low. This newly gained experience about model behavior can help in deploying new

technologies and improving existing ones. Simulation is used in various research departments such as Economics, Social Science, Engineering and Computer Science, in areas ranging from flight simulators to earthquake analysis and weather forecasting.

The Simulation Process

The hardest part in simulation is to model real-world systems into an understandable state for the simulator. Good simulation models are difficult to design and maintain. Therefore, it is important to have various simulation tools.

Validation is the process of assuring that a model simulation generates meaningful answers to the questions being investigated. Models are often approximations or abstractions from reality. Validation is the method to show that these approximations and abstractions are justified. Before validating the required simulation, it is necessary to point out the ”ground truth”: Small-model simulations can be compared with real-world implementations in its detail, whilst large ones can simulate well known expected phenomena in general. The memory consumption and simulation run time of large scale simulations limits the level of detail.

Types of Simulation

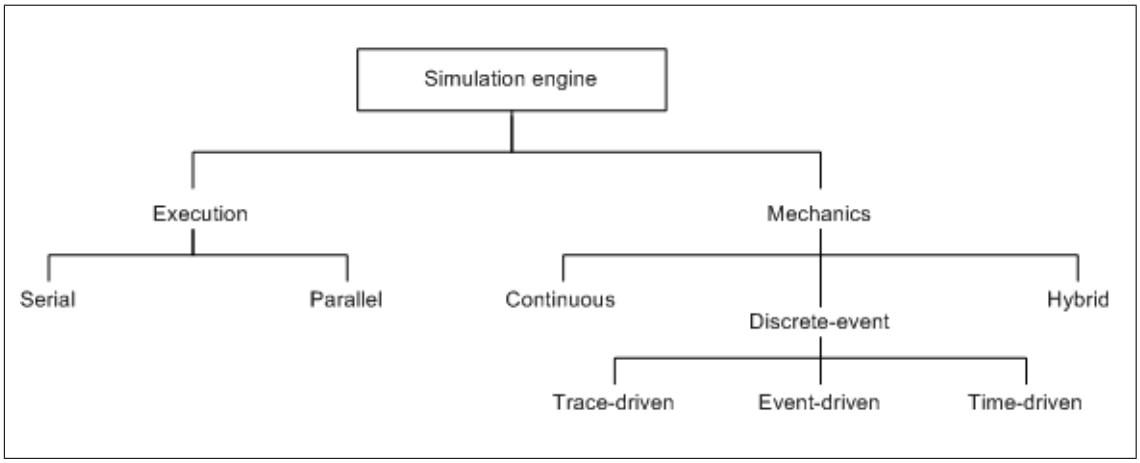


Figure 7.3: Types of Simulation, [21]

Computer simulations are divided into two main categories: discrete-events and continuous simulations (see Figure 7.3). In continuous simulations the quantities are represented by continuous variables, whereas in discrete simulation systems quantities of interest are represented by discrete-valued variables.

The dynamics of a discrete simulation can be considered as a sequence of events at discrete time points. As a third type of simulation, the Monte Carlo simulation is related to discrete-event simulation, which is commonly used to model stochastic systems. ”Monte Carlo

methods are a widely used class of computational algorithms for simulating the behavior of various physical and mathematical systems, and for other computations", [22]. Monte Carlo simulators usually make use of random numbers to model non-deterministic parts in order to simulate the system. Continuous simulations are implemented numerically by differential equations. The simulation program solves all the equations periodically.

For large-scaled simulations hybrid approaches have emerged as viable solutions, where parts are implemented as discrete simulations and other less important parts as continuous simulations. With hybrid simulation strategies it is possible to save a significant amount of computational performance in contrast to discrete-event strategies, especially for simulations with heavy traffic [23].

This simulation types can be executed either in parallel or serially. Parallel simulations have the advantage of faster simulation and shorter execution time, but are more difficult to implement.

Network Simulation

The rapid changes and growth of computer networks has spurred a lot of new development in new protocols and algorithms. New requirements in security, mobile networking, policy management and QoS support issues have become necessary. The main strengths of simulations lie in the ability to imitate complex real world problems and to analyze the behavior of a system.

Simulation in a general network simulation is a hard topic and not easy to handle. A main problem is the fact that a computer network is composed of many nodes such as routers, switches and hosts, making the modeling part of the simulation process a non-trivial task. There are certain decisions to make at the beginning or the simulation process:

1. What are the facts the simulation should show or prove?
2. Which are the important parts that should be investigated?
3. Which simulator provides the best possibilities to model the system?
4. Is the simulation accurate enough in order to use the results for research?

There are a lot of different methods to simulate computer networks, such as the parallel/distributed and the serial execution of simulation, or the tracking method of packets in fluid, packet and hybrid simulation models.

Parallel/distributed versus serial simulations

Parallel or distributed simulation refers to the execution of simulation programs on multiprocessor systems or networks of workstations. The primary goal of parallel/distributed simulations is to obtain higher performance.

The following further goals can be achieved through parallel/distributed simulations:

1. Reduced execution time. By subdividing a large simulation into small simulation parts, the execution time can reduced by a factor equal to the number of processors.

2. Geographical distribution. With the geographical distribution it is possible to simulate travel expenses from one node to another.

3. Fault tolerance, reliability. Use of multiple processors as backup systems.

Packet-, Fluid-Based and Hybrid Model Simulation

Discrete-event simulations are used extensively for protocol design and evaluation. Discrete-event simulations representing a system by a collection of states and a set of events that describe state changes. In this method of modeling networks, the simulator has information about each packet generated and is aware of the path that the packet has to cover. Each packet is tracked individually on each link, in each queue and at each data source and sink. Packet losses are computed deterministically. In the fluid model approach for network's data or packets, flows are

Simulation Model	**Advantage and Disadvantages**
Packet-Based	+ finite changes of state + trivial mathematical models + exact analysis for event + exact modeling of circumstances - limitation of size of the simulation - high memory and processor consumption
Fluid-Based	+ computational efficiency - reduced accuracy - high loss of information - lot of estimated or average values - Ripple Effect

Table 7.1: Comparison of Packet- and Fluid-Based Simulation Model

modeled as fluid flowing through pipes rather than discrete packet instances. A fluid simulator keeps track of the fluid rate changes at traffic sources and network queues. The flows are characterized by a set of mathematical models (often differential equations). Since a large number of packets are abstracted as a single flow, the computational overhead is expected to be relevant. The fluid approach is often used to show bottlenecks in networks when doing flow analysis. A known problem in fluid models is the ripple effect. The ripple effect describes the situation where the propagation of rate changes leads to rate changes in other flows which then need to be propagated [24] which limits scalability [25]. Further problems of packet and fluid-based simulations are listed in the Table 7.1.

Hybrid Models try to connect the positive parts of the two preceding models, the packet and the fluid model. A hybrid model has better computational efficiency than the packet-based model and is more accurate than the fluid based model.

Different Ways to speed up Simulation

A main issue in network simulation research is to find more efficient simulation techniques to speed up simulation processes (see Figure 7.4). Simulation can be speed up by using com-

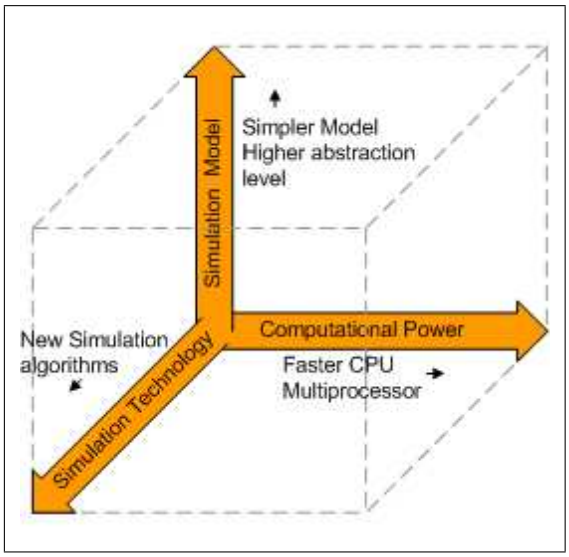


Figure 7.4: Speed up Simulation, [26]

puters with more computational power (faster CPU, multiprocessor system). New simulation algorithms can enhance simulations. A third approach is to reduce the complexity of network simulation by modeling in a more abstract way.

Use of Network Simulation in Research

In the past few years, networks have become too large to do easily real-world experiments and too complicated to analyze them with mathematical methods. Even if analytical methods are available, network simulation is often used to validate the analysis. Furthermore, requirements in network research such as security, mobile networking and quality-of-service have been changing a lot. Simulation has turned out to be an increasingly need and is now used for miscellaneous problems in network research.

Simulations provide methods to investigate newly developed protocols and their behavior, performance, interaction with other protocols, validation, feasibility and to remove points of uncertainty. It is easy and cost saving to evaluate design alternatives in different system configurations. Network simulators are often used when real-world systems are not available

for testing. This fact emerges when network systems are busy or in security mode, which does not allow to test the scenarios.

A further method to use simulations is to combine real systems and virtual simulated systems together. In most cases the real systems are substituted with emulated parts which are cheaper more often than not. Network emulation allows integrating real behavior into the synthetic simulation environment [27].

In Table 7.2 a short comparison of simulation and real world testing of network concerning issues is shown.

	Simulation	**Reality**
Reproducibility	+ easy	- difficult
Network Traffic	- simulated	+ real
Simplification	- high abstraction	+ no abstraction
Scenario creation	+ easy	- complex
Scalability	+ high	- minor
Costs	+ cheap	- costly
Duration	- computational time	+ realtime

Table 7.2: Comparison of Reality and Simulation

Simulation for Education Purposes

Network simulation has obvious advantages over a real network for education purposes. Educational tasks can potentially cause quite a few problems in a live network, but a simulated environment is not endangered by inexperienced operators. In such an environment one has the freedom to experiment, knowing that any problems that mistakes or experiments might cause do not matter. With a simulator, training can be more thoroughly prepared and requires less supervision by the educators.

Common Network Simulator Software

Multi-Protocol network simulators provide substantial benefits such as:

- improved validation of existing protocols
- an infrastructure for developing new protocols
- easier comparison of results

In the module the focus is on the ns-2 network simulator, but nevertheless a few other network simulators are described in a more abstract way in the next few lines.

GloMoSim and Qualnet

GloMoSim [28] is a scalable, discrete event simulation library that support studies of large-scale network models, up to millions of nodes, using parallel and distributed execution. The primary use of GloMoSim is the simulation of wireless and wired networks systems. GloMoSim is a library for the C-based parallel discrete-event simulation language PARSEC. GloMoSim is implemented depending on the OSI layered approach. The standard API specifies parameter exchanges and services between neighboring layers. A number of protocols are available at each layer, and models of these protocols or layers can be used at different levels of detail.

GloMoSim is freely available for education, research or non-profit organizations, whereas QualNet [29] is the commercial version of GloMoSim. QualNet provides a wider range of models and protocols for both wired and wireless networks (local, ad hoc, satellite and cellular) and has a better support for mixed (wired and wireless) network simulations. QualNet is delivered with a very good analysis and visualization tool. It provides trace and statistic file output. A further advantage is the possibility of rapid GUI-based model design.

JiST/SWANS

JiST is a high-performance discrete event simulation engine that runs over a standard Java virtual machine. The JiST system architecture consists of four distinct components: a compiler, a byte code rewriter, a simulation kernel and a virtual machine. One writes JiST simulation programs in plain, unmodified Java and compiles them to byte code using a regular Java language compiler. These compiled classes are then modified, via a byte code-level rewriter, to run over a simulation kernel and to support simulation time semantics. The entire simulation can be run within a standard Java virtual machine (JVM).

SWANS [30] is a scalable wireless network simulator built atop the JiST platform. SWANS is composed of independent software components that form complete wireless network of sensor network configurations following the OSI model. Its capabilities are similar to ns-2 and GloMoSim. Further performance comparisons show that SWANS is able to simulate even larger networks.

Scalable Simulation Framework (SSF) and SSFNet

The quality of a network simulator depends strongly on its scalability. SSF (Scalable Simulation Framework) offers a plurality of different simulation technologies which are enclosed in the simulation kernel and provides an easy programmable interface, the SSF Application Programming Interface (SSF API) for different module development. For module configuration SSF uses its own language, the Domain Modeling Language (DML).

SSFNet [31] is available for network simulation. SSFNet is an enhancement of Raceway, the Java based SSF implementation.

OMNeT++

OMNeT++ [32](Objective Modular Network Testbed in C++) is an object-oriented modular discrete event simulator built on C++ foundations. It offers C++ simulation class library and GUI support (editing and animation). OMNeT++ provides component architecture for models. Components (modules) are programmed in C++ and then assembled into larger components and models using a high-level NEtwork Description language (NED).

The INET Framework on top of OMNeT++ contains implementations IPv4, TCP, UDP protocol implementations and some application models. The list of protocols implemented by INET is growing.

Network Simulator ns-2

ns-2 (ns-2) [33] is an object-oriented discrete-event simulator and includes a large number of applications, protocols, different network types, network elements and traffic models. The development is part of the Virtual InterNetwork Testbed (VINT) project [34]. The project aims in building a network simulator that offers innovative methods and tools. The main idea of ns-2 is to unify the effort of network simulation research.

ns-2 is well-suited for packet switched networks and is used mostly for small scale simulations of queuing algorithms and transport protocol congestion control. It provides support for various implementations of TCP, routing, multicast protocols, link layer and MAC.

The Language Concept

Rather than use a single programming language that defines a monolithic simulation, ns-2 works with different programming models.

There are two class hierarchies used. On the one hand, tasks such as low-level event processing or packet forwarding through a router, requiring high performance and efficiency. These tasks are best served by an implementation in a compile language as C++, called the compiled hierarchy. C++ is fast to run but slow to modify. On the other hand, the interpreted scripting language Tcl (OTcl), called the interpreted hierarchy, provides a flexible and interactive ways to define particular network topologies, dynamic configuration of protocol objects and the specification and placement of traffic sources and also the output form of the simulated model.

In a nutshell C++ implements the simulation kernel, the core parts of high-performance primitives and Tcl scripting language expresses the definition, configuration and control of the simulation.

The simulation is configured, controlled and operated through the interface provided by Tcl's class "Simulator". This main class provides methods for creating and managing the topology, initializing the traffic and choosing the scheduling method.

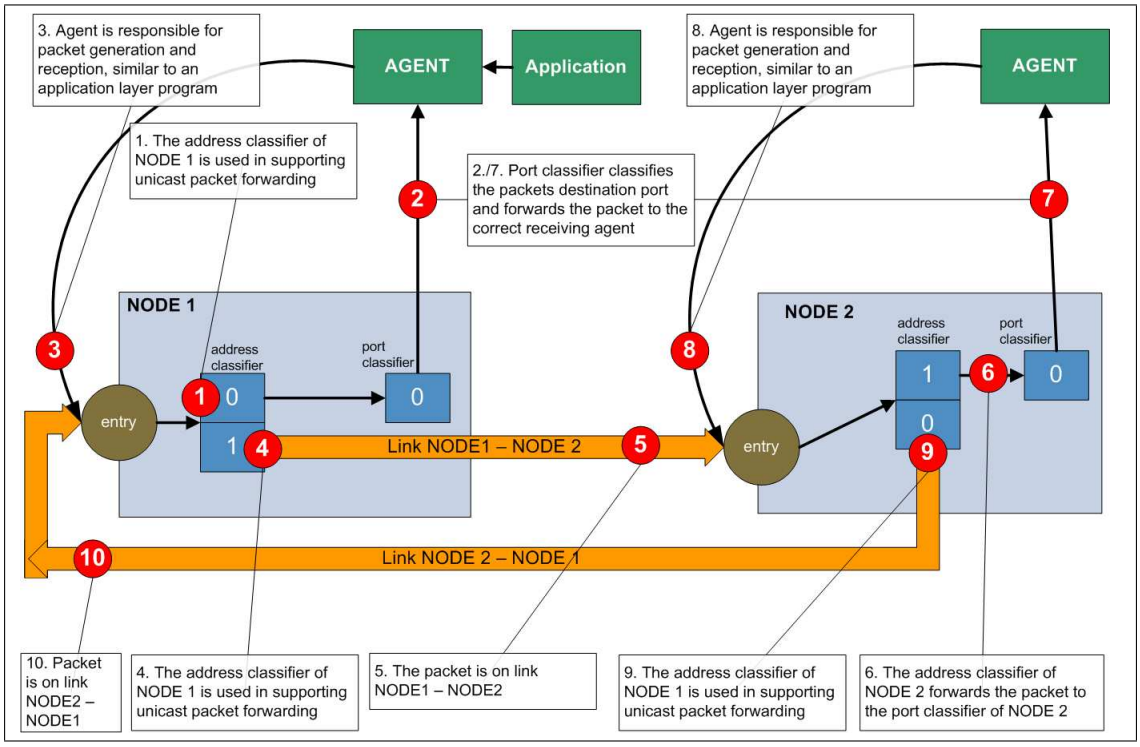


Figure 7.5: Packet Cycle on Node and Link in ns-2

Topology elements are created with the simple primitives "node" and "link". The function of a node is to receive a packet and map it to the relevant outgoing interface (see Figure 7.5). This is done by classifier objects. A unicast node (default node) has an address classifier that does unicast routing and a port classifier. A multicast node has a classifier that differs multicast packets from unicast ones and a multicast classifier that performs multicast routing. Agents are endpoint components of a node where network-layer packets are constructed or consumed.

A link is another major component in ns-2 and is characterized via delay and bandwidth. Links are built as connector objects. The data structure represents a link by a queue connector objects. Such connectors receive packets, perform a function and either send the packet to the next connector or drop the packet.

ns-2 Preliminaries

To execute a simulation, a Tcl script is needed. There are a few basic commands, which will be needed in every simulation. This first script does not execute any simulation process but provides a basic template for further use (see Listing A.1 for the complete file).

A ns-2 simulation Tcl script always starts with a first command, which sets a new instance of the Simulator class of ns-2 (7.3, line 1). Afterwards, the format and output direction of trace files is set by declaring the output file names (7.3, lines 4, 8) and the simulator is set to trace all events into the two different formats (7.3, lines 5, 9).

To set the simulation run time, the simulator instance is assigned to execute the ”finish” procedure after a certain amount of seconds (7.4, line 11). The finish procedure (7.4, lines 13-20) first clears the trace buffer, closes the trace files, executes the Network Animator (nam) and exits the ns application.

```
1 set ns [new Simulator]
3 # Open Trace File
4 set tracefile [open out.tr w]
5 $ns trace-all $tracefile
7 #Open the NAM Trace File
8 set namfile [open out.nam w]
9 $ns namtrace-all $namfile
```

Listing 7.3: ns-2 Preliminaries, Part of A.1

```
11 $ns at 125.0 "finish"
13 proc finish {} {
14     global ns tracefile namfile
15     $ns flush-trace
16     close $tracefile
17     close $namfile
18     exec nam out.nam &
19     exit 0
20 }
22 $ns run
```

Listing 7.4: ns-2 Preliminaries, Part of A.1

When the simulation model is defined completely, the simulator can be run (7.4, line 22).

Nodes,Links and Traffic

The next few steps generates a network topology understandable for the simulator ns-2. The topology will contain four nodes on which one node is the router and two nodes send data to the fourth node through the routing node (see Listing A.2 for the complete file).

Nodes: Lines 22-26 in 7.5 create four instances of the class Node.

```
22 #Create four nodes
23 set n0 [$ns node]
24 set n1 [$ns node]
25 set n2 [$ns node]
26 set n3 [$ns node]
```

Listing 7.5: ns-2 Nodes Links and Traffic, Part of A.2

Links: Lines 28-31 in Listing 7.6 create three instances of class duplex-link connecting the nodes with 10 ms propagation delay and a capacity of 10 Mb/sec for each direction. To define a unidirectional link instead of a bi-directional link, *simplex-link* should be used instead of *duplex-link*. In ns-2 an output queue of a node is defined in the link. The output queue of node n0 is defined as DropTail Queue. Other queue objects derived from the base class Queue are Fair Queuing(FQ), Stochastic Fairness Queuing (SFQ), Deficit Round-Robin (DRR), Random Early Detection (RED) and Class Based Queueing (CBQ) (see Chapter 7.3 in the ns manual [35]) queue objects.

```
28 #Create links between the nodes
29 $ns duplex-link $n0 $n2 1Mb 10ms DropTail
30 $ns duplex-link $n1 $n2 1Mb 10ms DropTail
31 $ns duplex-link $n3 $n2 1Mb 10ms DropTail
```

Listing 7.6: ns-2 Nodes Links and Traffic, Part of A.2

Traffic: There are different traffic sources:
TCP is a connection-oriented protocol. It uses acknowledgements created by the destination to know whether packets are received or not. The TCP connection is defined in line 34 in 7.7 and is connected to node n0 in line 35 in 7.7. TCP's parameters with fixed default values can be changed as an example in line 37 in 7.7 where the standard packet size of 1000 bytes is changed to 552 bytes.

```
34 set tcp0 [new Agent/TCP]
35 $ns attach-agent $n0 $tcp0
36 $tcp0 set fid_ 1
37 $tcp0 set packetSize_ 552
```

Listing 7.7: ns-2 Nodes Links and Traffic, Part of A.2

Once the TCP connection is defined, the FTP application is connected to the TCP connection (7.8, lines 39-41).The next lines (7.8, lines 43-47) define the behavior of the destination node of the TCP connection. The TCPSink agent has an active role in the protocol. It generates acknowledgements packets. In line 47 of Listing 7.8 the two agents are connected each other.

Similar to the TCP setup, the next lines 49-64 in 7.8 create a **UDP** connection between node n1 and node n3. Since UDP is a connection-less protocol it is enough to generate a null agent in line 62 in 7.8. As the application a Constant Bit Rate (CBR) with the parameters set to

a packet size of 1000 bytes and 200 packets per second is selected. The random option in line 59 is a flag indicating whether or not to introduce random noise additionally to the scheduled transmission.

```
39 #Setup a FTP over TCP connection
40 set ftp0 [new Application/FTP]
41 $ftp0 attach-agent $tcp0
43 #Setup
44 set sink0 [new Agent/TCPSink]
45 $ns attach-agent $n3 $sink0
47 $ns connect $tcp0 $sink0
49 # Setup a UDP connection
50 set udp1 [new Agent/UDP]
51 $ns attach-agent $n1 $udp1
52 $udp1 set fid_ 2
54 # Setup a CBR over UDP connection
55 set cbr1 [new Application/Traffic/CBR]
56 $cbr1 attach-agent $udp1
57 $cbr1 set packetSize_ 1000
58 $cbr1 set interval_ 0.005
59 $cbr1 set random_ false
61 # Setup a Null Agent
62 set null1 [new Agent/Null]
63 $ns attach-agent $n3 $null1
64 $ns connect $udp1 $null1
```

Listing 7.8: ns-2 Nodes Links and Traffic, Part of A.2

By scheduling the start and end of FTP and CBR application in lines 66-69 in 7.9, the simulator instance is ready to run a simulation on the defined topology by using preliminary setup from Listing A.1.

```
66 $ns at 1.0 "$ftp0 start"
67 $ns at 4.0 "$ftp0 stop"
68 $ns at 0.5 "$cbr1 start"
69 $ns at 4.5 "$cbr1 stop"
```

Listing 7.9: ns-2 Nodes Links and Traffic, Part of A.2

Wireless Network Simulation in ns-2

The wireless model consists of MobileNodes (see Figure 7.6). Thus, a MobileNode is the basic Node object with added functionalities of a wireless and mobile node that for example can move in a given topology or has the ability to receive and transmit signals to and from a wireless channel. A major difference between MobileNode and Node is that a MobileNode is not connected by links to other mobile nodes.

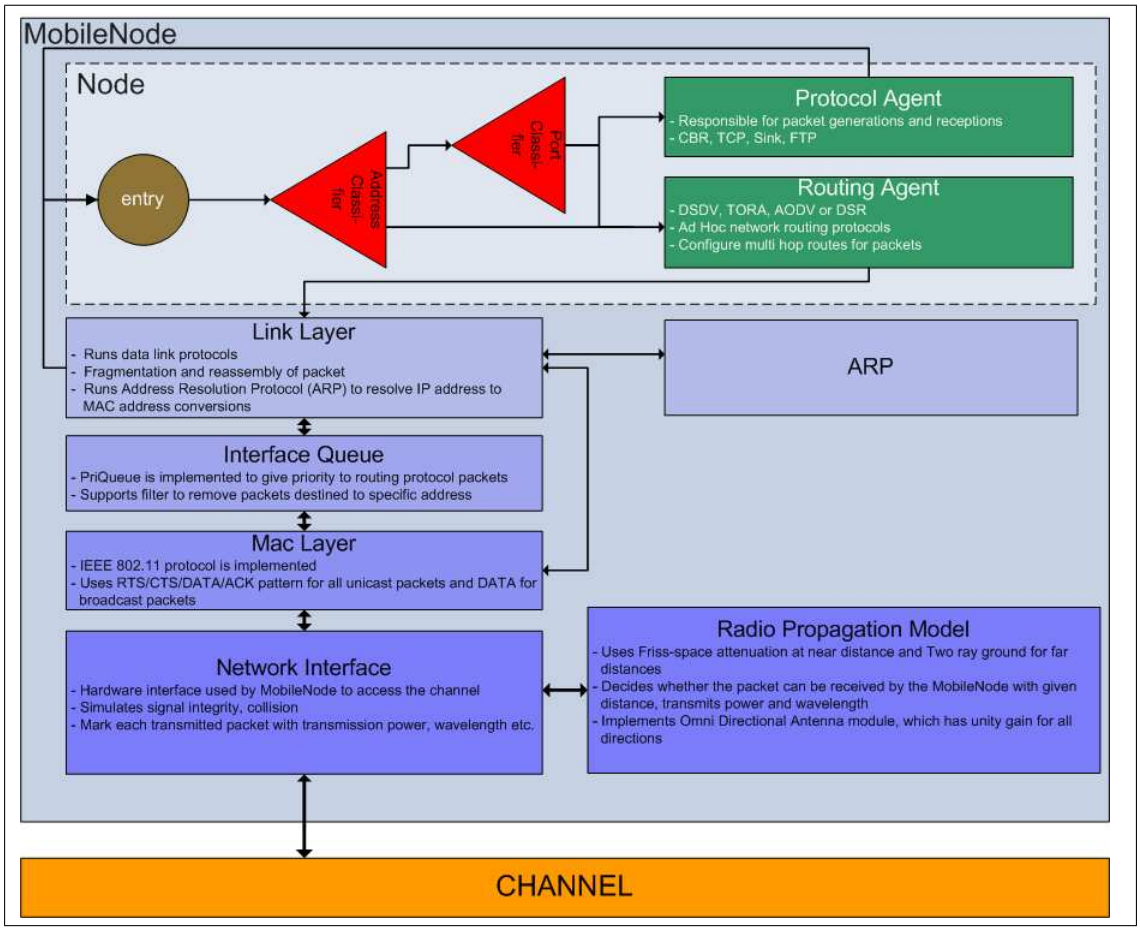


Figure 7.6: MobileNode Architecture

ns-2 has further various modules added to simulate node mobility and wireless networking:

- **Ad-hoc Routing** (Dynamic Source Routing (DSR), Destination-Sequenced Distance-Vector Routing (DSDV), Ad-hoc On-demand Distance Vector (AODV))
- **MAC 802.11**, commonly known by the brand Wi-Fi, denotes a set of Wireless LAN standards.
- **Radio Propagation Model** is an empirical mathematical formulation for the characterization of radio wave propagation.
- **Channel**

A Wireless Simulation Scenario

A wireless simulation scenario is far as similar as a wired scenario, but there must be applied some changes in the setup and additional definitions to the Tcl model file (see Listing A.3 for

the complete file). As a first change the area parameter should be set (see lines 24 and 25 in 7.10) as an instance of the class Topography.

```
24 set topo [new Topography]
25 $topo load_flatgrid 670 670
```

Listing 7.10: ns-2 Wireless Scenario, Part of A.3

The setup of a MobileNode instance requires much more information than a normal wired node (Lines 39-52 in 7.11). A MobileNode is generated with all the given values of adhoc-routing protocol, network stack, channel, topography, propagation model, with wired routing turned on or off (required for wired-cum-wireless scenarios) and tracing turned on or off at different levels (router, MAC, agent). As a new object in wireless network simulations, it requires to handle the General Operations Director (GOD) element. GOD stores the smallest number of hops from one node to another and is automatically generated by the scenario file (Lines 36-37 in 7.11).

```
36 # create god
37 set god_ [create-god 3]
39 # Mobile Node configuration
40 $ns_ node-config -adhocRouting $opt(adhocRouting) \
41        -llType $opt(ll) \
42        -macType $opt(mac) \
43        -ifqType $opt(ifq) \
44        -ifqLen $opt(ifqlen) \
45        -antType $opt(ant) \
46        -propType Propagation/TwoRayGround \
47        -phyType $opt(netif) \
48        -channelType Channel/WirelessChannel \
49        -topoInstance $topo \
50        -agentTrace ON \
51        -routerTrace OFF \
52        -macTrace ON
```

Listing 7.11: ns-2 Wireless Scenario, Part of A.3

Further, to define movement and position information of each node ns-2 provides the Mobile Movement Generator. Detailed instructions can be found in the ns-2 manual ([35],Chapter 16). As an example of output of the Mobile Movement Generator see Listing A.4. The next step is to set up the traffic for the simulation. The easiest way to do this is with the Traffic Generator (see Listing A.5 for example), a supporting tool of ns-2.

Implementing Protocols with ns-2

In the section ”Implementing Protocols with ns-2” the student will learn how to implement a routing protocol for the ns-2 network simulator. The section is very important for the hands-on session that each student has to pass in the practical part of this module.

The section is organized as follows:

- Introduction of routing protocol
- Physical structure of files in ns-2
- Define the packet header
- Implementing the agent
- Run the simulation

The first step is to study the routing protocol which should be implemented. After that it is a good way to think about the packets to add, and the main step will be to implement the routing mechanism into the agent. To introduce protocol implementing, the document ”Implementing a new MANet Unicast Routing Protocol in NS2”, [36] is a required reading for this e-learning module section. In the next few paragraphs, the development of a routing protocol is described in detail.

Location-Aided Routing (LAR)

Location-aided Routing protocol LAR [37] is a advancement of easy flooding protocols for route discovery in a Mobile Ad-hoc Network (MANet). LAR makes use of location information of each node provided by the Global Positioning System (GPS), the currently only fully functional Global Navigation Satellite System (GNSS).

The easiest way to find a route is to flood the network with a route request. Consider a node S that needs to find a route to node D. Node S sends a route request to all its neighbors. A receiving neighbor checks, if it is the destination node or not. When it is not, the node sends the route request to all its neighbors. Each route request includes information about the path. When the destination node D receives the route request message it sends a route reply message to the sender S through the path stored in the route request message. To avoid an increasing number of route request messages, each message owns a sequence number for each node and is able to detect if it has already received the message or not. If the message was already received, there is no more delivery to a neighboring node and the message is discarded.

Figure 7.7 depicts how the flooding algorithm works. The most important aspect is the fact that when a route request message from B to C is faster than a route request message from F to C, the route request message from F to C is discarded and the message from node C with path S-A-B-C is reaching D first. So, the route reply message has the path information S-A-B-C-D and this is the routing information for S.

In addition to the flooding algorithm, LAR uses the location information of Global Positioning System (GPS) to reduce routing overhead. For the protocol it is assumed that at each time the current location of each node is known and the nodes are moving in a 2-dimensional plane.

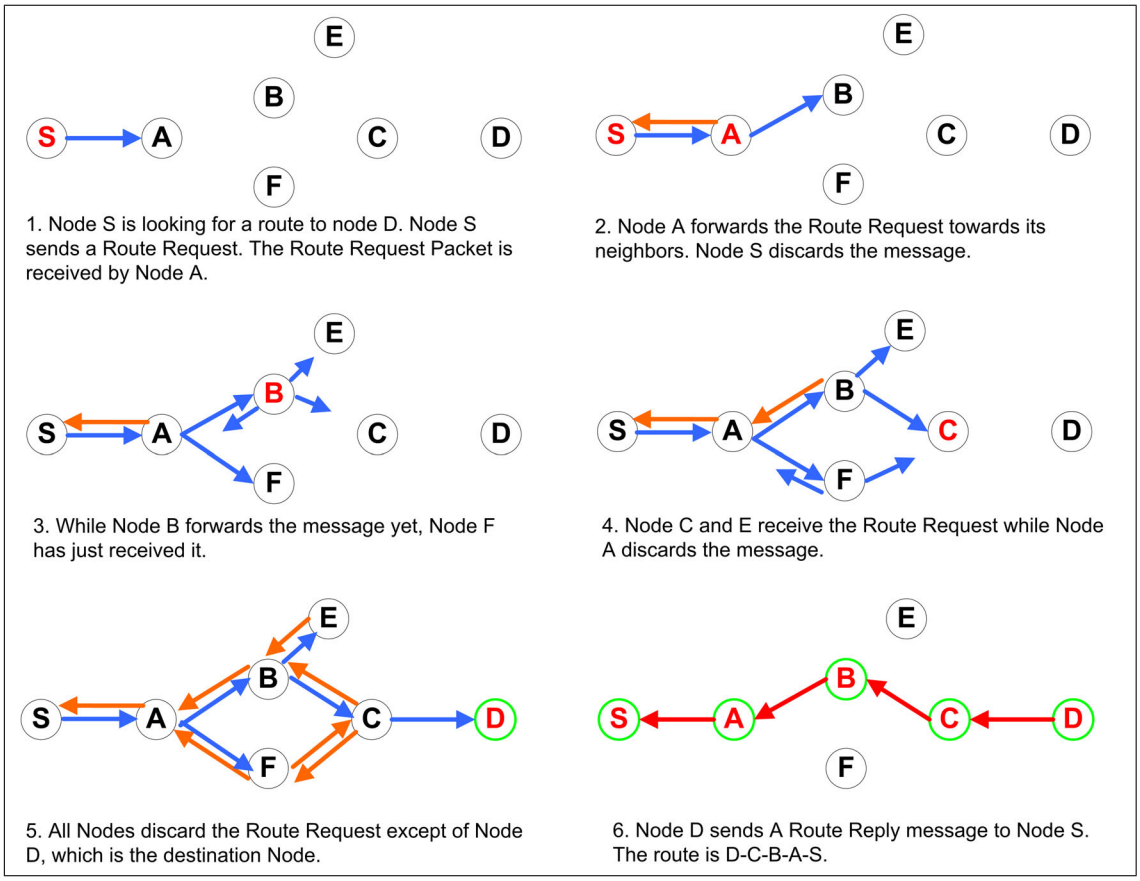


Figure 7.7: LAR Flooding Algorithm

First of all the **Expected Zone** has to be defined for the routing mechanism. Consider node S the sending node and node D the destination. The expected zone of a node D is the area where the node D is expected from the viewpoint of node S. Assume that node S knows the position L of node D at time t0 and the average speed v then node S can determine the position of node D at time t1. The Expected Zone is a circular region centered at location L with radius (see Figure 7.8 (a)): $r = v(t_0 - t_0)$

The Expected Zone is estimated by node S and determines a region that potentially contains D at time t1. If node S does not know a previous location of node D the Expected Zone is enlarged to the area of the entire Ad Hoc network. In this case the LAR algorithm is reduced to the basic flooding algorithm.

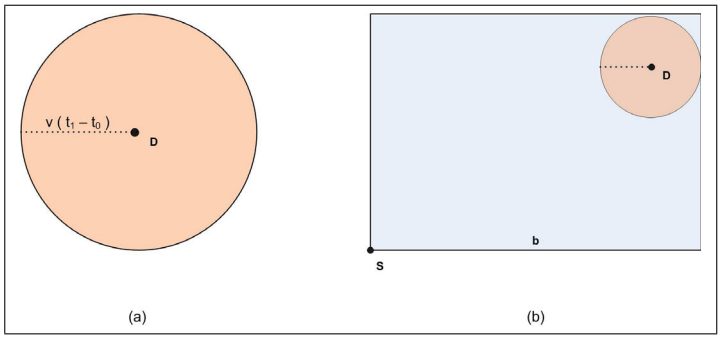


Figure 7.8: LAR (a) Expected Zone, (b) Request Zone

The **Request Zone** is defined by the sender node S, a rectangular area containing node S and including the Expected Zone (see Figure 7.8 (b)). The size of the Request Zone has to be adapted for two reasons:

- If node S does not belong to the expected zone, the path from host S to host D has hosts along the route outside the expected zone. The request zone has to be larger.
- After a time period without discovering a successful route, it is possible to expand the Request Zone.

This shows that the LAR algorithm is essentially identical to the flooding algorithm with the modification that a node that is not in the Request Zone does not forward a route request to its neighbors.

In **LAR Scheme 1** the Request Zone is the smallest rectangle that includes the current location of node S and the Expected Zone around D as described above. The sides of the rectangle are parallel to the X and Y axes.

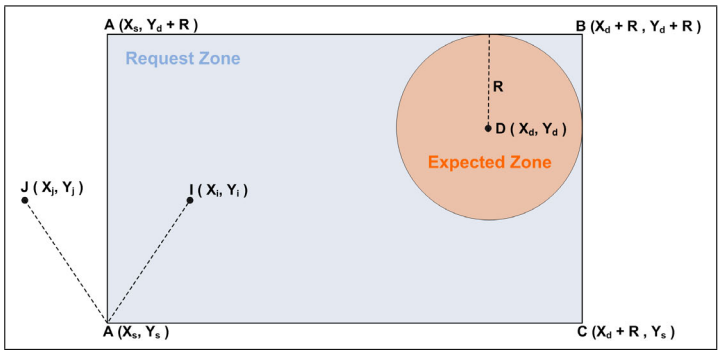


Figure 7.9: LAR Scheme 1

In Figure 7.9 node S requests a route to node D. With stored coordinates (X_d, Y_d) of

node D and the radius calculated as above the Request Zone can be discovered. After node S sends a route request message to its neighbors, each neighbor determines if it is in the Request Zone or not. In Figure 3 node I is in the Request Zone and forwards the route request to its neighbors. Node J is not in the Request Zone and discards the message.

In LAR scheme 1 the Request Zone is specified by the position of node S and the estimated position of node D and this Request Zone is stored in the route request message. In **LAR scheme 2** node S includes 2 different values in the route request (see Figure 7.10):

- The distance $DIST_s$ from S to D where the location of D is the position of D at time t_0.
- The coordinates of D at time t_0.

When a node I receives the route request it calculates the distance $DIST_{(i)}$ from node I to node D.

- For $DIST_{(s)} => DIST_{(i)}$ the node I forwards the request to its neighbors and includes $DIST_{(i)}$ instead of $DIST_{(s)}$ in the route request
- Otherwise, if $DIST_{(s)} < DIST_{(i)}$ the node I discards the route request.

This is repeated until the route request reaches node D. When no route is found it is possible to change the conditions to a * $DIST_{(x)} + b => DIST_{(y)}$ where a and b are parameters.

When node N and node I receive the route request message from node S, both nodes forward the route request to their neighbors, because N and I are closer (means that $DIST_n < DIST_s$ and $DIST_i < DIST_s$) to D than S. When node K receives the message from node I, K discards the message because $DIST_i < DIST_k$.

Scheme 2 of LAR will be the base for the protocol implementation.

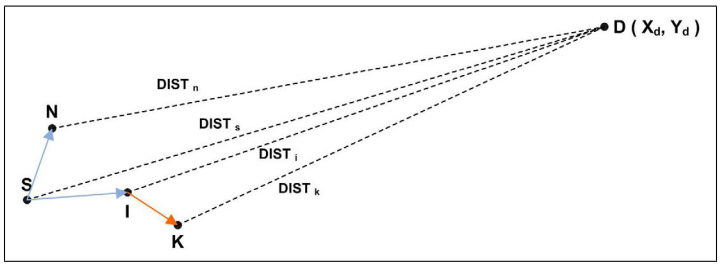


Figure 7.10: LAR Scheme 2

Preparations

In order to start the implementation of a new MANet protocol for ns-2, in the demo case the LAR protocol, it needs some general preliminary steps are required. In the most simply case a

protocol implementation consists of a basic C++ class and a new packet header.

For the LAR example, there will be the following three files to be compiled into ns-2:

- **lar.h** This is the header file where all necessary timers and routing agent are defined
- **lar.cc** In this file you implement timers, the routing agent and Tcl hooks
- **lar_pkt.h** All LAR protocol packets are declared in this file

If continuing with the logical structure of the implementation, it should be implemented an agent by inheriting one from the Agent class of the ns-2 basic implementation. In the ns-2 manual it is stated that: "Agents represent endpoints where network-layer packets are constructed or consumed, and are used in the implementation of protocols at various layers", [35]. The Agent class provides the link to the Tcl interface, so you can control your agent through simulation scripts written in Tcl. A good additional source to the module content would be the manual "Implementing a new MANet unicast routing protocol in NS2", [36].

Furthermore, the LAR protocol uses a new packet type defined in lar_pkt.h, which represent the format of the packet generated through the routing agent. If the protocol has to send packets periodically or after a certain time period it is useful to implement timers inherited from the Timer class of the ns-2 implementation.

Implementing the Packet Types

All data structures, constants and macros related to the new packets are implemented in the packet type definition. The full source code is available in Listing A.6. The new packet header definition enhances the standard class Packet of ns-2.

The attributes of the new packet type are implemented inside the *struct* element (examples in 7.12, lines 13-17) with the associated member functions for the attributes (examples in 7.13, lines 36-39). To define attributes there are special data types defined in ns-2 which can be used, such as *nsaddr_t* for network addresses.

```
13    nsaddr_t    pkt_src_;      // Node which originated this packet
14    u_int16_t   pkt_len_;      // Packet length (in bytes)
15    u_int8_t    pkt_seq_num_; // Packet sequence number
16    char        forwardCode_; // F=flood, S=Scheme 2, U=unset,
17    char        larCode_;      // D=data, R=routing, A=routereply
```

Listing 7.12: LAR Packet Type, Part of A.6

```
36    inline nsaddr_t&    pkt_src()     { return pkt_src_; }
37    inline u_int16_t&   pkt_len()     { return pkt_len_; }
38    inline u_int8_t&    pkt_seq_num(){ return pkt_seq_num_; }
39    inline char&        forwardCode(){ return forwardCode_; }
```

Listing 7.13: LAR Packet Type, Part of A.6

The standard packet header from ns-2 packet.h is included which defines the Packet class of ns-2. The goal is to add the new LAR packet header structure *hdr_lar_pkt* to a packet for information exchange between the objects in the simulation. To do this, two methods are provided to utilize to access the new header in any packet: *offset()* and *access()* in on lines 59-62 in Listing 7.14. When binding the new packet type to the Tcl interface, the new packet header is usable in simulations.

```
59    inline static int& offset() { return offset_; }
60    inline static hdr_lar_pkt* access(const Packet* p) {
61       return (hdr_lar_pkt*)p->access(offset_);
62    }
```

Listing 7.14: LAR Packet Type, Part of A.6

The Routing Agent - Header File

Once the packet header used by the routing protocol is defined, we define the header file for our routing agent. The lines 4-13 in Listing 7.15 (see Listing A.7 for complete file) include some required files.

```
4  #include "lar_pkt.h"
5  #include "trace.h"
6  #include "object.h"
7  #include "agent.h"
8  #include "packet.h"
9  #include "ip.h"
10 #include "mobilenode.h"
11 #include "timer-handler.h"
12 #include "random.h"
13 #include "classifier-port.h"
```

Listing 7.15: LAR Header File, Part of A.7

Some of them are described below:

- **lar/lar_pkt.h** definition of new packet(s) used by the routing agent.
- **common/agent.h** base Agent class, our routing agent inherits from it.
- **common/timer-handler.h** TimerHandler base class, required for custom timers.
- **trace/trace.h** possibility to write custom simulation results to a trace file.
- **tools/random.h** pseudo random numbers.
- **classifier/classifier-port.h** definition of the PortClassifier which is used to pass packets to upper layers (e.g. UDP).
- **common/ip.h** access the ip header.

Each routing agent maintains its own routing table and state. This may be implemented by encapsulating all this information in a separate class or by storing it in the routing agent itself. In the LAR example this information is stored in the agent itself. Thus we define a structure and a associative container of these structures, which represents our routing table (7.16, lines 27-37 and 7.18, line 56). We maintain a counter for sequence number to identify packets (7.19, line 72). We also store the address of the node the routing agent is attached to (7.19, line 73).

```
27 typedef struct
28 {
29     double x;
30     double y;
31     char    method;   //flood or scheme 2
32     nsaddr_t   route[maxRouteLength];
33     int hops;
34     double timeOfLastRequest;
35     bool requestPending;
36     bool noRoute;
37 } larDestType;
```

Listing 7.16: LAR Header File, Part of A.7

A PortClassifier object is declared to handle packets destined for the node itself (7.18, line 60). The PortClassifier gives the packets to the corresponding agents in the upper layer. For getting position and other information of the node, especially used by the location-aided routing protocol, the agent also stores a reference to its node (7.18, line 57). If the routing agent needs a timer, for instance to check periodically for neighbors, a simple example timer is declared (7.17, lines 40-48). This timer stores a reference to the routing agent, and inherits a method *expire()* from TimerHandler, which must be overloaded.

```
40 class LAR_PktTimer : public TimerHandler {
41    public:
42       LAR_PktTimer(LAR* agent) : TimerHandler() {
43          agent_ = agent;
44       }
45    protected:
46       LAR * agent_;
47       virtual void expire(Event* e);
48 };
```

Listing 7.17: LAR Header File, Part of A.7

```
56       map<nsaddr_t, larDestType, less<int> > routeTable;
57       MobileNode *node;
59    protected:
60       PortClassifier*          dmux_;
```

Listing 7.18: LAR Header File, Part of A.7

The routing agent also inherits two methods, *recv()* and *command()* (7.19, lines 75-76), from the Agent base class, which must be overwritten. The *recv()* method is called whenever the agent receives a packet, while the *command()* function is the place where the Tcl commands are linked to our compiled implementation. The constructor of LAR class is declared and it receives an identifier used as the routing agent's address as the argument (7.19, line 74).

```
72       u_int8_t            seq_num_;
73       nsaddr_t            ra_addr_;
74       LAR(nsaddr_t);
75       void   recv(Packet*, Handler*);
76       int    command(int, const char*const*);
```

Listing 7.19: LAR Header File, Part of A.7

The Routing Agent - Source File

Now we can implement the routing agent. First we must bind our new routing agent class to Tcl, so we can use it in the simulations. This is done similar like the binding of the new packet type. The class constructor and the implementation of the function *create()*, are shown in Listing 7.20 (lines 22-26) , which returns a LAR agent instance (see Listing A.8 for complete source).

In order to have a valid timer we have to implement the expire method. An new event is generated (Listing 7.20, lines 29-35) and is registered to the scheduler. A new event will occur five seconds later when the expire method is invoked from the scheduler.

```
20 static class LARClass: public TclClass {
21     public:
22         LARClass() : TclClass("Agent/LAR") {}
23         TclObject* create(int argc, const char*const* argv) {
24             assert(argc==5);
25             return (new LAR((nsaddr_t)Address::instance().str2addr(argv
                   [4])));
26         }
27 } class_rtProtoLAR;
29 void LAR_PktTimer::expire(Event* e) {
30     agent_->reset_lar_pkt_timer();
31 }
33 void LAR::reset_lar_pkt_timer() {
34     pkt_timer_.resched((double)5.0);
35 }
```

Listing 7.20: LAR Source File, Part of A.8

The next part is the constructor implementation with *PT_LAR* as the argument to the base class constructor. In the same line 37 (Listing 7.21) we create the *LAR_PktTimer* object and initializing the *dmux_* pointer with 0. To access variables via Tcl, they must be bound in the constructor. This is exemplarily done in line 40 (Listing 7.21) but not required by the LAR implementation. Bound variables like *accessible_var_* have to be declared in the header file as well. In order to access those variables from Tcl space you use *Agent/LAR set accessible_var_ true* in your simulation Tcl script.

```
37 LAR::LAR(nsaddr_t id) : Agent(PT_LAR), dmux_(0), pkt_timer_(this) {
38     ra_addr_ = id;
39     node = NULL;
40     //bind_bool("accessible_var_", &accessible_var_);
41 }
```

Listing 7.21: LAR Source File, Part of A.8

The *command()* method is inherited from the Agent class and contains the commands used in the Tcl scripts. The variable *argc* contains the numbers of arguments for the Tcl instruction and *argv* contains the arguments as an array. The first argument contains *cmd* and the second contains the requested operation. All additional arguments are the arguments used by that command. Every processed command has to be terminated by returning *TCL_OK*, if all goes fine, or *TCL_ERROR*, in case of an error.

For routing agents the commands *start*, *port-demux* and *tracetarget* have to be implemented. The command *start* sets up all necessary actions to start the routing agents operation, e.g., the timer can be started when required (Listing 7.22, line 46). The PortClassifier object can be set by invoking the *port-demux* instruction in Tcl ((Listing 7.22, lines 50-57). The last mandatory command is *tracetarget* which sets the Trace object (Listing 7.22, lines 58,63). The Trace object can be used for custom trace file output.

Own commands may be implemented in the *command()* method if required. As example the lines 64-67 (Listing 7.22) shows the *node* command, which sets the MobileNode the routing agent is placed. If the command is unknown by the *command()* implementation of the subclass, it must be delegated to the super class (Listing 7.22, line 69).

```
43 int LAR::command(int argc, const char*const* argv) {
44    if (argc == 2) {
45       if(strcasecmp(argv[1], "start") == 0){
46          pkt_timer_.resched(0.0);
47          return TCL_OK;
48       }
49    } else if (argc == 3) {
50       if(strcmp(argv[1], "port-dmux") == 0 ) {
51          dmux_ = (PortClassifier *)TclObject::lookup(argv[2]);
52          if (dmux_ == 0) {
53             fprintf (stderr, "%s: %s lookup of %s failed\n", __FILE__
                  ,
54             argv[1], argv[2]);
55             return TCL_ERROR;
56          }
57          return TCL_OK;
58       } else if(strcmp(argv[1], "tracetarget") == 0 ) {
59          logtarget_ = (Trace*)TclObject::lookup(argv[2]);
60          if(logtarget_ == 0) {
61             return TCL_ERROR;
62          }
63          return TCL_OK;
64       } else if (strcasecmp (argv[1], "node") == 0) {
65          node = (MobileNode*)TclObject::lookup(argv[2]);
66          return TCL_OK;
67       }
68    }
69    return (Agent::command(argc, argv));
70 }
```

Listing 7.22: LAR Source File, Part of A.8

As mentioned above the *recv()* method is the next method to be implemented. It is inherited from the agent base class. It receives not only incoming data or routing packets, but also data packets generated by upper layer agents on this node. In the LAR example it has to be checked first if the type of the packet is LAR, then the packet is passed to the *recv_lar_pkt(pkt,h)* method. For all other packets it has to be checked first if they are self-sent (7.23, line 82). If the number of forwards is not equal to zero, the packet made a loop and the packet should be dropped. If the packet has been generated within the node, we should add the headers and tails of the network layer (7.23, line 88) to the packet's length. It is assumed that LAR works over IP. If the packet's address is not *IP_BROADCAST* a route to the destination node should be found (7.23, line 96). The method *route_resolve(pkt, h)* simply looks for a valid route in the routing table and if it

finds one, it forwards the data according to that route. If not, the routing agent starts the route discovery process by sending a route request. After gaining a valid route the node forwards the data. In the case of the packet's destination address being *IP_BROADCAST* the data has only to be forwarded (7.23, line 98).

```
82      if(ih->saddr() == ra_addr()) {
83          //there exists a loop -> drop the packet
84          if(ch->num_forwards() > 0) {
85              drop(pkt, DROP_RTR_ROUTE_LOOP);
86              return;
87          } else if(ch->num_forwards() == 0) {
88              ch->size() += IP_HDR_LEN;
89          }
90      } else if (--(ih->ttl_) == 0) {
91          drop(pkt, DROP_RTR_TTL);
92          return;
93      }
95      if ((u_int32_t)ih->daddr() != IP_BROADCAST) {
96          route_resolve(pkt, h);
97      }else{
98          forward_data(pkt, h, (larDestType*) 0);
99      }
```

Listing 7.23: LAR Source File, Part of A.8

The next method *recv_lar_pkt* handles the LAR routing packets received by *recv()*. Its implementation heavily depends on the concrete routing protocol. But the following scheme may be used as a general template. Lines 103-105 (Listing 7.24) get the common header, IP header and LAR header as usual. Afterwards we verify that the source and destination ports are set to *RT_PORT* (7.24, lines 107-108). This port is attached to 255 for routing agents. Lines 113-116 (7.24) check if the same packet has been received or sent before, and dismiss the packet when true. Finally the resource is released (7.24, line 120). In the LAR example this method also generates route replies if it is the destination node or already knows a valid route to the destination. It forwards the routing packets to the next neighbors if the node is inside the forwarding region. It also updates its routing table if a route reply was received and the node was the origin of the route request, or simply forward a route reply back to the source.

```
103     struct hdr_cmn* ch = hdr_cmn::access(pkt);
104     struct hdr_ip* ih   = hdr_ip::access(pkt);
105     struct hdr_lar_pkt* larhdr = hdr_lar_pkt::access(pkt);
107     assert(ih->sport() == RT_PORT);
108     assert(ih->dport() == RT_PORT);
110     node->update_position();
111     larhdr->newPkt() = false;
```

```
113     if(this->duplicateRcvdPacket(ch->uid_) || this->
            duplicateSentPacket(ch->uid_)) {
114         Packet::free(pkt);
115         return;
116     }
118     /* processing of LAR routing packets */
120     Packet::free(pkt);
```

Listing 7.24: LAR Source File, Part of A.8

The *forward_data* method forwards the data packets received by the routing agent. If the *ttl_* variable equals zero the packet is dropped (7.25, lines 128-131). If the packet is an incoming one and the destination address is set to the node itself or broadcast, then the node's *dmux_* is needed to accept the incoming packet (7.25, lines 133-136) and passes it to the corresponding agent. Otherwise, the packet has to be send out after setting its header fields (7.25, line 138 ff.).

```
128     if (iphdr->ttl_ == 0) {
129         drop(p, DROP_RTR_TTL);
130         return;
131     }
133     if(cmnhdr->ptype() != PT_LAR && cmnhdr->direction() == hdr_cmn::UP
            && ((u_int32_t)iphdr->daddr() == IP_BROADCAST || iphdr->daddr
            () == ra_addr())) {
134         dmux_->recv(p,h);
135         return;
136     }
138     if (rt) {
139         cmnhdr->direction() = hdr_cmn::DOWN;
140         cmnhdr->addr_type() = NS_AF_INET;
141         cmnhdr->next_hop_ = rt->route[1];
142     }else{
143         // Broadcst message
144         cmnhdr->direction() = hdr_cmn::DOWN;
145         cmnhdr->addr_type() = NS_AF_NONE;
146     }
```

Listing 7.25: LAR Source File, Part of A.8

Needed Changes in ns-2

If the implementation is finished, it has to be integrated into ns-2. In order to do this, there are some files which have to be changed.

In the LAR routing agent a **new packet type PT_LAR** is introduced. It has to be defined in the file <nsrootdirectory>/common/packet.h. The packet type has to be added into the structure *enum_packet_t* as it is done in Listing 7.26 in line 6. In the same file in line 17 (Listing 7.27), the string definition for trace files for this packet type will be set.

```
enum packet_t {
    PT_TCP,
    PT_UDP,
    PT_CBR,
(...)
    PT_LAR,
    PT_NTYPE // This MUST be the LAST one
};
```

Listing 7.26: Required changes in ns-2 packet.h, Part of A.9

```
p_info() {
    name_[PT_TCP]= "tcp";
    name_[PT_UDP]= "udp";
    name_[PT_CBR]= "cbr";
(...)
    name_[PT_LAR]="LAR";
    name_[PT_NTYPE]= "undefined";
}
```

Listing 7.27: Required changes in ns-2 packet.h, Part of A.9

In a next step some **Tcl files** have to be adapted. First the new packet type is added, gets the default values for bound variables and the functions for creating wireless nodes with LAR as routing agent are provided.

For proper working of the new routing protocol, there are also some changes needed in the Tcl scripts. Because simulations are invoked and controlled through Tcl, the interpreter hierarchy has to know how to use the new functionalities written in C++ as it is done in Listing 7.28 in line 5.

```
foreach prot {
    AODV
    ARP
(...)
    LAR
    NV
}
```

Listing 7.28: Required changes in ns-2 ns-packet.tcl, Part of A.10

First, all bound variables must have a default value. This is done by defining them in tcl/lib/ns-default.tcl.

```
Agent/LAR set accessible_var_ true
```

Then in the file tcl/lib/ns-packet.tcl the new packet header has to be added to the common ns-2 packet so it can be used.

As a further step some changes have to be done in the tcl/lib/ns-lib.tcl script, that the wireless nodes can use the newly added routing protocol. This is done by adding a new method which creates the new routing agents and adds them to the nodes. And in the already existing method to create wireless nodes, there must be done one small change to make the new routing agent known by this method as it is shown in Listing 7.29 in lines 12-14 and in Listing 7.30 in lines 26-32.

```
12    LAR {
13        set ragent [$self create-lar-agent $node]
14    }
```

Listing 7.29: Required changes in ns-2 ns-lib.tcl, Part of A.12

```
26 Simulator instproc create-lar-agent { node } {
27    # Create Lar routing agent
28    set ragent [new Agent/LAR [$node node-addr]]
29    $self at 0.0 "$ragent node $node"
30    $node set ragent_ $ragent
31    return $ragent
32 }
```

Listing 7.30: Required changes in ns-2 ns-lib.tcl, Part of A.12

The last step of protocol implementation is binding the new agent into the ns-2 implementation. First, the Makefile of ns-2 should be adapted as it is shown in Listing 7.31 in line 5.

```
1 OBJ_CC = \
2     tools/random.o tools/rng.o tools/ranvar.o
3     common/misc.o common/timer-handler.o \
4 (...)
5     lar/lar.o lar/lar_pkt.o \
6 (...)
7     $(OBJ_STL)
```

Listing 7.31: Required changes in ns-2 Makefile, Part of A.11

After adapting the Makefile the source code has to be compiled into ns-2 with the following commands:

```
1 [ns-2]$ make clean
2 [ns-2]$ make depend
3 [ns-2]$ make
4 [ns-2]$ make install
```

Now the LAR routing agent is usable in wireless simulation scenarios

Analyzing and Visualizing ns-2 Network Simulations

In this section, methods for analyzing and visualizing ns-2 network simulations are described. The are various trace formats used by the ns-2 network simulator. The mostly used are:

- Normal trace format
- Wireless trace format (New Wireless Trace and Old Wireless Trace.)
- NAM trace format

Normal Trace Format

The first field describes the type of event taking place at the node and can be one of the five following types:

Event	Type	
receive	r	packet receive event at the destination node of a link
drop	d	packet drop (packet delivered to drop-target)
error	e	simulation error
enqueue	+	simulation error
dequeue	-	a packet departure (usually at a queue)

The additional fields are described in Table 7.3, while a example of an extract of a trace file can be viewed in Listing A.13.

Event	Abbreviation	Type	Value
		%g %d %d %s %d %s %d %d. %d %d. %d %d %d	
Normal Event	r: Receive d: Drop e: Error +: Enqueue -: Dequeue	double	Time
		int	Source Node
		int	Destination Node
		string	Packet Name
		int	Packet Size
		string	Flags
		int	Flow ID
		int	Source Address
		int	Destination Address
		int	Sequence Number
		int	Unique Packet ID

Table 7.3: Normal Trace Format - Field Definitions

New and Old Wireless Trace Format

The first field of the **New Wireless trace format** describes the type of event taking place at the node and can be one out of the four types:

Event	**Type**	
Send	s	a packet departure
Receive	r	packet receive event at the destination node of a link
Drop	d	packet drop (packet delivered to drop-target)
Forward	f	packet forward

The first letter of the following tags designate the type of the flag:

- N: Node
- I: IP Level Packet
- H: Next Hop
- M: MAC Level Packet
- P: Packet specific

Depending on the packet type, there are many additional flags used by the New Wireless trace format (see [35]). The additional fields are described in Table 7.4, while a example of an extract of a trace file can be viewed in Listing A.14.

The **Old Wireless trace format** is described in detail in ”The ns manual”, [35].

Additional trace formats for wireless simulations are:

- AODV routing protocol trace formats
- DSDV routing protocol trace formats
- DSR routing protocol trace formats
- TORA routing protocol trace formats
- Mobile node movement and energy trace formats

Event	Abbreviation	Flag	Type	Value
Wireless Event	s: send r: receive d: drop f: forward	-t	double	Time (* For Global Setting)
		-Ni	int	Node ID
		-Nx	double	Node X Coordinate
		-Ny	double	Node Y Coordinate
		-Nz	double	Node Z Coordinate
		-Ne	double	Node Energy Level
		-Nl	string	Network trace Level
		-Nw	string	Drop Reason
		-Hs	int	Hop source node ID
		-Hd	int	Hop destination Node ID, -1, -2
		-Ma	hexadecimal	Duration
		-Ms	hexadecimal	Source Ethernet Address
		-Md	hexadecimal	Destination Ethernet Address
		-Mt	hexadecimal	Ethernet Type
		-P	string	Packet Type (arp, dsr, imep, tora, etc.)
		-Pn	string	Packet Type (cbr, tcp)

Table 7.4: New Wireless Trace Format - Field Definitions

NAM Trace Format

In Network Animator (NAM) trace format there are more events traced as in the other formats. The NAM trace format is very extensive and is used for the Network Animator (NAM). Detailed information can be found in "The ns manual" [35] or can be generated with the command *"nam -p"* if NAM is installed. As an example the details of the event Link is listed in Table 7.6 and an extract of a NAM Trace File can be found in Listing A.15.

Event	Type
Dummy Event	T
Node	n
Link	l
Packet	h: Hop, r: Receive, d: Drop, +: Enqueue, -: Dequeue
Session	E: Enqueue, D: Dequeue, P: Drop
Agent	a

Table 7.5: NAM Trace Format - Events

Event	Abbreviation	Flag	Type	Value
Link		-t	time	Time
		-s	int	Source ID
		-d	int	Destination ID
		-r	double	Transmission Rate
		-D	double	Delay
		-h	double	Length
		-O	orientation	Orientation
		-b	string	Label
		-c	color	Color
		-o	color	Previous Color
		-S	string	State (UP, DOWN)
		-l	string	Label
		-L	string	Previous Label
		-e	color	Label Color

Table 7.6: NAM Trace Format - Field Definitions

Analyzing Methods

For each analysis of a ns-2 network simulation, the trace file is the basic input of data. This simple text file can be processed by various programming languages which are able to read text files, like AWK, Perl or Java. For this paragraph Perl is selected as the processing language due to the fact, Perl is strong in performing string handling and on the other hand Java is used to parse trace files within the application VAT4Net.

As an example of analysis, a very simple case is chosen. The simulation topology looks as follows:

- four nodes, while two nodes act as sending nodes, one node is the receiving node and one node acts as router
- the agent on the two sending nodes is UDP and the traffic source is CBR (Constant Bit Rate)

The output files of this simulation are generated once in Normal trace format and otherwise in the NAM trace format.

There are many characteristics a network simulation can show such as routing overhead, throughput or packet loss rate. The example will show the calculation of packet end-to-end delay in the simulated network. The end-to-end delay is defined as time between the point in time, the source want to send a packet and the moment the packet reaches its destination.

For processing this calculation on the trace file, a Perl script is implemented (see Listing A.16 for complete file). For the end-to-end Delay calculation of each packet, the time stamp

is required when the source wants to send the packet and the time stamp when the destination receives the packet. The difference between these two time stamps is the end-to-end Delay for the packet. The time stamp when the packet is send is the time stamp of the first appearance of the packet in the trace file (Listing 7.32, lines 40-44). The time stamp is stored in a hash with key as packet id and value as time stamp. Dropped packets are cleared from the hash sets (Listing 7.32, line 35-38). When a packet is received by a node the time stamp is stored in a second hash set with the same key as the first hash set. After running through the trace file line-per-line the end-to-end delay is calculated (Listing: 7.32, lines 47-50).

```
29      if(exists $packetSend{$word[11]}){
30          if($word[0] eq "r")
31          {
32              $packetReceived{$word[11]}=$word[1];
33              $destination{$word[11]}=$word[3];
34          }
35          if($word[0] eq "d") {
36               delete $packetReceived{$word[11]};
37               delete $packetSend{$word[11]};
38          }
39      }
40      else
41      {
42          $packetSend{$word[11]}=$word[1];
43          $source{$word[11]}=$word[2];
44      }
45  }
47  #calculate end to end delay
48  foreach my $key1 ( keys %packetSend ) {
49      $endToEndDelay{$key1}=$packetReceived{$key1}-$packetSend{$key1};
50  }
```

Listing 7.32: Perl Script for end-to-end Calculation, Part of A.16

The last step is to prepare the data for further processing. In the next paragraph it is shown how to visualize the data with gnuplot. So it is eligible to store the data in a file in two columns, first column with the value of the time step when the packet is sent and in the second column the end-to-end delay. The output is generated (7.33, lines 55-62).

```
55  sub printToFile(){
56      open (FILE, ">endToEnd_v2.dat");
57      flock (FILE, 2);
58      foreach my $key1(sort{$packetSend{$a} cmp $packetSend{$b} } keys %
            packetSend ) {
59          print FILE "$packetSend{$key1} $endToEndDelay{$key1}\n";
60       }
61      close FILE;
62  }
```

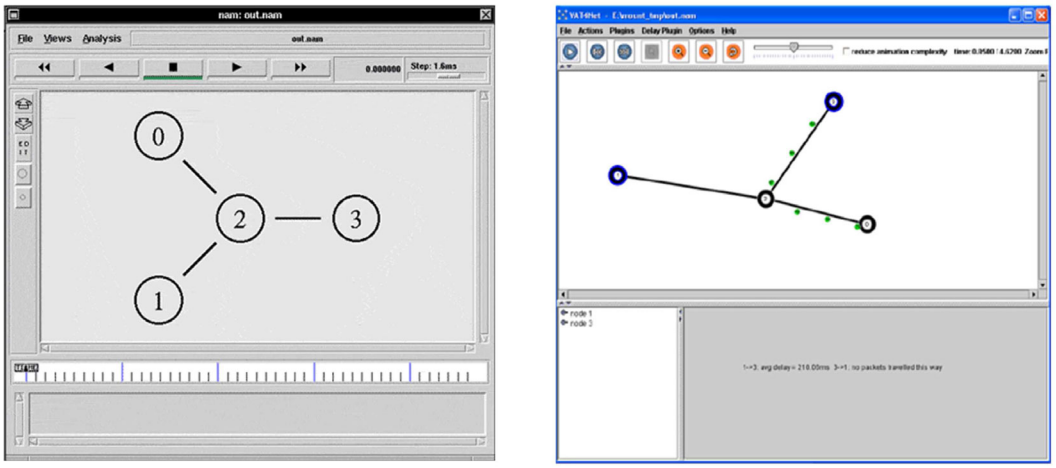
Listing 7.33: Perl Script for end-to-end Calculation, Part of A.16

Visualizing Methods

To visualize network dynamics on the basis of ns-2 trace files, there are several tools available.

NAM (Figure 7.11(a)) is a Tcl/TK based animation tool for viewing network simulation traces and real world packet traces. It supports topology layout, packet level animation and various data inspection tools. Additional information about NAM is available in the ns-manual [35]. An x-window system is required to install and run NAM.

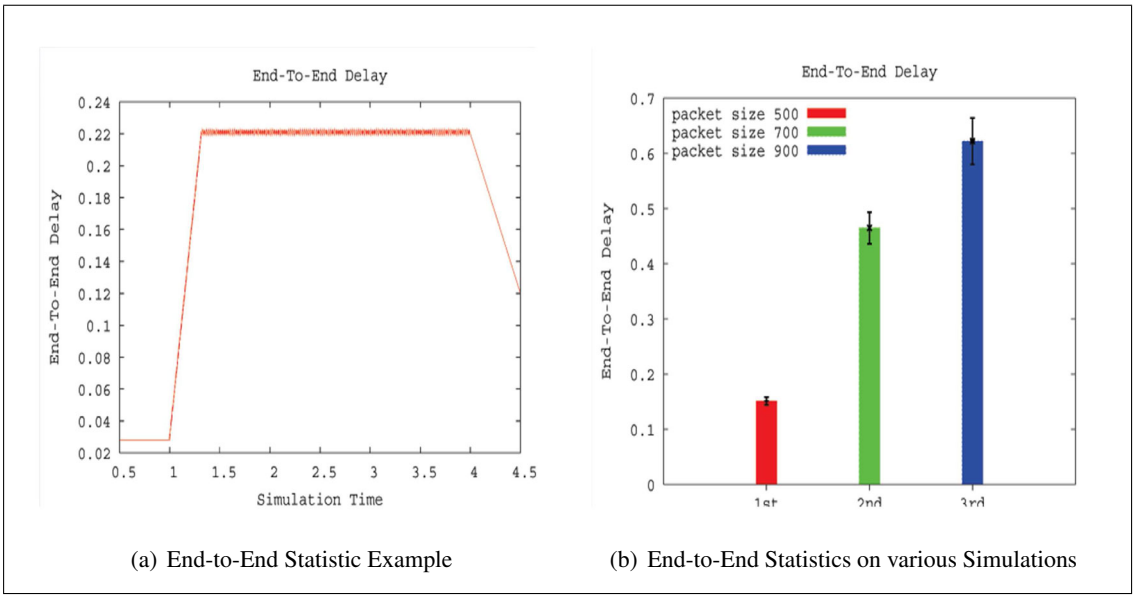
VAT4Net(Figure 7.11(b)) is a Java implementation which runs either in Applet or stand-alone mode. It has been developed during this thesis. Details on VAT4Net can be found in Chapter 5.



(a) NAM (b) VAT4Net

Figure 7.11: Visualizing Tools for ns-2 Network Simulations

After looking at the visualization and its network dynamics it would be interesting to extract various statistical data out of the example simulation. In the example gnuplot is chosen. gnuplot is a command-line program that can generate two- and three-dimensional plots of functions and data. In Listing A.17 the end-to-end Delay calculated above in the ”Analyzing Methods” paragraph is plotted with gnuplot [38] (see Figure 7.12(a)). In Figure 7.12(b) different network simulation with different parameters are analyzed and compared to each other.



(a) End-to-End Statistic Example

(b) End-to-End Statistics on various Simulations

Figure 7.12: Statistical Plot generated with gnuplot [38].

7.4.4 Content of the Additional Sections in the Theory Chapter

Readings

As a first additional theory chapter the ”Readings” chapter takes place in the theory part of the module. **Required readings** are articles which essentially should be read by the participant of the course. Each of this reading will be checked in the quiz section or will be used in the practical part as necessary knowledge. **Recommended readings** are interesting and fit the theory but are not a must. Recommended readings should be a good enhancement for further information and knowledge.

As required readings for the module the following two papers are chosen:

- ”Advances in Network Simulation”, [39] - This is the standard paper of the ns-2 development project VINT. It includes basic information about the ns-2 project and its importance.

- ”Hybrid Packet/Fluid Flow Network Simulation”,[24] - The paper discusses a possible approach to implement hybrid network simulations. The paper should enhance the theory section regarding this theme.

In the recommended readings section more or less all quotation of references from the theory part are placed to give the learner a start point to read through further resources and deepen his knowledge.

Personal Synthesis, Self Test and Quiz

The structure and content of these additional sections is specified by the Didactics and Design Guide [11].

The first of two **personal syntheses** should advice the student to think about the learned material and to organize the new acquired knowledge. These synthesis is a didactic way to consolidate the knowledge by expressing it in own words. The personal syntheses will be rated by the tutor and has influence of passing successfully the module.

Furthermore, the **self test** allows the students orientating themselves in the theory part and discovering gap in their knowledge. The Self Test is always available and for wrong answers it leads to the appropriate theory part.

The **quiz** at the end of the theory section is also included into grading of the course and acts as the final test before proceeding to the lab session. Without passing successfully this quiz, it makes no sense to go further to the laboratory sessions because it could be assumed that there are parts of the theory that are not worked through properly.

7.5 Module Chapter - Knowledge Application and Exploration

7.5.1 Structure

The practical work in the module is divided into four different hands-on session (see Listing 7.34). Three out of four hands-on sessions are constitutive exercises to get familiarized with the network simulator and the implementation methods presented in the theoretical part. The fourth exercise represents the major task in the practical part and includes the implementation of a routing protocol.

```
3 Knowledge Application/Exploration
   3.1 Introduction
   3.2 Hands-on Session
      3.2.1 Hands-on Session 1 - Tcl Exercise
      3.2.2 Hands-on Session 2 - Wireless Tcl Exercise
      3.2.3 Hands-on Session 3 - RFC 865
      3.2.4 Hands-on Session 4 - Implementing AODV
```

Listing 7.34: Module Chapter Overview

7.5.2 Content

Tcl Exercise - a standard setup of a wired network simulation

In the first hands-on session the student has to complete the exercise as follows. The evaluation criteria will be the NAM trace file generated by ns-2.

a. FTP over TCP

As a first step the student has to create a basic topology as it is depicted in Figure 7.13. The user

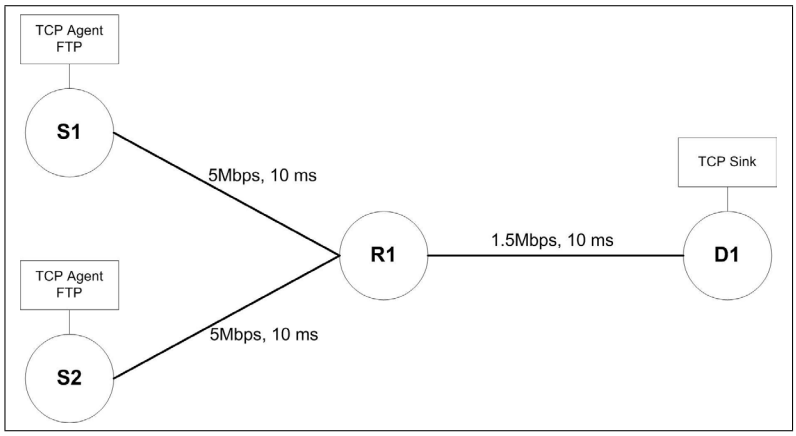


Figure 7.13: Network topology for hands-on Session 1 with 4 nodes.

should generate a simualtion as follows:

- Set up a TCP connection between S1 and the sink agent at D1. Attach a FTP application at S1.
- Set up a TCP connection between S2 and the sink agent at D1. Attach a second FTP application at S2.
- Set up D1 as a TCP sink agent.
- Set up the links between S1 and R1 and S2 and R1 with 5 Mbps and a delay of 10 ms.
- Set up the link between R1 and R2 with 1.5Mbps and a delay of 10 ms.
- The simulation run time takes 25 seconds.
- Start the FTP application at 0.5 seconds.
- Use a drop tail queue for all nodes and monitor the queues.
- Use NAM as trace file format

b. CBR

Applying some changes to the first topology, the second simulation should be run as following:

- Remove the TCP connection between S2 and D1. Instead of TCP apply UDP.
- Add a null agent to D1.
- Attach a CBR application to the TCP agent on node S2. Set the packet size to 1024 and the interval time to 0.0005.
- Start the CBR application at 0.5 sec and end it at 4.5 sec.

To grade this hands-on exercise the student has to deliver parts of the trace file and a comparison between the two different simulations a. and b. The student has to analyze the packet loss rate in detail.

Wireless Tcl Exercise - a more advanced Tcl exercise with wireless nodes and movement setup

In this hands-on session the student has to configure a wireless simulation. The basic wireless node setup and movement model are predefined. The student has to implement the following parts:

- Setup 5 wireless nodes with the predefined options.
- Node 0: UDP agent, CBR traffic, connected with NULL agent on node 4
- Node 2: UDP agent, CBR traffic, connected with NULL agent on node 3
- Node 2: UDP agent, CBR traffic, connected with NULL agent on node 4
- Node 1: TCP agent, FTP traffic, connected with TCP sink agent on node 3
- Node 1: TCP agent, FTP traffic, connected with TCP sink agent on node
- Node 2: TCP agent, FTP traffic, connected with TCP sink agent on node 0
- Analyze the trace file and visualize it with vat4net

The grading criteria will be a detailed calculation on the packet loss rate and the end-to-end delay of the simulations. Intermediate results should be submitted. It should be clear how the results are gained from the simulation.

RFC 865 - Implementing Quote of the Day

The main idea of this hands-on session is to extend an existing ns-2 protocol implementation. In this hands-on session the student has to implement RFC 865 [40]. Implementing "Quote of the Day Protocol" as an agent "agent/QOD" in ns-2 is a basic exercise to show how to implement and extend new protocols into ns-2.

Most parts of the source code are predefined and the student has only to fill in some gaps. A new packet type QOD is predefined in the file <nsrootdirectory>/QOD/qod_pkt.h (see Listing A.18). The C++ header file of the main agent implementation is also available (see Listing A.18) and can be used for the implementation. With help of the two files, the main C++ class qod.cc with the two following methods have to be implemented:

- **command:** In the command method the Tcl command "send" has to be implemented, which will be invoked in the qod.tcl file (Listing A.20, line 28).
- **recv:** The receive method must implement two main functionalities. The first functionality is to react, as a server, to a "quote of the day" request, and thus answer to the requestor by sending back a quote. The second functionality musts process the received/requested quote. In this simple example only print out the quote to standard output.

After implementing these methods, the new agent has to be included and compiled into ns-2 and the necessary changes on ns-2 has to be done. To test the implemented code a predefined Tcl file is available as already mentioned above (Listing A.20). As a result of this exercise, the output of the simulation and the qod.cc file should be provided.

Implementing AODV - the main exercise of this module

In the last exercise the student has to set up the Ad hoc On Demand Distance Vector (AODV) routing protocol for ad hoc mobil networks. AODV is a on demand algorithm like LAR, meaning that it builds routes as desired by a source node without a route to the destination. The version of AODV to implement is simplified as described in the following section. The full version of AODV is described in the RFC3561 [41].

The student has to implement a smaller version of AODV, called Small AODV in the following. He has to implement only the RREQ(route request) and the RREP(route reply) mechanism of AODV, without any RERR (error) packets and other more complex procedures.

1. RREQ and RREP
When a node wishes to send data to a unknown node without a routing table entry in the network, it broadcasts a RREQ message. If an other node than the source node receives the RREQ message, it has two choices: if it knows a route to the destination or if it is the destination, it can send a RREP back to the source node. Otherwise, it will rebroadcast the RREQ to all its neighbors. All of the nodes use a sequence number in the RREQ to ensure that they do not rebroadcast a RREQ.

2. Sequence numbers
Sequence numbers serve as time stamps. They allow nodes to compare how fresh their information on other nodes is. Every time a node sends out any type of message it increases its own sequence number. Each node records the sequence numbers of all the other nodes it talks to. A higher sequence number signifies a fresher route.

The student has not to implement the whole protocol but to use, the templates prepared in the ns-2 directory on the laboratory computer in directory <nsrootdirectory>/SMALLAODV. This hands-on session is similar to the third session, but is much more complex. There are many methods more to implement.

For grading this hands-on session the student has to do some tests on the implemented protocol and the basic AODV implementation of ns-2. All the analyzing and visualizing methods are available for the evaluation of the protocol.

7.6 Module Chapter - Prove Your Knowledge and Skills

At the end of the practical part the student has to demonstrate the gained knowledge in the following parts used to grade the students work. See Listing 7.1) for details on the structure of this chapter.

Personal Synthesis, Final Quiz and Survey

In the **second essay** the student has to prove the acquired knowledge. ”The synthesis should contain reflections on adopted strategies and the observed results”, [11] of the practical part.

The **final quiz** serves to grade the results of the hands-on sessions. The student must prove that he has learned, done the practical part with necessary accuracy and understood the learning material.

The **survey** should help the developers to adapt and improve the content of the module. The student is awarded with some additional point for grading.

Chapter 8

Conclusions and Outlook

This Diploma thesis consists of two main parts:

- The development of the module ”Implementing Protocols on Network Simulators” in the context of the EuQoS e-learning infrastructure.
- The implementation of a supporting tool for animation and visualization of network simulations VAT4Net.

8.1 EuQoS E-learning Module

Use and Problems

The module ”Implementing Protocols on Network Simulators” is already in use now. As one of the modules of the EuQoS course system, it was used and reviewed in the Master's course ”Multimedia Communication” at University of Bern. Due to the feedback provided by the students some improvements and corrections have been applied. A main criticism was the degree of difficulty of the provided examples concerning protocol implementations in ns-2. This tasks requires a good knowledge in C++. A good point would be to include more basic C++ explanations into the e-learning chapters as it has been done already in some parts. The other parts of the theory are easy to understand and work through. The hands-on session exercises are practicable whenever the student has comprehended the theory part, except of the fact, that the missing knowledge in C++ could be a problem.

Work Process

Network Simulation is a well known subject in network research, anyhow there are not piles of standard books available. There are lots of specialized academic papers which concentrate on some subjects but never try to give a general view on the whole topic. The module gives a summary on the topic including some standard simulation theory and ends with a tutorial part which helps to understand one of the most common simulators ns-2. While ns-2 is rather complex to understand and marginally documented, the module development was a self-learning process on the simulator and its theory.

8.2 VAT4Net

Work Process

VAT4Net has been implemented from scratch. There were various different aspects from different branches in computer science incorporating into the implementation process. The implementation of SSH connection and client-server architecture into the application demanded knowledge in network and distributed systems. Finding an appropriate node placing algorithm was meanwhile a graph drawing problem. A problem from the branch of computational geometry and graphics was the animation (of network simulations).

A main problem was the memory consumption and the time complexity of the application. In a first prototype there has been many limitations in the selection of the trace files processed with VAT4Net, e.g. the size of the trace file or the number of animated entities and events. In a new development phase the process of loading data was changed into a streaming process behavior, which requires buffering and releasing data. With this enhancements, larger trace files can be used with VAT4Net.

Use and Problems

While working on the hands-on sessions the student has to use VAT4Net as a supporting tool to gain results an have a closer look at his own simulations. In this case, VAT4Net is available as client server application while the server is running on the laboratory computers. A main problem is the connection speed between client and server. There is need of higher bandwidth (minimum 512 Mbit/s). Further, a trace file should not be to long (maximum 100 MB) because of the duration of the preprocessing on the server side causing a delay of transmission.

Further Enhancements

Nevertheless, there are points open for future work. Some of them are listed below:

- Opening a trace file on the server should be implemented with a file chooser and a selectable directory.
- For large trace file support there should be a selectable begin and end time for preprocessing, animation and statistical data processing.
- There are some elements and events not yet implemented but available from a trace file, e.g. traffic agents. It would be a nice-to-have feature to animate them.
- In the current implementation a plugin is loaded when the classes have the same origin as the core. Future implementations may try to load plugins from ”foreign” locations.
- A further improvement could be the implementation of a selectable trace file preprocessor that supports other trace file formats either from ns-2 or other network simulators.
- Some additional plugins could be implemented.

Bibliography

[1] M. Ally, *Foundations of educational theory for online learning.* Athabasca, AB: Athabasca University, 2004, vol. In T. Anderson and F. Elloumi (Eds.), Theory and practice of online learning (pp. 3-31).

[2] H. Troitzsch, C. Sengstag, D. Miller, and C. Clases, "Roadmap to e-learning @ ETH Zurich," *NET - Network for Educational Technology*, 2006.

[3] I. Hamburg, C. Lindecke, and H. ten Thij, "Social aspects of e-learning and blending learning methods," *In: E-Comm-Line 2003: 4th European Conference on E-Commerce, E-Work, E-Learning, E-Health, E-Banking, E-Business, On-line Services, Virtual Institutes, and their Influences on the Economic and Social Environment, september 25-26, 2003, Bucharest, Romania. Bucharest: IPA SA, R and D Inst. for Automation*, pp. 11–15, 2003.

[4] Economist Intelligence Unit and IBM, *The 2003 e-Learning Readiness Rankings*. Economist Intelligence Unit,London, 2003.

[5] "EuQoS, end-to-end quality of service support over heterogeneous networks project," http://www.euqos.eu/.

[6] M.-A. Steinemann, T. Bernoulli, T. Staub, *et al.*, "Specification of contents and didactical concept of the course to be offered," *EuQoS Deliverable D6.1.1, CEC Deliverable Number 004503/UoB/DS/D6.1.1./A1, May 31, 2005*, 2005.

[7] A. Weyland and T. Braun, *OSLab Course Author Guide*, 2006.

[8] T. Braun *et al.*, "Online practical telecommunications training - the VITELS project," *33th International Symposium IGIP,IEEE,ASEE 2004 (IGIP) , Fribourg, Switzerland*, 2004.

[9] "Switch, swiss academic and research network," http://www.switch.ch/.

[10] "Shibboleth - an Internet2 middleware project," http://shibboleth.internet2.edu/.

[11] M.-A. Steinemann, A. Weyland, J. Viens, T. Braun, S. Abdellatif, C. Chassot, and P. Pinheiro, *EuQoS - Didactics and Design Guide, Version 0.91*, 2005.

[12] P. Vapi *et al.*, *file framework generator and formatter, FFGF Perl Version*, available online: https://subversion.cnds.unibe.ch/svn/e-learning/ffgf/.

[13] D. Estrin, M. Handley, J. Heidemann, S. McCanne, Y. Xu, and H. Yu, "Network visualization with the VINT network animator nam," University of Southern California, Tech. Rep. 99-703, 1999. [Online]. Available: citeseer.ist.psu.edu/estrin99network.html

[14] "JAVA - Sun Developer Network (SND)," http://java.sun.com/.

[15] B. Scheuermann, H. Fuessler, M. Transier, M. Busse, M. Mauve, and W. Effelsberg, "Hug-inn: a 3d visualizer for wireless ns-2 traces," in *MSWiM '05: Proceedings of the 8th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*. New York, NY, USA: ACM Press, 2005, pp. 143–150.

[16] "Ganymed SSH-2 for Java," http://www.ganymed.ethz.ch/ssh2/.

[17] T. M. J. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software - Practice and Experience*, vol. 21, no. 11, pp. 1129–1164, 1991. [Online]. Available: citeseer.ist.psu.edu/fruchterman91graph.html

[18] "JFreeChart - a free java chart library," http://www.jfree.org/.

[19] "Mindterm Java Applet," http://www.appgate.com/mindterm.

[20] J. Krähenbühl, T. Staub, and T. Bernoulli, *Implementing Protocols on Network Simulators*. e-learning module of EuQoS e-learning portal, 2006.

[21] A. Sulistio, C. S. Yeo, and R. Buyya, "A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools," *Softw. Pract. Exper.*, vol. 34, no. 7, pp. 653–673, 2004.

[22] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, pp. 1087–1092, 1953.

[23] T. Yung, J. Martin, M. Takai, and R. Bagrodia, "Integration of fluidbased analytical model with packet-level simulation for analysis of computer networks," 2001.

[24] C. Kiddle, R. Simmonds, C. Williamson, and B. Unger, "Hybrid packet/fluid flow network simulation," in *PADS '03: Proceedings of the seventeenth workshop on Parallel and distributed simulation*. Washington, DC, USA: IEEE Computer Society, 2003, p. 143.

[25] J. Incera, R. Marie, D. Ros, and G. Rubino, "Fluidsim: a tool to simulate fluid models of high-speed networks," *Perform. Eval.*, vol. 44, no. 1-4, pp. 25–49, 2001.

[26] B. Liu, D. R. Figueiredo, Y. Guo, J. F. Kurose, and D. F. Towsley, "A study of networks simulation efficiency: Fluid simulation vs. packet-level simulation," in *INFOCOM*, 2001, pp. 1244–1253.

[27] K. Fall, "Network emulation in the VINT/NS simulator," *Proceedings of the fourth IEEE Symposium on Computers and Communications*, 1999. [Online]. Available: citeseer.ist.psu.edu/93741.html

[28] X. Zeng, R. Bagrodia, and M. Gerla, "GloMoSim: A library for parallel simulation of large-scale wireless networks," in *Workshop on Parallel and Distributed Simulation*, 1998, pp. 154–161. [Online]. Available: citeseer.ist.psu.edu/zeng98glomosim.html

[29] "QualNet," http://www.qualnet.com/.

[30] R. Barr, Z. J. Haas, and R. van Renesse, *Scalable Wireless Ad hoc Network Simulation. Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad hoc Wireless, and Peer-to-Peer Networks.* CRC Press, 2005.

[31] J. H. Cowie, D. M. Nicol, and A. T. Ogielski, "Modeling the global internet," *Computing in Science and Engg.*, vol. 1, no. 1, pp. 42–50, 1999.

[32] A. Varga, "The OMNeT++ discrete event simulation system," in *European Simulation Multiconference (ESM'2001)*, June 2001.

[33] "ns-2," http://www.isi.edu/nsnam/ns/index.html.

[34] "VINT - virtual internetwork testbed project," http://www.isi.edu/nsnam/vint/index.html.

[35] K. Fall and K. Varadhan, "The ns manual (formerly ns notes and documentation," VINT - Virtual InterNetwork Testbed Project, 2002.

[36] F. J. Ros and P. M. Ruiz, "Implementing a new manet unicast routing protocol in ns-2," Dept. of Information and Communications Engineering University of Murcia, Tech. Rep., 2004.

[37] Y.-B. Ko and N. H. Vaidya, "Location-aided routing (LAR) in mobile ad hoc networks," in *Mobile Computing and Networking*, 1998, pp. 66–75. [Online]. Available: citeseer.ist.psu.edu/article/ko98locationaided.html

[38] "gnuplot," http://www.gnuplot.info/.

[39] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu, "Advances in network simulation," *Computer*, vol. 33, no. 5, pp. 59–67, 2000.

[40] "RFC 865 - quote of the day protocol," http://www.ietf.org/rfc/rfc0865.txt.

[41] "RFC 3561 - ad hoc on-demand distance vector (aodv) routing," http://www.ietf.org/rfc/rfc3561.txt.

[42] "GloMoSim," http://pcl.cs.ucla.edu/projects/glomosim/.

[43] "SWANS," http://jist.ece.cornell.edu/.

[44] "SSFNet," http://www.ssfnet.org/homePage.html.

[45] "OMNet++," http://www.omnetpp.org/.

Appendix A

Code Listings

This chapter contains all code listings, which are referenced in the Diploma thesis.

Listings

```
set ns [new Simulator]

# Open Trace File
set tracefile [open out.tr w]
$ns trace-all $tracefile

#Open the NAM Trace File
set namfile [open out.nam w]
$ns namtrace-all $namfile

$ns at 125.0 "finish"

proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam out.nam &
    exit 0
}

$ns run
```

Listing A.1: ns-2 Preliminaries

```
set ns [new Simulator]

# Open Trace File
set tracefile [open out.tr w]
$ns trace-all $tracefile

#Open the NAM Trace File
set namfile [open out.nam w]
$ns namtrace-all $namfile

$ns at 20.0 "finish"

proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam out.nam &
    exit 0
}

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
```

```
25 set n2 [$ns node]
26 set n3 [$ns node]
27
28 #Create links between the nodes
29 $ns duplex-link $n0 $n2 1Mb 10ms DropTail
30 $ns duplex-link $n1 $n2 1Mb 10ms DropTail
31 $ns duplex-link $n3 $n2 1Mb 10ms DropTail
32
33 #Create a TCP connection on n0
34 set tcp0 [new Agent/TCP]
35 $ns attach-agent $n0 $tcp0
36 $tcp0 set fid_ 1
37 $tcp0 set packetSize_ 552
38
39 #Setup a FTP over TCP connection
40 set ftp0 [new Application/FTP]
41 $ftp0 attach-agent $tcp0
42
43 #Setup
44 set sink0 [new Agent/TCPSink]
45 $ns attach-agent $n3 $sink0
46
47 $ns connect $tcp0 $sink0
48
49 # Setup a UDP connection
50 set udp1 [new Agent/UDP]
51 $ns attach-agent $n1 $udp1
52 $udp1 set fid_ 2
53
54 # Setup a CBR over UDP connection
55 set cbr1 [new Application/Traffic/CBR]
56 $cbr1 attach-agent $udp1
57 $cbr1 set packetSize_ 1000
58 $cbr1 set interval_ 0.005
59 $cbr1 set random_ false
60
61 # Setup a Null Agent
62 set null1 [new Agent/Null]
63 $ns attach-agent $n3 $null1
64 $ns connect $udp1 $null1
65
66 $ns at 1.0 "$ftp0 start"
67 $ns at 4.0 "$ftp0 stop"
68 $ns at 0.5 "$cbr1 start"
69 $ns at 4.5 "$cbr1 stop"
70 $ns run
```

Listing A.2: ns-2 Nodes Links and Traffic

```
set opt(chan)       Channel/WirelessChannel
set opt(prop)       Propagation/TwoRayGround
set opt(netif)      Phy/WirelessPhy
set opt(mac)        Mac/802_11
set opt(ifq)        Queue/DropTail/PriQueue
set opt(ll)     LL
set opt(ant)        Antenna/OmniAntenna
set opt(x)      670    ;# X dimension of the topography
set opt(y)      670    ;# Y dimension of the topography
set opt(ifqlen)         50          ;# max packet in ifq
set opt(seed)       0.0
set opt(tr)     694demo.tr      ;# trace file
set opt(nam)                694demo.nam    ;# nam trace file
set opt(adhocRouting)   DSDV
set opt(nn)                 3              ;# how many nodes are simulated
set opt(cp)     "traffic.file"
set opt(sc)     "movement.file"
set opt(stop)       200.0     ;# simulation time

# create simulator instance
set ns_ [new Simulator]

# topology
set topo [new Topography]
$topo load_flatgrid 670 670

# ns trace
set tracefd [open wireless_scenario.tr w]
$ns_ trace-all $tracefd


# nam trace
set namtrace [open wireless_scenario.nam w]
$ns_ namtrace-all-wireless $namtrace 670 670

# create god
set god_ [create-god 3]

# Mobile Node configuration
$ns_ node-config -adhocRouting $opt(adhocRouting) \
       -llType $opt(ll) \
       -macType $opt(mac) \
       -ifqType $opt(ifq) \
       -ifqLen $opt(ifqlen) \
       -antType $opt(ant) \
       -propType Propagation/TwoRayGround \
       -phyType $opt(netif) \
       -channelType Channel/WirelessChannel \
```

```
49          -topoInstance $topo \
50          -agentTrace ON \
51          -routerTrace OFF \
52          -macTrace ON
53
54  for {set i 0} {$i < 3} {incr i} {
55    set node_($i) [$ns_ node]
56    #disable random motion
57    $node_($i) random-motion 0
58    $node_($i) topography $topo
59  }
60
61  #Load movement file
62  source movement.file
63
64  #Load traffic file
65  source traffic.file
66
67  # Define node initial position in nam
68  for {set i 0} {$i < 3 } { incr i} {
69     $ns_ initial_node_pos $node_($i) 20
70  }
71
72 for {set i 0} {$i < 3 } {incr i} {
73     $ns_ at 150.0 "$node_($i) reset";
74 }
75
76  # Tell ns/nam the simulation stop time
77  $ns_ at 150.0 "$ns_ nam-end-wireless 150.0"
78  $ns_ at 150.0 "$ns_ halt"
79
80  # Start your simulation
81  $ns_ run
```

Listing A.3: ns-2 Wireless Scenario

```
$ns_ at 50.000000000000 "$node_(2) setdest 369.463244915743
    170.519203111152 3.371785899154"
$ns_ at 51.000000000000 "$node_(1) setdest 221.826585497093
    80.855495003839 14.909259208114"
$ns_ at 33.000000000000 "$node_(0) setdest 89.663708107313
    283.494644426442 19.153832288917"
$god_ set-dist 1 2 2
$god_ set-dist 0 2 3
$god_ set-dist 0 1 1
$node_(2) set Z_ 0.000000000000
$node_(2) set Y_ 199.373306816804
$node_(2) set X_ 591.256560093833
$node_(1) set Z_ 0.000000000000
$node_(1) set Y_ 345.357731779204
$node_(1) set X_ 257.046298323157
$node_(0) set Z_ 0.000000000000
$node_(0) set Y_ 239.438009831261
$node_(0) set X_ 83.364418416244
```

Listing A.4: ns-2 Wireless Scenario Movement File

```
#
# 0 connecting to 2 at time 127.93667922166023
#
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(0) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(2) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 4.0
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 36.93667922166023 "$cbr_(0) start"

set tcp [new Agent/TCP]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns_ attach-agent $node_(1) $tcp
$ns_ attach-agent $node_(2) $sink
$ns_ connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns_ at 3.00000000000000 "$ftp start"
```

Listing A.5: ns-2 Wireless Scenario Traffic File

```
#ifndef __lar_pkt_h__
#define __lar_pkt_h__

#include <packet.h>


/*Header Macros*/
#define HDR_LAR_PKT(p) hdr_lar_pkt::access(p)
#define maxRouteLength        32

struct hdr_lar_pkt {

    nsaddr_t    pkt_src_;      // Node which originated this packet
    u_int16_t   pkt_len_;      // Packet length (in bytes)
    u_int8_t    pkt_seq_num_; // Packet sequence number
    char       forwardCode_; // F=flood,S=Scheme 2, U=unset,
    char       larCode_;      // D=data,R=routing, A=routereply
    double       sendTime_;
    bool       newPkt_;        //true if this is packet done only one hop
    double        sourceX_;
    double        sourceY_;
    int        hops_;
    double        lastHopX_;
    double        lastHopY_;
    nsaddr_t    destinationID_;
    nsaddr_t    route_[maxRouteLength];

    double        destinationX_;
    double        destinationY_;
    double        requestSendTime_;
    nsaddr_t    requestID_;

    nsaddr_t    replyID_;
    double        replySendTime_;

    inline nsaddr_t&   pkt_src()     { return pkt_src_; }
    inline u_int16_t&  pkt_len()     { return pkt_len_; }
    inline u_int8_t&   pkt_seq_num(){ return pkt_seq_num_; }
    inline char&       forwardCode(){ return forwardCode_; }
    inline char&       larCode()   { return larCode_; }
    inline double&     sendTime()    { return sendTime_; }
    inline bool&       newPkt()      { return newPkt_; }
    inline double&     sourceX()     { return sourceX_; }
    inline double&     sourceY()     { return sourceY_; }
    inline int&        hops()      { return hops_; }
    inline double&     lastHopX()    { return lastHopX_; }
    inline double&     lastHopY()    { return lastHopY_; }
    inline nsaddr_t&   destinationID() { return destinationID_; }
```

```
49
50      inline double&  destinationX()    { return destinationX_; }
51      inline double&  destinationY()    { return destinationY_; }
52      inline double& requestSendTime(){return requestSendTime_;}
53      inline nsaddr_t& requestID()      { return requestID_; }
54
55      inline nsaddr_t &replyID() { return replyID_; }
56      inline double   &replySendTime() { return replySendTime_; }
57
58      static int offset_;
59      inline static int& offset() { return offset_; }
60      inline static hdr_lar_pkt* access(const Packet* p) {
61         return (hdr_lar_pkt*)p->access(offset_);
62      }
63  };
64  #endif
```

Listing A.6: LAR Packet Type

```
#ifndef __lar_h__
#define __lar_h__

#include "lar_pkt.h"
#include "trace.h"
#include "object.h"
#include "agent.h"
#include "packet.h"
#include "ip.h"
#include "mobilenode.h"
#include "timer-handler.h"
#include "random.h"
#include "classifier-port.h"

#include <set>
#include <map>
#include <list>
#include <iostream>

using namespace std;

#define CURRENT_TIME      Scheduler::instance().clock()
#define maxRouteLength    32

class LAR;          / forward declaration

typedef struct
{
   double x;
   double y;
   char   method;  //flood or scheme 2
   nsaddr_t  route[maxRouteLength];
   int hops;
   double timeOfLastRequest;
   bool requestPending;
   bool noRoute;
} larDestType;


class LAR_PktTimer : public TimerHandler {
   public:
      LAR_PktTimer(LAR* agent) : TimerHandler() {
         agent_ = agent;
      }
   protected:
      LAR * agent_;
       virtual void expire(Event* e);
};
```

```
49
50
51  class LAR : public Agent {
52
53      friend class LAR_PktTimer;
54
55      private:
56          map<nsaddr_t, larDestType, less<int> > routeTable;
57          MobileNode *node;
58
59      protected:
60          PortClassifier*          dmux_;
61          Trace*                   logtarget_;
62          LAR_PktTimer             pkt_timer_;
63
64          inline nsaddr_t&     ra_addr()         { return ra_addr_; }
65
66          void forward_data(Packet*, Handler*, larDestType*);
67          void recv_lar_pkt(Packet*, Handler*);
68
69          void reset_lar_pkt_timer();
70
71      public:
72          u_int8_t            seq_num_;
73          nsaddr_t            ra_addr_;
74          LAR(nsaddr_t);
75          void   recv(Packet*, Handler*);
76          int    command(int, const char*const*);
77  };
78  #endif
```

Listing A.7: LAR Header File lar.h

```
#include "lar.h"

using namespace std;

// if a route reply is not recieved in this amount of time the
// request is flooded
double LARRouteRequestTimeout = 0.5;

//lar factor for scheme 2
double LARDelta = 0.0;

int hdr_lar_pkt::offset_;
static class LARHeaderClass: public PacketHeaderClass {
   public:
      LARHeaderClass():  PacketHeaderClass("PacketHeader/LAR", sizeof
          (hdr_lar_pkt)) {
          bind_offset(&hdr_lar_pkt::offset_);
      }
} class_rtProtoLAR_hdr;

static class LARClass: public TclClass {
   public:
      LARClass() : TclClass("Agent/LAR") {}
      TclObject* create(int argc, const char*const* argv) {
          assert(argc==5);
          return (new LAR((nsaddr_t)Address::instance().str2addr(argv
              [4])));
      }
} class_rtProtoLAR;

void LAR_PktTimer::expire(Event* e) {
   agent_->reset_lar_pkt_timer();
}

void LAR::reset_lar_pkt_timer() {
   pkt_timer_.resched((double)5.0);
}

LAR::LAR(nsaddr_t id) : Agent(PT_LAR), dmux_(0), pkt_timer_(this) {
   ra_addr_ = id;
   node = NULL;
   //bind_bool("accessible_var_", &accessible_var_);
}

int LAR::command(int argc, const char*const* argv) {
   if (argc == 2) {
      if(strcasecmp(argv[1], "start") == 0){
          pkt_timer_.resched(0.0);
```

```
47          return TCL_OK;
48      }
49  } else if (argc == 3) {
50      if(strcmp(argv[1], "port-dmux") == 0 ) {
51          dmux_ = (PortClassifier *)TclObject::lookup(argv[2]);
52          if (dmux_ == 0) {
53              fprintf (stderr, "%s: %s lookup of %s failed\n", __FILE__
                    ,
54              argv[1], argv[2]);
55              return TCL_ERROR;
56          }
57          return TCL_OK;
58      } else if(strcmp(argv[1], "tracetarget") == 0 ) {
59          logtarget_ = (Trace*)TclObject::lookup(argv[2]);
60          if(logtarget_ == 0) {
61              return TCL_ERROR;
62          }
63          return TCL_OK;
64      } else if (strcasecmp (argv[1], "node") == 0) {
65          node = (MobileNode*)TclObject::lookup(argv[2]);
66          return TCL_OK;
67      }
68  }
69  return (Agent::command(argc, argv));
70  }
71
72  void LAR::recv(Packet* pkt, Handler* h) {
73      struct hdr_cmn* ch = hdr_cmn::access(pkt);
74      struct hdr_ip* ih  = hdr_ip::access(pkt);
75
76      if(ch->ptype() == PT_LAR) {
77          ih->ttl_--;
78          recv_lar_pkt(pkt,h);
79          return;
80      }
81
82      if(ih->saddr() == ra_addr()) {
83          //there exists a loop -> drop the packet
84          if(ch->num_forwards() > 0) {
85              drop(pkt, DROP_RTR_ROUTE_LOOP);
86              return;
87          } else if(ch->num_forwards() == 0) {
88              ch->size() += IP_HDR_LEN;
89          }
90      } else if (--(ih->ttl_) == 0) {
91          drop(pkt, DROP_RTR_TTL);
92          return;
93      }
94
```

```
95      if ((u_int32_t)ih->daddr() != IP_BROADCAST) {
96          route_resolve(pkt, h);
97      }else{
98          forward_data(pkt, h, (larDestType*) 0);
99      }
100 }
101
102 void LAR::recv_lar_pkt(Packet* pkt, Handler* h) {
103     struct hdr_cmn* ch = hdr_cmn::access(pkt);
104     struct hdr_ip* ih  = hdr_ip::access(pkt);
105     struct hdr_lar_pkt* larhdr = hdr_lar_pkt::access(pkt);
106
107     assert(ih->sport() == RT_PORT);
108     assert(ih->dport() == RT_PORT);
109
110     node->update_position();
111     larhdr->newPkt() = false;
112
113     if(this->duplicateRcvdPacket(ch->uid_) || this->
            duplicateSentPacket(ch->uid_)) {
114         Packet::free(pkt);
115         return;
116     }
117
118     /* processing of LAR routing packets */
119
120     Packet::free(pkt);
121     return;
122 }
123
124 void LAR::forward_data(Packet* p,Handler* h, larDestType* rt) {
125     struct hdr_cmn* cmnhdr = hdr_cmn::access(p);
126     struct hdr_ip*  iphdr =  hdr_ip::access(p);
127
128     if (iphdr->ttl_ == 0) {
129         drop(p, DROP_RTR_TTL);
130         return;
131     }
132
133     if(cmnhdr->ptype() != PT_LAR && cmnhdr->direction() == hdr_cmn::UP
            && ((u_int32_t)iphdr->daddr() == IP_BROADCAST || iphdr->daddr
            () == ra_addr())) {
134         dmux_->recv(p,h);
135         return;
136     }
137
138     if (rt) {
139         cmnhdr->direction() = hdr_cmn::DOWN;
140         cmnhdr->addr_type() = NS_AF_INET;
```

```
141         cmnhdr->next_hop_ = rt->route[1];
142     }else{
143         // Broadcst message
144         cmnhdr->direction() = hdr_cmn::DOWN;
145         cmnhdr->addr_type() = NS_AF_NONE;
146     }
147 
148     Scheduler::instance().schedule(target_, p, 0.0);
149 }
```

Listing A.8: LAR Source File lar.cc

```
enum packet_t {
    PT_TCP,
    PT_UDP,
    PT_CBR,
(...)
    PT_LAR,
    PT_NTYPE // This MUST be the LAST one
};

(...)

p_info() {
    name_[PT_TCP]= "tcp";
    name_[PT_UDP]= "udp";
    name_[PT_CBR]= "cbr";
(...)
    name_[PT_LAR]="LAR";
    name_[PT_NTYPE]= "undefined";
}
```

Listing A.9: Required changes in ns-2 packet.h

```
foreach prot {
    AODV
    ARP
(...)
    LAR
    NV
}
{
    dd-packet-header $prot
}
```

Listing A.10: Required changes in ns-2 ns-packet.tcl

```
OBJ_CC = \
    tools/random.o tools/rng.o tools/ranvar.o
    common/misc.o common/timer-handler.o \
(...)
    lar/lar.o lar/lar_pkt.o \
(...)
    $(OBJ_STL)
```

Listing A.11: Required changes in ns-2 Makefile

```
switch -exact $routingAgent_ {
    DSDV {
        set ragent [$self create-dsdv-agent $node]
    }
    DSR {
        $self at 0.0 "$node start-dsr"
    }
    AODV {
        set ragent [$self create-aodv-agent $node]
    }
(...)
    LAR {
        set ragent [$self create-lar-agent $node]
    }
    DumbAgent {
        set ragent [$self create-dumb-agent $node]
    }
    default {
        puts "Wrong node routing agent!"
        exit
    }
}

(...)

Simulator instproc create-lar-agent { node } {
    # Create Lar routing agent
    set ragent [new Agent/LAR [$node node-addr]]
    $self at 0.0 "$ragent node $node"
    $node set ragent_ $ragent
    return $ragent
}
```

Listing A.12: Required changes in ns-2 ns-lib.tcl

```
+ 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610
- 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610
r 1.84471 2 1 cbr 210 ------- 1 3.0 1.0 195 600
r 1.84566 2 0 ack 40 ------- 2 3.2 0.1 82 602
+ 1.84566 0 2 tcp 1000 ------- 2 0.1 3.2 102 611
- 1.84566 0 2 tcp 1000 ------- 2 0.1 3.2 102 611
r 1.84609 0 2 cbr 210 ------- 0 0.0 3.1 225 610
+ 1.84609 2 3 cbr 210 ------- 0 0.0 3.1 225 610
d 1.84609 2 3 cbr 210 ------- 0 0.0 3.1 225 610
- 1.8461 2 3 cbr 210 ------- 0 0.0 3.1 192 511
r 1.84612 3 2 cbr 210 ------- 1 3.0 1.0 196 603
+ 1.84612 2 1 cbr 210 ------- 1 3.0 1.0 196 603
- 1.84612 2 1 cbr 210 ------- 1 3.0 1.0 196 603
+ 1.84625 3 2 cbr 210 ------- 1 3.0 1.0 199 612
```

Listing A.13: Extract of a Trace File in Normal Trace Format

```
s -t 0.267662078 -Hs 0 -Hd -1 -Ni 0 -Nx 5.00
  -Ny 2.00 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0
  -Ms 0 -Mt 0 -Is 0.255 -Id -1.255 -It message -Il 32 -If 0
  -Ii 0 -Iv 32


s -t 1.511681090 -Hs 1 -Hd -1 -Ni 1 -Nx 390.00
  -Ny 385.00 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0
  -Ms 0 -Mt 0 -Is 1.255 -Id -1.255 -It message -Il 32 -If 0
  -Ii 1 -Iv 32

r -t 10.000000000 -Hs 0 -Hd -2 -Ni 0 -Nx 5.00
  -Ny 2.00 -Nz 0.00 -Ne -1.000000 -Nl RTR -Nw --- -Ma 0 -Md 0
  -Ms 0 -Mt 0 -Is 0.0 -Id 1.0 -It tcp -Il 1000 -If 2 -Ii 2
  -Iv 32 -Pn tcp -Ps 0 -Pa 0 -Pf 0 -Po 0
```

Listing A.14: Extract of a Trace File in New Wireless Trace Format

```
n -t * -a 1 -s 1 -S UP -v circle -c black
n -t * -a 11 -s 11 -S UP -v circle -c black
n -t * -a 6 -s 6 -S UP -v circle -c black
(...)
l -t * -s 9 -d 0 -S UP -r 10000000 -D 0.002 -o right
l -t * -s 10 -d 7 -S UP -r 10000000 -D 0.002 -o left
l -t * -s 11 -d 6 -S UP -r 10000000 -D 0.002 -o left
(...)
+ -t 0.012664 -s 9 -d 0 -p tcp -e 1000 -c 0 -i 2 -a 0 -x {9.0 10.0 1
    ------- null}
- -t 0.012664 -s 9 -d 0 -p tcp -e 1000 -c 0 -i 2 -a 0 -x {9.0 10.0 1
    ------- null}
h -t 0.012664 -s 9 -d 0 -p tcp -e 1000 -c 0 -i 2 -a 0 -x {9.0 10.0 -1
    ------- null}
```

```
12 + -t 0.012664 -s 9 -d 0 -p tcp -e 1000 -c 0 -i 3 -a 0 -x {9.0 10.0 2
       -------- null}
13 - -t 0.013464 -s 9 -d 0 -p tcp -e 1000 -c 0 -i 3 -a 0 -x {9.0 10.0 2
       -------- null}
14 h -t 0.013464 -s 9 -d 0 -p tcp -e 1000 -c 0 -i 3 -a 0 -x {9.0 10.0 -1
        -------- null}
15 r -t 0.015464 -s 9 -d 0 -p tcp -e 1000 -c 0 -i 2 -a 0 -x {9.0 10.0 1
       -------- null}
16 + -t 0.015464 -s 0 -d 7 -p tcp -e 1000 -c 0 -i 2 -a 0 -x {9.0 10.0 1
       -------- null}
17 h -t 0.015464 -s 0 -d 8 -p tcp -e 1000 -c 0 -i 2 -a 0 -x {9.0 10.0 1
       -------- null}
18 r -t 0.016264 -s 9 -d 0 -p tcp -e 1000 -c 0 -i 3 -a 0 -x {9.0 10.0 2
       -------- null}
```

Listing A.15: Extract of a Trace File in NAM Trace Format

```
#!/usr/bin/perl
# =============
if( $#ARGV < 0 || $#ARGV > 0 ) { usage(); }  #execution conditions of
    perl script
#Argument ARGV[0] is the filename of the tracefile
open( FILE, $ARGV[0] ) || die "file $ARGV[0] not found.\n";
# variabledeclarations begin
my %packetSend;
my %packetReceived;
my %endToEndDelay;
my %source;
my %destination;
# variabledeclarations end
#read
while( $line = <FILE> )
{
    @word = split( '\s+', $line ); # split columns of a line and save
        into array
    #word[0]:event
    #word[1]:timestamp
    #word[2]:source
    #word[3]:destination
    #word[4]:packet type
    #word[5]:Packet Size
    #word[6]:Flags
    #word[7]:Flow ID
    #word[8]:Source Address
    #word[9]:Destination Address
    #word[10]:Sequence Number
    #word[11]:Unique Packet ID
    if(exists $packetSend{$word[11]}){
        if($word[0] eq "r")
        {
            $packetReceived{$word[11]}=$word[1];
            $destination{$word[11]}=$word[3];
        }
        if($word[0] eq "d") {
             delete $packetReceived{$word[11]};
             delete $packetSend{$word[11]};
        }
    }
    else
    {
        $packetSend{$word[11]}=$word[1];
        $source{$word[11]}=$word[2];
    }
}

```

```
#calculate end to end delay
foreach my $key1 ( keys %packetSend ) {
   $endToEndDelay{$key1}=$packetReceived{$key1}-$packetSend{$key1};
}
printToFile();
printStats();
exit;
# ----------------
sub printToFile(){
   open (FILE, ">endToEnd_v2.dat");
   flock (FILE, 2);
   foreach my $key1(sort{$packetSend{$a} cmp $packetSend{$b} } keys %
       packetSend ) {
      print FILE "$packetSend{$key1} $endToEndDelay{$key1}\n";
    }
   close FILE;
}
# ----------------
sub printStats
{
  my $i=0;
  foreach my $key1 ( keys %packetSend ) {

      print " $key1 ($source{$key1},$destination{$key1})=>  send:
          $packetSend{$key1}\t";
      print "received:    $packetReceived{$key1}\t";
      print "delay:    $endToEndDelay{$key1}\n";
      $i++;
  }
  print "There are $i received packets (whitout drop events)\n";
  exit;
}
# -------------
sub usage
{
  print STDERR "Stats:\t\tprint statistis of a .tr-tracefile\n";
  print STDERR "Usage:\t\tendToEnd.pl file.tr \n";
  exit;
}
# ============
```

Listing A.16: Perl Script for end-To-end Calculation

```
set terminal postscript enhanced color dashed
defaultplex "Helvetica" 20
show output
set out "endToEnd.ps"

unset key
set title "End-To-End Delay"
set xlabel "Simulation Time"
set ylabel "End-To-End Delay"
set style data linespoints
plot "endToEnd.dat" using 1:2 with lines
```

Listing A.17: gnuplot Script to Visualize Statistical Data

```
//qod_pkt.h
#ifndef __qod_pkt_h__
#define __qod_pkt_h__

#include <packet.h>

#define HDR_QOD_PKT(p) hdr_qod_pkt::access(p)

struct hdr_qod_pkt {
  char returnCode;
  char quote[68];
  double sendTime;

  static int offset_;
  inline static int& offset() { return offset_; }
  inline static hdr_qod_pkt* access(const Packet* p) {
   return (hdr_qod_pkt*)p->access(offset_);
  }
};
#endif
```

Listing A.18: Hands-on session 3 - new packet type QOD

```
//qod.h
#ifndef __qod_h__
#define __qod_h__

#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"
#include "ip.h"
#include "qod_pkt.h"


class QODAgent : public Agent {
 public:
  QODAgent();
  int command(int argc, const char*const* argv);
  void recv(Packet*, Handler*);

};

#endif
```

Listing A.19: Hands-on session 3 - QOD header file

```
#Create a simulator object
set ns [new Simulator]

#Setting up network topology
(...)

#Define a 'recv' function for the class 'Agent/QOD'
Agent/QOD instproc recv {from quote rtt} {
    $self instvar node_
    puts "node [$node_ id] received quote of the day: \"$quote\" from
        \
                $from with round-trip-time $rtt ms."
}

#Create two ping agents and attach them to the nodes n0 and n2
set p0_1 [new Agent/QOD]
$ns attach-agent $n0 $p0_1
set p0_2 [new Agent/QOD]
$ns attach-agent $n0 $p0_2
set p2 [new Agent/QOD]
$ns attach-agent $n2 $p2
set p3 [new Agent/QOD]
$ns attach-agent $n3 $p3
#Connect the two agents
$ns connect $p2 $p0_1
$ns connect $p3 $p0_2

#Schedule events
$ns at 0.2 "$p2 send"
$ns at 0.4 "$p3 send"
$ns at 0.6 "$p2 send"
#$ns at 0.6 "$p2 send"
#$ns at 0.6 "$p3 send"
$ns at 1.0 "finish"

#Run the simulation
$ns run
```

Listing A.20: Hands-on session 3 - simulation file qod.tcl

```
1  public class Preprocessor implements Runnable{
2
3      private ProcessorQueue queue;
4      private String processedFilename;
5      private LinkedList templist;
6      private Thread preprocessor;
7      (...)
8
9      public Preprocessor(String path,String filename){
10
11     }
12     public void startPreprocessor(){
13         preprocessor = new Thread(this);
14         preprocessor.start();
15     }
16
17     public void run(){
18         this.startPreprocessing();
19     }
20
21     /* Starts preprocessing - read in the file */
22     public void startPreprocessing(){
23         (...)
24         // Process each line individually
25         while(continueReading){
26             try{
27                 line = br.readLine();
28                 if(line==null){
29                     emptyQueue();
30                     continueReading = false;
31                 } else {
32                     processLine(line);
33                 }
34             }catch (IOException e){
35                 e.printStackTrace();
36             }
37         }
38         closeFile();
39         (...)
40         secondRun();
41         merge(this.path,this.processedFilename);
42         (...)
43     }
44     (...)
45     /**Process a line of the trace file - within each processing try
           to empty the queue if queue is full manage the queue until
           there is space for another queued elements
46      */
```

```
47  public void processLine(String line) throws IOException{
48
49      parseLine(line,1,queue);
50      (...)
51      while(!queue.isEmpty()){
52          if(queue.hasReadyElement())
53          {
54              q = queue.dequeue(); //dequeue an element
55              //Write all Elements which are in state "ready" to the
                    new file
56              (...)
57          }
58          else
59          {
60              //There is a first element which cannot be dequeued - if
                    queue is full, manage the queue otherwise take a next
                    line from the trace file
61              if(queue.isFull())
62              {
63                  //Write elements which are not yet ready and are
                        blocking the queue out to a temporary list. Go on
                        with elements that are handable
64                  (...)
65              }
66              break;
67          }
68      }
69
70  }
71  (...)
72  /** Parsing a line from the original NAM trace file and generating
        a new net.sf.vat4net.io.ProcessorQueueElement or adding the
        line to an existing one
73  */
74  private void parseLine(String line,int level,ProcessorQueue queue)
        {
75      (...)
76  }
77  (...)
78  /**   Merge the worked-through temporary list with the existing
        new trace file from the first run-through
79  */
80  public void merge(String path,String filename){
81      (...)
82  }
83  (...)
84  /**   File has to be processed a second time, there are receive
        elements left. For this do the same on the temporary list as
        done on the original trace file in the first run.
```

```
85          */
86      public void secondRun(){
87          (...)
88      }
89  }
```

Listing A.21: Preprocessor.java

```
1  public ProcessorQueueElement search(int packetID,int status,int
       packetDestination,int packetSource, String packetType)
2  {
3     for (Iterator it = queue.iterator(); it.hasNext();) {
4        q =(ProcessorQueueElement) it.next();
5        if(q.getPacketID()== packetID){
6           if(this.simulationType == WIRELESS && q.getPacketStatus() ==
                status && q.getPacketType().equals(packetType)){
7              return q;
8           }
9           if(packetSource == -1 && q.getPacketStatus() == status && q.
                getPacketDestination()==packetDestination && q.
                getPacketType().equals(packetType))
10          {
11             return q;
12          }
13          if(q.getPacketStatus() == status && q.getPacketDestination()
                ==packetDestination && q.getPacketSource()==packetSource
                && q.getPacketType().equals(packetType)){
14             return q;
15          }
16       }
17    }
18    return null;
19 }
```

Listing A.22: ProcessorQueue.java

```
1  package net.sf.vat4net.io;
2
3  public class ProcessorQueueElement {
4      (...)
5      public ProcessorQueueElement(String traceLine,boolean readyToWrite
           ,int packetID,int packetStatus,int packetDestination,int
           packetSource,String packetType){
6          //Packet Element/Trace Event
7          this.traceLine=traceLine;
8          this.readyToWrite = readyToWrite;
9          this.packetID = packetID;
10     (...)
11         this.queueElementType = QUEUEPACKETELEMENT;
12     }
13     public ProcessorQueueElement(String traceLine,boolean readyToWrite
           ){
14         //non Packet Element/Trace Event
15         this.traceLine=traceLine;
16         this.readyToWrite = readyToWrite;
17         this.queueElementType = QUEUEOTHERELEMENT;
18     }
19     (...)
20 }
```

Listing A.23: ProcessorQueueElement.java

```
1  public Buffer(int size){
2        buffer = new LinkedList();
3        maxElements = size;
4  }
5
6  public synchronized void put(Object o){
7     while(maxElements == buffer.size()){
8        try { wait(); }
9        catch (InterruptedException ie){ (...) }
10    }
11    buffer.add(o);
12    notifyAll();
13 }
14
15 public synchronized Object get(){
16    while ( buffer.isEmpty() ){
17       try { wait(); }
18       catch (InterruptedException ie){ (...) }
19    }
20    notifyAll();
21    return( buffer.removeFirst() );
22 }
```

Listing A.24: Buffer.java

```
1  public class ParsingServer {
2
3      public final String FILE_EXTENSION = ".nam";
4      public final String FILE_EXTENSION_PREPROCESSED = ".v4n";
5      (...)
6      public ParsingServer(File path) {
7          (...)
8      }
9
10     public static void main(String[] args) throws IOException {
11         //Extracting informations from args[] array
12         (...)
13         if(tmpPath.exists() && tmpPath.isDirectory()) {
14             System.out.println("starting server at " + host + ":" + port
                   + " looking for files in " + tmpPath);
15             ParsingServer parserServer = new ParsingServer(tmpPath);
16             parserServer.setup(host, port);
17         } else {
18             (...) //Error message
19         }
20     }
21     private void setup(String hostname, int port) throws IOException {
22         if(ssc != null) {
23             ssc.close();
24         }
25         ssc = ServerSocketChannel.open();
26         InetSocketAddress isa = new InetSocketAddress(InetAddress.
               getByName(hostname), port);
27         ssc.socket().bind(isa);
28         while(true) {
29             try {
30                 waitForConnection();
31             } catch (Throwable t) {
32                 (...) //Error message
33             }
34         }
35     }
36
37     private void waitForConnection() throws IOException {
38         SocketChannel sc = ssc.accept();
39         (...)
40         String filename = getFile();
41         if(filename != null) {
42             startServing(sc);
43         }
44         (...) //Error on serving/findig file
45     }
46
```

```
47      private String getFile() {
48          //Either preprocess a .nam file first and return the filename
                afterwards or return the available .v4n file name which is
                preprocessed yet
49          (...)
50          return foundFile;
51      }
52      (...)
53      private void startServing(SocketChannel sc) {
54          try {
55              byte networkConf;
56              long startTime = System.currentTimeMillis();
57              ObjectOutputStream out = new ObjectOutputStream(sc.socket().
                    getOutputStream());
58              (...)
59              FileReader in = new FileReader(f);
60              while(this.proc!=null && !this.proc.finished()){
61                  (...) // Inform client, that file is on preprocessing
                        state
62              }
63              while((line = bIn.readLine()) != null){
64                  commandAvailable(sc,out);
65                  if(connectionClosedByClient){
66                      break;
67                  }
68                  out.writeObject(line);
69                  out.flush();
70                  out.reset();
71              }
72              out.writeObject(eof);
73              out.flush();
74              out.reset();
75              bIn.close();
76              while(true && !connectionClosedByClient){
77                  out.writeObject("NOD");
78                  commandAvailable(sc,out);
79                  if(connectionClosedByClient){
80                      break;
81                  }
82              }
83              out.close();
84          } catch (IOException e) {
85              (...) // Error message
86          }
87      }
88
89      public void commandAvailable(SocketChannel sc,ObjectOutputStream
            out){
90          try{
```

```
91              if (sc.socket().getInputStream().available() > 0) {
92                  int command = sc.socket().getInputStream().read();
93                  if(command == 0 && !this.incomingStatsCommand){
94                      this.connectionClosedByClient = true;
95                  }else if (command == 2 && !this.incomingStatsCommand) {
96                      this.incomingStatsCommand = true;
97                  }
98                  else if(command==3 && this.incomingStatsCommand){
99                      (...)
100                     Class pluginClass = this.getClass().getClassLoader().
                            loadClass(stats[0]);
101                     this.plugin = (StatisticsPlugin)pluginClass.
                            newInstance();
102                     this.plugin.setStatisticDataSetReady(false);
103                     this.plugin.calculate(this.loadedFile,Integer.parseInt
                            (stats[1]),stats[2]);
104                     (...)
105                 }
106                 else if(this.incomingStatsCommand){
107                     this.statCommand += (char)command;
108                 }
109                 (...)
110
111             }else if(this.plugin!=null){
112                 //Plugin is ready with calculation and can send data out
113                 if(this.plugin.getStatisticDataSetReady() && this.
                        dataSent){
114                     this.plugin.sendStatisticDataSet(out);
115                     (...)
116                 }
117             }
118         } catch (IOException e) {
119             (...) // Error Message
120         }
121     }
122 }
```

Listing A.25: ParsingServer.java

```
1  public class ParserStub implements Runnable {
2      (...)
3      public boolean connect(String host, int port) {
4          (...)
5          sc = SocketChannel.open();
6          InetSocketAddress isa = new InetSocketAddress(InetAddress.
               getByName(host), port);
7          sc.connect(isa);
8          successful = true;
9          (...)
10         return successful;
11     }
12     public void startLoadingData(Buffer buffer){
13
14         this.buffer = buffer;
15         readingFileThread = new Thread(this);
16         readingFileThread.start();
17     }
18     public void run() {
19         (...)
20         sc.socket().getOutputStream().write((byte)1);
21         ObjectInputStream ois = new ObjectInputStream(sc.socket().
               getInputStream());
22         Object streamObject;
23         while((streamObject = ois.readObject())!=null){
24             if(streamObject instanceof String){
25                 if(streamObject.equals("V-t * preprocessing")){
26                     //preprocessing debug message
27                 }
28                 else if(streamObject.equals("NOD")){
29                     //no object debug message
30                 }
31                 else if(streamObject.equals("stats")){
32                     //statistic object debug message
33                 }
34                 else if(streamObject.equals("EOF")){
35                     (...)
36                     buffer.put("EOF");
37                 }
38                 else{
39                     buffer.put((String)streamObject);
40                 }
41             }
42             else if(Class.forName(this.statPlugin.getReturnType()).
                   isInstance(streamObject)){
43                 this.statPlugin.showStat(streamObject);
44             }
45             streamObject =null;
```

```
46          }
47          (...)
48      }
49      public void sendStatisticRequest(StatisticsPlugin statPlugin,
            String command,String values) {
50          this.statPlugin = statPlugin;
51          (...)
52          sc.socket().getOutputStream().write((byte)2);
53          String statRequest = statPlugin.getClass().getName()+"%"+
                command+"%"+values;
54          sc.socket().getOutputStream().write(statRequest.getBytes());
55          sc.socket().getOutputStream().write((byte)3);
56          sc.socket().getOutputStream().flush();
57          (...)
58      }
59      (...)
60  }
```

Listing A.26: ParserStub.java

```
1  public class MnuLoadActionListener implements ActionListener {
2      (...)
3      public MnuLoadActionListener(MainFrame frmRoot) {
4          this.frmRoot = frmRoot;
5          this.buffer = new Buffer(Preferences.getInt("buffer_size",300))
               ;
6      }
7      public void actionPerformed(ActionEvent e) {
8          if(Preferences.getBoolean("use_pref_addr", false)
9              // Connect to a VAT4Net server without SSH enabled
10             (...)
11
12         } else if(Preferences.getBoolean("enableSSHTunnel",true)) {
13             int tmpPort = setupSshTunnel(-1);
14             (...)
15             if(tmpPort > 0) {
16                 addr = "localhost";
17                 port = tmpPort;
18
19             } else {
20                 (...) // Errror message
21             }
22         } else {
23             // Connect to a VAT4Net server without SSH enabled
24             (...)
25         }
26         if(port > 0 && addr != null) {
27             (...)
28             this.parser = new ParserStub();
29             frmRoot.setDataSource(parser);
30             parser.connect(addr,port);
31             parser.startLoadingData(this.buffer);
32             V4N_Parser V4Nparser = new V4N_Parser();
33             TimeController tc = new TimeController(this.buffer,V4Nparser
                   ,frmRoot);
34             tc.startAnimation();
35             (...)
36         }
37     }
38     public void retryToConnect(int connectionType){
39         (...)
40     }
41
42     /* Sets up the SSH tunnel and returns the local port that is
           forwarded after successful setup.*/
43     private int setupSshTunnel(int tempport) {
44         (...)
```

```
45          String username = Preferences.getString("sshUsername", "no
                username available");
46          String password = Preferences.getString("sshPassword", "no
                password available");
47          String hostname = Preferences.getString("sshHostname", "no
                hostname available");
48          String rHost = Preferences.getString("sshRHost", "no rhost
                available");
49          int rPort = Preferences.getInt("sshRPort", -1);
50          (...)
51          Connection conn = new Connection(hostname);
52          conn.connect(null, 0, TIMEOUT_MILLIS);
53
54          boolean isAuthenticated = conn.authenticateWithPassword(
                username, password);
55          if(isAuthenticated) {
56             conn.createLocalPortForwarder(localport, rHost, rPort);
57          } else {
58             (...) // Errror message
59          }
60          (...)
61          return localport;
62       }
63       (...)
64    }
```

Listing A.27: MnuLoadActionListener.java

```
1  public class TimeController implements Runnable{
2      (...)
3      public TimeController(Buffer buffer,V4N_Parser parser,MainFrame
           frmRoot){
4          (...)
5          this.timeStep = Preferences.getDouble("
               animation_time_step_start", 0.05);
6          this.period=1000/Preferences.getInt("frames_per_second", 5);
7          (...)
8      }
9      public void startAnimation(){
10         netPanel = frmRoot.loadNetwork(this);
11         parser.initializeParser(this,netPanel);
12         timeControllerThread = new Thread(this);
13         timeControllerThread.start();
14     }
15     public void run()
16     {
17         (...)
18         boolean networkIsInitialized = false;
19         double parserTimeStamp=0.0;
20         while(!networkIsInitialized)
21         {
22             line = (String)buffer.get();
23             (...)
24             parser.parseLine(line);
25             if(!networkIsInitialized && parserTimeStamp>this.
                   animationStartTime){
26                 networkIsInitialized = true;
27                 netPanel.setResetStatus(true);
28                 netPanel.setNetwork(network);
29                 network.update(this.time);
30                 network.connectLinks();
31                 netPanel.initialiseNetworkPanel();
32                 netPanel.updateTimeDisplay();
33                 netPanel.setNetworkInitStatus(true);
34                 netPanel.repaint();
35             }
36         }
37         double beforeTime, afterTime, timeDiff, sleepTime;
38         boolean endOfFile = false;
39         while(this.time<this.endTime)
40         {
41             beforeTime = System.currentTimeMillis();
42             synchronized(this)
43             {
44                 while(this.isAnimationPaused())
45                 { (...) /*waiting */ }
```

```
46              }
47              netPanel.setResetStatus(false);
48              while(parserTimeStamp<(this.time+1*this.getTimeStep()) && !
                    endOfFile){
49                  line = (String)buffer.get();
50                  if(line.equals("EOF")){
51                      endOfFile = true;
52                  }
53                  else{
54                      parser.parseLine(line);
55                      parserTimeStamp = Double.valueOf(getOption(line,'t')).
                            doubleValue();
56                  }
57              }
58              network.update(this.time);
59              netPanel.updateTimeDisplay();
60              if(!this.jumpInSimulation){
61                  netPanel.repaint();
62                  afterTime = System.currentTimeMillis();
63                  timeDiff = afterTime - beforeTime;
64                  sleepTime = (period - timeDiff);
65                  //let thread sleep for the remaining time (1000/fps not
                        used)
66                  if (sleepTime > 0) {
67                      (...) // sleep
68                  }
69                  (...)
70                  beforeTime = System.currentTimeMillis();
71              }
72              else{
73                  (...) // Do not repaint or sleep when fast-forward
74              }
75              this.time = this.time + this.getTimeStep();
76          }
77          (...)
78      }
79      (...) //set and get methods
80  }
```

Listing A.28: TimeController.java

```
1  public class NetworkPanel extends JPanel {
2      (...)
3      public void initialiseNetworkPanel(){
4          if(this.getNetwork().getWirelessRangeX()!=null && this.
               getNetwork().getWirelessRangeY()!=null)
5          {
6              //for wireless networks the dimension is given by the
                   wireless range
7              network.nodePlacing2(this.getNetwork().getWirelessRangeX().
                   intValue(),this.getNetwork().getWirelessRangeY().intValue
                   ());
8              this.setPreferredSize(new Dimension((int)(this.getNetwork().
                   getWirelessRangeX().intValue())/*this.stretchFactor)*/,(
                   int)(this.getNetwork().getWirelessRangeY().intValue()/*
                   this.stretchFactor*/)));
9              this.getScrollPane().setViewportView(this);
10         }
11         else
12         {
13             //for other networks nodes are placed over the viewable
                   space (panel in its inital appearance)
14             network.nodePlacing2((int)this.panelWidth,(int)this.
                   panelHeight);
15             (...) //initialize panel
16         }
17     }
18     public void paintComponent(Graphics g)
19     {
20         (...)
21         if(this.networkIsInitialized){
22             (...)
23             network.drawNetwork(g2);
24             (...)
25         }
26         this.setResetStatus(false);
27     }
28     (...) // set and get methods
29 }
```

Listing A.29: NetworkPanel.java

```
1  public class V4N_Parser {
2     (...)
3     public void parseLine(String line){
4        switch(line.charAt(0)){
5           (...)
6           case 'n': // Node
7                 nodeSetup(generateHashMap(splitLine(line)));
8
9              break;
10          case 'l': // Link
11                linkSetup(generateHashMap(splitLine(line)));
12             break;
13          case 'h': // Hop
14          case 'r': // Receive
15          case 'd': // Drop Line
16          case '+': // Enqueue Packet
17          case '-': // Dequeue Packet
18                packetSetup(line);
19             break;
20          (...)
21          default:
22             break;
23       }
24    }
25    private void nodeSetup(HashMap nodeTable) {
26       Node node;
27       int nodeID = Integer.parseInt(nodeTable.get("-s").toString());
28       //check if node already exists, when yes, try to make an update
             element
29       if(tc.getNetwork().existNode(nodeID)){
30          node = tc.getNetwork().getNode(new Integer(nodeID));
31       }
32       else
33       {
34          node = new Node(nodeID, netPanel);
35          tc.getNetwork().addNode(node);
36       }
37       String time = nodeTable.get("-t").toString();
38       NodeUpdateElement element = new NodeUpdateElement(time);
39       (...)
40       element.setXCoord(new Double(Double.parseDouble(nodeTable.get("
            -x").toString())));
41       element.setYCoord(new Double(Double.parseDouble(nodeTable.get("
            -y").toString())));
42       element.setMovementDuration(new Double(Double.parseDouble(
            nodeTable.get("-T").toString())));
43       (...)
44       node.addUpdateElement(element);
```

```
45      }
46      private void linkSetup(HashMap linkTable){
47          (...)
48      }
49      public void packetSetup(String line){
50          Integer packetIDint = new Integer(Integer.parseInt(getOption(
                line, 'i')));
51          Packet packet = tc.getNetwork().getPacket(packetIDint);
52          if(packet == null)
53          {
54              packet = new Packet(packetIDint.intValue(),netPanel);
55              tc.getNetwork().addPacket(packet); //network is here
                    NetworkData notNetwork
56          }
57          (...)
58          //switching between different packet trace results and special
                cases
59          (...)
60          if(packetElement.get(0).toString().charAt(0)=='+' &&
                packetElement.get(1).toString().charAt(0)=='-' /* (...)*/){
61              (...)
62              element = new PacketUpdateElement(packet,startTime,endTime,
                    source,destination,packetType,lastHop,netPanel);
63              element.setEnqueueTime(getOption(packetElement.get(0).
                    toString(),'t'));
64              element.setDequeueTime(getOption(packetElement.get(1).
                    toString(),'t'));
65              element.setHopTime(getOption(packetElement.get(2).toString()
                    ,'t'));
66          }
67          (...)
68          packet.addUpdateElement(element);
69      }
70      (...)
71  }
```

Listing A.30: V4N_Parser.java

```
1  public class Network {
2     private HashMap nodes = new HashMap(); //Node List
3     //All other network elements are held in a list
4     (...)
5
6     /*Draw the Network onto the panel */
7     public synchronized void drawNetwork(Graphics2D g)
8     {
9        //draw Network environment
10       networkshape.draw(g);
11       // draw nodes
12       for (Iterator it = nodes.values().iterator(); it.hasNext();) {
13          ((Node) it.next()).draw(g);
14       }
15       //draw all other network elements
16       (...)
17    }
18    /*Updates all object in the network*/
19    public synchronized void update(double time)
20    {
21       /** Update Nodes first*/
22       Node node;
23       for (Iterator it = nodes.values().iterator(); it.hasNext();) {
24          node = (Node) it.next();
25          if(node.doUpdate()){
26             node.update(time);
27          }
28       }
29       //updating all other network elements
30       (...)
31    }
32    //Adding, removing and searching methods
33    (...)
34
35    /*New node placing approach by THOMAS M. J. FRUCHTERMAN* AND
         EDWARD M. REINGOLD Graph Drawing by Force-Directed Placing */
36    public void nodePlacing2(int width,int height){
37       (...)
38    }
39    (...)
40    /*Connect Links to Nodes with their source and destination ID*/
41    public void connectLinks()
42    {
43       for(int i = 0 ; i < networkLink.size() ; i++)
44       {
45          Node firstNode = (Node)nodes.get(new Integer(((Link)
              networkLink.elementAt(i)).getSource()));
```

```
46              Node secondNode = (Node)nodes.get(new Integer(((Link)
                    networkLink.elementAt(i)).getDestination()));
47              ((Link)networkLink.elementAt(i)).getShape().setNodes(
                    firstNode, secondNode);
48          }
49      }
50      (...)
51  }
```

Listing A.31: Network.java

```
1  public class Node {
2      (...)
3      public Node(int nodeID,NetworkPanel panel){
4          (...)
5          this.nodeID = nodeID;
6          this.updateEvents = new Vector();
7          (...)
8          //Node Shape
9          nodeshape = new NodeShape(this);
10         //Event Listener (Mouse)
11         panel.addMouseListener(new NodeMouseAdapter(nodeshape,panel,
               mousePoint));
12          panel.addMouseMotionListener(new NodeMouseMotionAdapter(
                nodeshape,panel,mousePoint));
13     }
14     /**Draw method for node, draws the node in actual state, the draw
           method is defined in NodeShape*/
15     public void draw(Graphics2D g)
16     {
17         nodeshape.draw(g);
18     }
19
20     /**Update method for the node. Updates node at given animation
           time.*/
21     public void update(double time)
22     {
23         (...)
24         //Update node position
25         this.getNodeShape().setXCoord(this.startPointX +((percOfWay
               /100)*(((this.startPointX + (this.getXVelocity()*this.
               getMovementDuration()))-this.startPointX))));
26         this.getNodeShape().setYCoord(this.startPointY +((percOfWay
               /100)*(((this.startPointY + (this.getYVelocity()*this.
               getMovementDuration()))-this.startPointY))));
27         (...)
28         //Update color of node shape
29         nodeshape.setColor(((NodeUpdateElement)this.updateEvents.get(i)
               ).getColor());
30         (...)
31     }
32     /* Adding an update element (event) to a node */
33     public void addUpdateElement(NodeUpdateElement element){
34         (...)
35     }
36 }
```

Listing A.32: Node.java

```
1  public class NodeShape
2  {
3      public NodeShape(Node node)
4      {
5          this.x = 0;
6          this.dispX = 0;
7          this.dx = 0;
8          this.y = 0;
9          this.dispY = 0;
10         this.dy = 0;
11         this.node = node;
12         this.nodeSelected = false;
13         (...)
14         this.wirelessRange = Preferences.getInt(this, "wireless_range",
               250);
15     }
16     public void draw(Graphics2D g2)
17     {
18         if(node.getWirelessStatus()==Node.WIRELESS_NODE)
19         {
20             //drawing wireless range around the node
21             (...)
22         }
23         //drawing the shape (strokes, circles, text...)
24         (...)
25     }
26     (...)
27 }
```

Listing A.33: NodeShape.java

```
area := W * L; { W and L are the width and length of the frame }
G := (V, E); { the vertices are assigned random initial positions }
k := sqrt(area/|V|);
function fa(z) := begin return x^2/k end;
function fr(z) := begin return k^2/z end;

for i := 1 to iterations do begin
{ calculate repulsive forces}
    for v in V do begin
    { each vertex has two vectors: .pos and .disp }
        v.disp := 0;
        for u in V do
            if (u # v) then begin
            { D is short hand for the difference}
            { vector between the positions of the two vertices )
            D := v.pos - u.pos;
            v.disp := v.disp + (D/|D|) * fr(|D|)
        end
    end

    { calculate attractive forces }
    for e in E do begin
        { each edge is an ordered pair of vertices .v and .u }
        D  := e.v.pos - e.u.pos
        e.v.disp := e.v.disp - (D/|D|) * fa(|D|);
        e.u. disp := e.u.disp + (D/|D|) * fa(|D|)
    end
    { limit the maximum displacement to the temperature t }
    { and then prevent from being displaced outside frame}
    for v in V do begin
        v.pos := v.pos + (v.disp/|v.disp|) * min(v.disp,t);
        v.pos.x := min(W/2, max(-W/2, v.pos.x));
        v.pos.y := min(L/2, max(-L/2, v.pos.y))
    end
    { reduce the temperature as the layout approaches a better
       configuration }
    t := cool(t)
end
```

Listing A.34: NodePlacing Algorithm by Fruchterman and Reingold [17]

```
1  public class Queue {
2      (...)
3      public Queue(int source, int destination,NetworkPanel panel,Link
           link)
4      {
5          (...)
6      }
7      public Queue(int source, NetworkPanel panel)
8      {
9          (...)
10     }
11     (...)
12     /**Enqueue packet on this queue */
13     public void enqueue(PacketUpdateElement packetElement) {
14         queuedPackets.addElement(packetElement);
15         this.queueshape.addQueueHeight(packetElement.getShape().
               getRadius()*2);
16     }
17
18     /**Dequeue packet on this queue */
19     public void dequeue(PacketUpdateElement packetElement) {
20         queuedPackets.removeElement(packetElement);
21         this.queueshape.minusQueueHeight(packetElement.getShape().
               getRadius()*2);
22     }
23     (...)
24 }
```

Listing A.35: Queue.java

```
public class Packet {
    (...)
    public Packet(int packetID,NetworkPanel panel){

        this.updateEvents = new Vector();
        this.packetID = packetID;
        this.panel = panel;
        this.packetCompletelyDone = false;
    }
    public void draw(Graphics2D g){
        for(int i = 0;i<updateEvents.size();i++){
            ((PacketUpdateElement)updateEvents.get(i)).draw(g);
        }
    }
    (...)
    /*Update method for v4n packet format */
    public void update(double time)
    {
        for(int i = 0;i<updateEvents.size();i++){
            if(((PacketUpdateElement)updateEvents.get(i)).doUpdate() &&
                ((PacketUpdateElement)updateEvents.get(i)).getStartTime()
                <=time)
            {
                ((PacketUpdateElement)updateEvents.get(i)).update(time);
            }
            else if(((PacketUpdateElement)updateEvents.get(i)).doUpdate
                ()){
                break;
            }
            else{
                updateEvents.remove(i);
            }
        }
    }
    //get and set methods
    (...)
}
```

Listing A.36: Packet.java

```
1  public class PacketUpdateElement extends UpdateElement {
2      (...)
3      public PacketUpdateElement(Packet packet,String startTime,String
           endTime,int source, int destination, int packetType,boolean
           lastPacketHop,NetworkPanel panel){
4          this.packet = packet;
5          this.srcID = source;
6          this.dstID = destination;
7          this.startTime = parseTimeStamp(startTime);
8          this.endTime = parseTimeStamp(endTime);
9          this.packetType = packetType;
10         this.lastPacketHop = lastPacketHop;
11         packetshape = new PacketShape(this);
12         (...)
13     }
14     public void draw(Graphics2D g){
15         (...) // invoke the adequate draw method for the packet type (
               normal, on air, reduced animation complexity)
16     }
17
18     public void update(double time){
19         if(panel.getNetwork().getWirelessRangeX()!=null){
20             if(((Node)panel.getNetwork().getNode(new Integer(this.srcID)
                   )).getWirelessStatus()==1){
21                 this.packetType = PACKET_ONAIR;
22             }
23         }
24         //ENQUEUE
25         if(time >= this.startTime){
26             if(!this.enqueued){
27                 handleEnqueue(time);
28             }
29
30         }
31         //DEQUEUE
32         if(time >=this.dequeueTime){
33             if(!this.dequeued){
34                 handleDequeue(time);
35             }
36         }
37         //HOP
38         if(time >= this.hopTime){
39             if(!this.hopped){
40                 handleHop(time);
41             }
42             else{
43                 calculatePart(time);
44             }
```

```
45          }
46          //RECEIVE
47          if(time >= this.endTime && this.dropTime<0){
48              if(!this.received){
49                  handleReceive(time);
50              }
51          }
52          //DROP
53          if(time>=this.dropTime && this.dropTime>=0){
54              if(!this.dropped){
55                  handleDrop(time);
56              }
57              else{
58                  calculateDropPart(time);
59              }
60          }
61      }
62      private void calculatePart(double time){
63          (...) // calculate percent of way on a link  yet covered
64      }
65      private void calculateDropPart(double time){
66          (...) // calculate percent of drop way yet covered
67      }
68      (...)
69      private void handleEnqueue(double time){
70          (...) // do the handle actions on a packet state
71      }
72
73      private void handleDequeue(double time){
74          (...) // do the handle actions on a packet state
75      }
76
77      private void handleHop(double time){
78              (...) // do the handle actions on a packet state
79      }
80      private void handleReceive(double time){
81          (...) // do the handle actions on a packet state
82      }
83      private void handleDrop(double time){
84          (...) // do the handle actions on a packet state
85      }
86      //get and set methods
87      (...)
88  }
```

Listing A.37: PacketUpdateElement.java

```
1  public class DelayPlugin implements StatisticsPlugin,Runnable  {
2      (...)
3      public void setMainFrame(MainFrame mainFrame) {
4          this.mainFrame = mainFrame;
5          JComponent tmpComp = mainFrame.getNetworkPnl();
6          if(tmpComp instanceof NetworkPanel) {
7              netPnl = (NetworkPanel) tmpComp;
8              endToEndDelayItem.setEnabled(true);
9              (...)
10         } else {
11             endToEndDelayItem.setEnabled(false);
12             (...)
13         }
14     }
15     public JComponent getComponent() {
16         statPnl = new StatisticsPanel();
17         pane = new JScrollPane(statPnl);
18         return pane;
19     }
20     public void refreshComponent(ChartPanel chartPanel){
21         mainFrame.setStatisticPanel(chartPanel);
22     }
23     public String getName() {
24         return "Delay Plugin";
25     }
26     public JMenu getActionMenu() {
27         String name = DelayPlugin.class.getName();
28         JMenu menAction = new JMenu(name.substring((name.indexOf('.') >
               0)? name.indexOf('.'):0));
29
30         endToEndDelayItem = new JMenuItem("End-to-end delay");
31         endToEndDelayItem.addActionListener(new ActionListener() {
32             public void actionPerformed(ActionEvent e) {
33                 calcDelay(END_TO_END);
34             }
35         });
36         menAction.add(endToEndDelayItem);
37         (...)
38         return menAction;
39     }
40     private void calcDelay(int type) {
41         (...)
42         List selectedNodes = netPnl.getSelectedNodes();
43         //Error message when settings do not achieve the specification
               of the plugin
44         (...)
45         //Let the server do the statistical calculations
46         if(mainFrame.getDataSource() instanceof ParserStub){
```

```
47          if(type == END_TO_END){
48              this.title = "End to End Delay";
49              List.class.cast(this.returnType);
50          }
51          (...)
52
53          ((ParserStub)mainFrame.getDataSource()).sendStatisticRequest
                (this,String.valueOf(type),((Node)selectedNodes.get(0)).
                getNodeID()+","+((Node)selectedNodes.get(1)).getNodeID())
                ;
54      }
55      //The calculation is done one the local computer
56      else{
57          if(type == END_TO_END){
58              this.title = "End to End Delay";
59              this.calculate((File)mainFrame.getDataSource(),type,((
                    Node)selectedNodes.get(0)).getNodeID()+","+((Node)
                    selectedNodes.get(1)).getNodeID());
60          }
61          (...)
62      }
63  }
64  public void showStat(Object statsData){
65      if(statsData instanceof List){
66          XYSeriesCollection dataset = new XYSeriesCollection();
67          XYSeries series1 = new XYSeries("First");
68          double data[][] =(double[][])((List)statsData).toArray(new
                double[2][((List)statsData).size()]);
69          for(int i = 0; i< data[0].length;i++){
70              series1.add(data[0][i],data[1][i]);
71          }
72          dataset.addSeries(series1);
73          JFreeChart chart =  ChartFactory.createScatterPlot(
74                  //chart settings
75                  (...)
76                  dataset,
77          );
78          chartPanel = new ChartPanel(chart,false);
79          this.refreshComponent(chartPanel);
80          this.calculateStatisticThread = null;
81      }
82  }
83  public void calculate(File file,int type, String command){
84      //Start calculation thread
85      (...)
86  }
87  (...)
88  public void run(){
89      if(this.type == END_TO_END){
```

```
90              (...)
91              this.dataset = calculateEndToEnd(bIn,Integer.parseInt(nodes
                    [0]),Integer.parseInt(nodes[1]));
92              if(mainFrame.getDataSource() instanceof File){
93                  this.showStat(this.dataset);
94              }
95              (...)
96          }
97          (...)
98      }
99      (...)
100     private List calculateEndToEnd(BufferedReader in, int node1, int
            node2) {
101         double [][]data = new double [0][0];
102         // Do End-to-End calcualtion
103         (...)
104         List dataset = Arrays.asList(data);
105         this.setStatisticDataSetReady(true);
106         return dataset;
107     }
108     (...)
109     public void sendStatisticDataSet(ObjectOutputStream out){
110         (...)
111         out.writeObject(this.dataset);
112         out.flush();
113         out.reset();
114         (...)
115     }
116
117 }
```

Listing A.38: DelayPlugin.java

```
1  public class PluginLoader {
2     private Main main;
3     private ArrayList plugins = new ArrayList();
4
5     public PluginLoader(Main main) {
6        this.main = main;
7     }
8
9     public ArrayList getPluginList() {
10       URL listURL = FileLoaderUtil.createURL("plugins.properties");
11       (...)
12       Properties pluginsFile = new Properties();
13       pluginsFile.load(listURL.openStream());
14
15       for (Enumeration keyEnum = pluginsFile.keys(); keyEnum.
             hasMoreElements();) {
16          String key = (String) keyEnum.nextElement();
17          Class pluginClass = this.getClass().getClassLoader().
                loadClass(key);
18          plugins.add(pluginClass.newInstance());
19       }
20       (...)
21       return plugins;
22    }
23 }
```

Listing A.39: PluginLoader.java