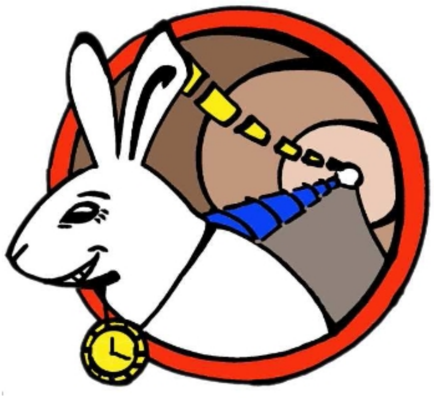
QUEUE SCHEDULING IN THE WHITE RABBIT SWITCH

SOLVEIGH MATTHIES



Chasing rabbits...

Master of Computer Science

Lehrgebiet Kooperative Systeme
FernUniversität Hagen

2012

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

CERN	**C**onseil **E**uropéen pour la **R**echerche **N**ucléaire
CoS	**C**lass **o**f **S**ervice
ESR	**E**xperimental **S**torage **R**ing
FAIR	**F**acility for **A**ntiproton and **I**on **R**esearch
FOPI	**Fo**ur **Pi**
FRS	**Fr**agment-**S**eparator
GPS	**G**lobal **P**ositioning **S**ystem
GSI	**G**esellschaft für **S**chwer**i**onenforschung
HADES	**H**igh **A**cceptance **D**i-**E**lectron **S**pectrometer
HIT	**H**eidelberger **I**onenstrahl-**T**herapiezentrum

IDE	**I**ntegrated **D**evelopment **E**nvironment
OMNeT	**O**bjective **M**odular **N**etwork **T**estbed in C++
PC	**P**ersonal **C**omputer
PHELIX	**P**etawatt **H**igh **E**nergy **L**aser for Heavy **I**on E**x**periments
PHY	**Phy**sical **L**ayer
PLL	**P**hase-**L**ocked **L**oop
PPS	**P**ulse **P**er **S**econd
PTP	IEEE 1588 **P**recision **T**ime **P**rotocol
QoS	**Q**uality **o**f **S**ervice
RS232	**R**ecommended **S**tandard 232
SHIP	**S**eparator for **H**eavy **I**on reaction **P**roducts
SyncE	**Sync**hronous **E**thernet
UNILAC	**Uni**versal **L**inear **Ac**celerator
UTC	**U**niversal **C**oordinated **T**ime
VHDL	**V**ery **H**igh Speed Integrated Circuit Hardware **D**escription **L**anguage
WR	**W**hite **R**abbit
WRS	**W**hite **R**abbit (Ethernet) **S**witch

1 INTRODUCTION

1.1 BACKGROUND

The **GSI Helmholtz Centre for Heavy Ion Research GmbH** (former name: **G**esellschaft für **S**chwer**i**onenforschung)[1] close to Darmstadt, Hesse runs a unique particle accelerator complex with the focus on studying heavy ions. Basic scientific research as well as applied research in atomic physics and related natural science fields such as plasma physics, biophysics and nuclear structure and reactions are performed. Another field of study is medical research. The facility is used by scientists and researchers from around the world for different experiments.

The heavy ion accelerator facility consists of **Uni**versal **L**inear **Ac**celerator (UNILAC), the Universal Linear Accelerator (energy of 2-11,4 MeV per nucleon), SIS18, the heavy ion synchrotron (1-2 GeV/u) and **E**xperimental **S**torage **R**ing (ESR), the Experimental Storage Ring (0,5-1 GeV/u). Attached to the accelerator complex are the experiments **S**eparator for **H**eavy **I**on reaction **P**roducts (SHIP), an electro-magnetic filter for separation and identification of fusion products such as heavy elements, **Four Pi** (FOPI), a large particle detector for research of nuclear reactions, **H**igh **A**cceptance **D**i-**E**lectron **S**pectrometer (HADES), a di-electron spectrometer to investigate the properties of hadrons under high pressure and temperature, **Fr**agment-**S**eparator (FRS), a segmentation unit that is used to produce radioactive isotopes and fusion samples, a radiation treatment unit for radiation of cancer tumours with accelerated carbon ions.

Accelerators

Experiments

Additional to the accelerator facilities and the experiments **G**esellschaft für **S**chwer**i**onenforschung (GSI) hosts a high power laser, **P**etawatt **H**igh **E**nergy **L**aser for Heavy **I**on E**x**periments (PHELIX). Intense laser pulses in petawatt range are used to heat plasma and can be combined with heavy ion beams to provide innovative research possibilities. During the past 25 years six new chemical elements were discovered at GSI (Bohrium, 1981; Meitnerium, 1982; Hassium, 1984; Darmstadtium, 1994; Roentgenium, 1994; Coperinicium, 1996). A new and highly effective type of cancer tumour therapy using accelerated carbon ion beams was developed. The technique allows to irradiate tumours that are close to vital organs and therefore not to treat using conventional therapies. Established at the University of Heidelberg Medical Center is **H**eidelberger **I**onenstrahl-**T**herapiezentrum (HIT)[2] that began

PHELIX

New elements

HIT

1 http://www.gsi.de

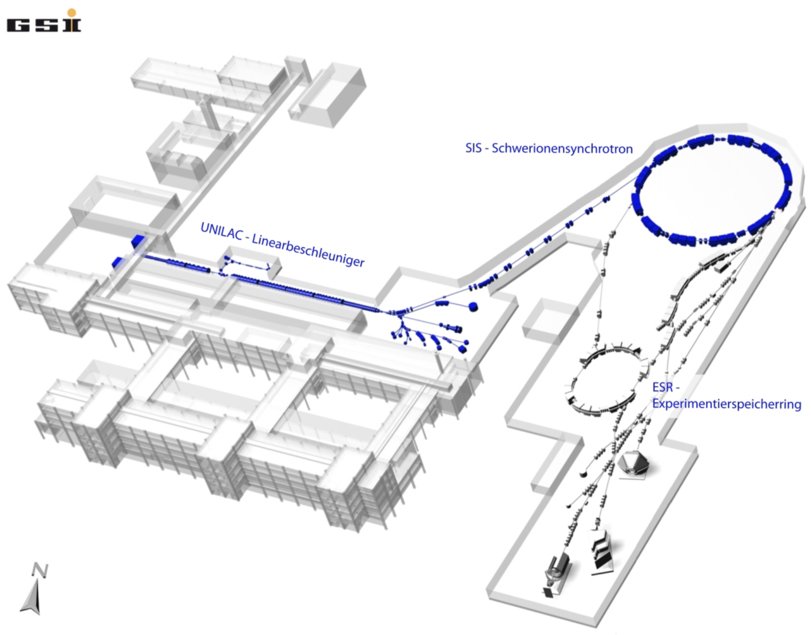

GSI
SIS - Schwerionensynchrotron
UNILAC - Linearbeschleuniger
ESR - Experimentierspeicherring
N

Figure 1: Simplified scheme of GSI's accelerator complex

treating patients in 2009. End of 2011 the construction of one of the largest research facilities in Europe started adjacent to GSI. **F**acility for **A**ntiproton and **I**on **R**esearch (FAIR)[3] will be realized as a international accelerator center.

FAIR

1.2 TIMING SYSTEM

A timing system is the essential ingredient for an accelerator. Each machine in the accelerator complex is able to interoperate with another because of sharing a common and highly precise timing system. Accelerators and large-scale scientific experiments require a precise timing due to time-critical operations. Precise and reliable timing with sub-nanosecond accuracy is the precondition for synchronizing the devices within the accelerator. Being a special form of networks, timing systems provide a general time in a distributed environment to ensure that all connected devices use the same time. For the future accelerator facility a new timing system - White Rabbit[4] - will be established.

White Rabbit

...

2 http://www.klinikum.uni-heidelberg.de/Heidelberger-Ionenstrahl-Therapie-HIT.106580.0.html

3 http://www.fair-center.de/

4 http://www.ohwr.org/projects/white-rabbit

2 WHITE RABBIT

2.1 THE WHITE RABBIT PROJECT

White Rabbit [3] [4] is the code name for an internationally collaborative project. GSI and **C**onseil **E**uropéen pour la **R**echerche **N**ucléaire (CERN)[1] as well as other partners from industry work together in an international collaboration to develop a fully deterministic Ethernet-based network for general purpose data transfer and synchronization (e.g. timing). The two main technologies used are

- **Synch**ronous **E**thernet (SyncE) to realize syntonization
- White Rabbit Precision Time Protocol (enhanced IEEE 1588 **P**recision **T**ime **P**rotocol (PTP) (IEEE 1588)) .

Synchronization

The sub-nanosecond synchronization is achieved in parallel and transparently to the data exchange.
The individual clocks of the switches and nodes in the timing network may be slightly different though all the devices have the same frequency of 125 MHz. All devices are supposed to have the same clock which is generated by the system's timing master. To have no extra traffic cost the clock is encoded in the Ethernet carrier and recovered by the **P**hase-**L**ocked **L**oop (PLL) of the **Ph**ysical **L**ayer (PHY) of the device. The offsets between the clocks are removed after measuring and compensating the delays introduced by the links. That way a common notion of time for all the devices in the network is achieved and they are able to operate synchronous.

Timing Master

Main Goals

The main goals of the White Rabbit project are

- Synchronization with sub-nanosecond accuracy of approximately 1000 nodes over fibre and copper lengths of up to 10 km.
- Scalable and modular platform without requiring specialized configuration and maintenance (automatic reconfiguration).
- Determinism and Reliability: deterministic and robust delivery of high priority messages with low transport delay is necessary.
- Open design to be free from a particular hard- and software vendor.

In short White Rabbit will combine the real-time performance of industrial networks and flexibility and scalability of Ethernet with the

1 http://www.cern.ch

accuracy of dedicated timing systems. The **W**hite **R**abbit (WR) technology will be made from commercially available standard network gear. White Rabbit will be easy to use by relying on well established standards. The hardware designs as well as the source code will be publicly available.

White Rabbit@FAIR

White Rabbit has been chosen as the base for the future timing system of GSI's FAIR complex. At CERN White Rabbit is the choice for the renovated control system of the injector chain which consists of Linacs, PSB and PS). White Rabbit has also attracted attention in the commercial world. Several companies[2] already started developing White Rabbit-compatible hardware, software and drivers which will be commercially available.

2.1.1 *The White Rabbit Switch*

White Rabbit Switch

Class of Service

One of the key devices in the timing network will be the **W**hite **R**abbit (Ethernet) **S**witch (WRS). White Rabbit requires dedicated network switches for propagation of clock frequency / phase as well as time stamps. Within a single physical connection the WRS allows multiplexing high precision timing and packet data. The WRS is equipped with Class of Service[3] support. Currently CoS is mapped into Quality of Service[4]. Queue Scheduling allows multiple packet flows to share the link capacity according to the QoS policy. Technologies like "Overprovisioning"[5] and "Best Effort Delivery"[6] for the Queue Scheduling and "Tail-Drop" for the Queue Management[7] are used. Figure 2 displays a rough scheme of a switch. Though being fully compliant with the IEEE 802.1 standard[8] those strategies may display poor performance within scenarios of heavy traffic. Such scenarios are likely to occur during accelerator operation. As a consequence delays in transmission of packets would be expected. When queues are full packet loss may occur.

Being a full-duplex, non-blocking and manageable Gigabit Ethernet switch the WRS supports both fibre and copper connections. Further

2 `http://www.ohwr.org/companies`

3 Class of Service – **C**lass **o**f **S**ervice (CoS), Classification of Gigabyte Ethernet traffic by tagging the packets with different priorities.

4 Quality of Service – **Q**uality **o**f **S**ervice (QoS), Provisioning of resources in a network to provide a minimum level of service for every CoS specified.

5 An alternative to complex QoS control mechanisms is to provide high quality communication by generously over-provisioning a network so that capacity is based on peak traffic load estimates.

6 Best effort delivery describes a network service in which the network does not provide any guarantees that data is delivered or that a user is given a guaranteed quality of service level or a certain priority.

7 Queue Management is a technique to decide when to drop a packet if the queue of a switch is full.

8 `http://standards.ieee.org/getieee802/download/802.1X-2004.pdf`, `http://de.wikipedia.org/wiki/IEEE_802.1X`

a set of enhanced features enables the WRS to be compliant to the requirements of high performance control and timing systems:

- Sub-nanosecond synchronization engine (IEEE 1588-compliant)
- Eight priority levels with configurable routing mode (store/forward, cut-through)
- Full routing latency determinism for highest priority packets
- Improved network redundancy protocols by using a hardware-assisted Spanning Tree Protocol that allows microsecond-range recovery of a broken link.

The WRS provides the following communication interfaces: *WRS interfaces*

- Two uplink ports (WR slaves) to receive the time and frequency reference from the upper layer of the network. It is possible to designate one of the uplink ports as primary PTP slave while the other one serves as a backup timing receiver.
- Eight downlink ports (WR masters) to propagate the timing to other switches or nodes.
- External 10 MHz, 125 MHz and PPS inputs to enable the WRS to work as a WR/PTP grandmaster that takes the reference timing from an atomic clock or a **G**lobal **P**ositioning **S**ystem (GPS) receiver. For measurement and testing purposes the 125 MHz input can be used as a direct clock input.
- **P**ulse **P**er **S**econd (PPS) and 125 MHz outputs to provide a slave'in-phase recovered clock.
- 100BaseT Ethernet port to be used for device management and administration using an external, non-WR network.
- Multi-purpose **R**ecommended **S**tandard 232 (RS232) port that can receive or output the **U**niversal **C**oordinated **T**ime (UTC) time-of-day information or that can be used as a management serial console.

White Rabbit Switches may appear both as master and as slave. For an example of a timing network using the WRS see Figure 3.

Switches and nodes may be added dynamically. Other Gigabit Ethernet switches and non-WR nodes can be connected to the network as well.

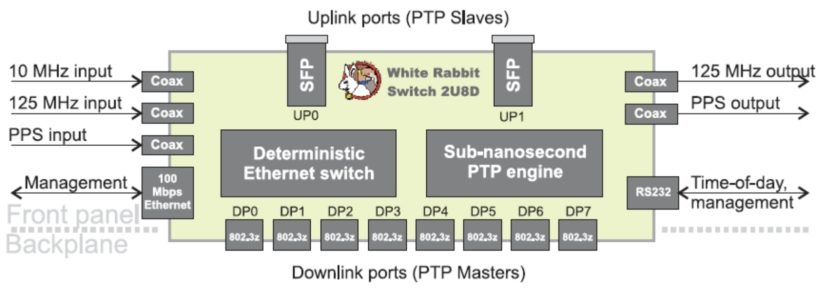


Figure 2: Scheme of the White Rabbit Switch

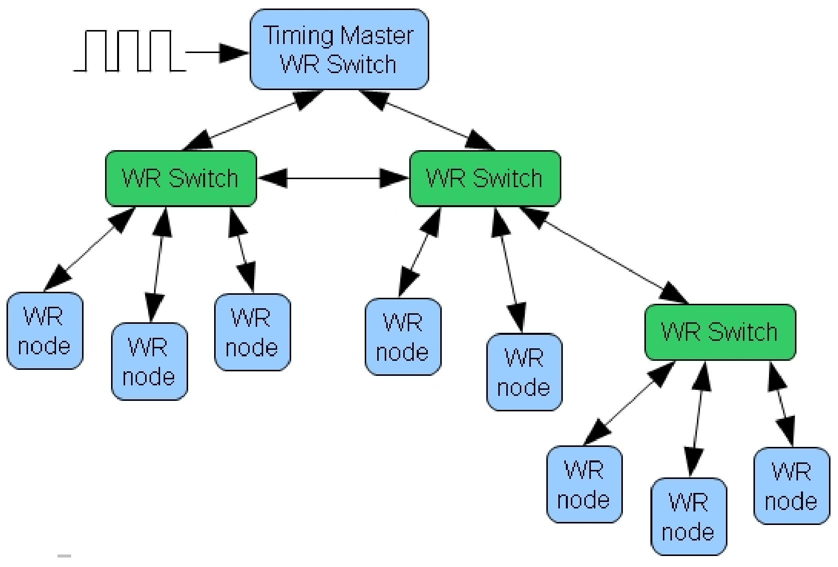


Figure 3: Scheme of a White Rabbit Timing network

3 THE PLAN

3.1 FOCUS AND GOAL

The aim of this master thesis is to improve the algorithms of queue scheduling and queue management in the White Rabbit Switch. Possible alternatives for queue scheduling will be discussed, simulated and evaluated (see Figure 4). GSI and CERN requirements like

Requirements

- **Determinism**: transfer of real-time messages with a guaranteed maximum latency
- **Robustness**: extremely low packet loss

must be considered.
Packets with highest priority (7) have to be treated appropriately and must be transmitted as fast as possible with an upper bound latency of 13 μ s through the switches under certain conditions. Packets with lower priorities (0..6) may be treated less critical, e.g. with an upper bound latency of 100 ms for priority 4.
The focus of this work will be the simulation of the current scheduling approach in comparison to alternative ways of implementation. Advantages and disadvantages of the scheduling methods used will be discussed. A simulation of possible Queue Scheduling strategies will help to visualize and measure the possible improvements. The final implementation needs to be realized in hardware inside the White Rabbit Switch using **V**ery **H**igh Speed Integrated Circuit Hardware **D**escription **L**anguage (VHDL) and will not be topic of this work.

Simulation

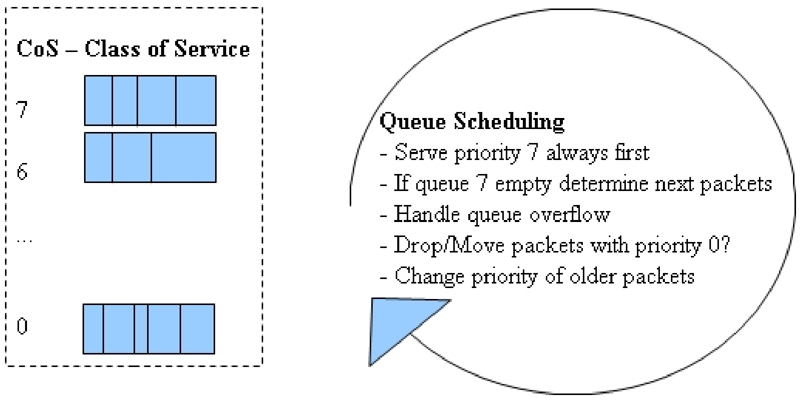

CoS – Class of Service
7
6
...
0
Queue Scheduling
- Serve priority 7 always first
- If queue 7 empty determine next packets
- Handle queue overflow
- Drop/Move packets with priority 0?
- Change priority of older packets

Figure 4: Queue Scheduling Management

4 SCHEDULING

4.1 SCHEDULING

Generally scheduling describes the assignment of a rare resource to a group of candidates. A typical computer science example for a situation that uses scheduling is the division of the computers processor between processes. The scheduler is a vital part of the operating system.
Another example is found in database handling: scheduling defines how many parallel transactions may occur without destroying the databases consistency. The term task is used as a placeholder in the following. Instead process or packet can be inserted, depending on the context.

4.2 SCHEDULING ALGORITHMS

Scheduling requirements

During the years many different strategies were developed to realize different scheduling approaches meeting varying demands. Such demands are

- Throughput – high number of successful tasks per time unit.
- Latency, specifically:
  - Turnaround – total time between submission of a task and its completion.
  - Response time – amount of time between submission of a request and the production of the first response.
- Fairness / Waiting Time – No starvation, every candidate must have its turn.

4.2.1 *Quality Features*

Quality features

To be able to benchmark scheduling algorithms several quality features are defined. Such features allow to compare different scheduling approaches. Typical quality features of scheduling algorithms are

- Maximal efficiency
- Minimal turnaround time
- Minimal response time

- Maximal throughput
- Fairness.

A scheduling algorithm is never able to suit all the needs mentioned above. An optimal solution is not available. The preference of one feature leads to neglect of another.

4.2.2 *Priority Scheduling*

A priority scheduler handles tasks with the highest priority first. Lower priority tasks follow according to their rank. Lower priority tasks may be interrupted by higher priority tasks. This method has the advantage of short waiting and response times for high priority tasks. Deadlines can be easily met by assigning higher priorities. A disadvantage of this procedure is the occurrence of starvation for low priority tasks if fixed priorities are used.

4.2.3 *Combination of Scheduling Strategies*

Concurrent use of different strategies tries to combine the advantages by simultaneous elimination of the disadvantages.

4.2.4 *Feedback Scheduling*

Feedback Scheduling algorithms consider the past of a task during the choice of the next task. This may happen by assigning higher priorities to longer waiting tasks ('ageing'). During waiting the priority of a task gets higher until the task is accomplished. This method has the disadvantage that its task management takes more time. Each time the whole list of tasks needs to be considered. This leads to a considerable performance overhead.

4.3 SCHEDULING IN THE WHITE RABBIT SWITCH

This thesis will cover the scheduling algorithms that are interesting for improving the scheduling within the WRS, such as priority scheduling algorithms or the combination of different scheduling approaches.
Depending on the complexity of the scheduling algorithm computing overhead will occur. The goal is to minimize computing overhead while providing a deterministic service time for high priority packages.
. . .

4.3.1 *The current (Scheduling) Strategy in the White Rabbit Switch*

The current strategy within the WRS is relatively simple. Packets with the priority 7 will be sent first. Afterwards packets with lower priorities are delivered in descending order.

Algorithm in pseudo-code

In pseudo-code the algorithm looks like this:

```
for $i = 7 \to 0$ do
  if $size(queueout) \neq 0$ then
    send frame from the queue of priority i
  end if
  $i \leftarrow i - 1$
end for
```

TODO advantages / disadvantages ...

5 SIMULATION

5.1 SIMULATION

The imitation of the operation of a real-world process or system over time is called a simulation. A simulation represents certain key characteristics or behaviours of a selected physical or abstract system. By creating a model of the real-world process it is possible to mimic an existing system.

Simulation model

A simulation model is developed to study the behaviour of a system as it evolves over time. The model is built from a set of assumptions that describe the operation of the system in the real world. The objects of interest within the model, also called entities, have relationships that are expressed in mathematical, logical or symbolic form.

Entities

Advantages

The usage of a simulation has certain advantages over a real-life test [2]:

- Low-cost and relatively fast set-up of the simulation environment.
- Easy exchange of variables, the data-set used or parts of the included algorithms.
- Changing the input data and observing the simulation output allows to find out which variables are the most important.
- Quick investigation and analysis of different outputs and different performances of the system.
- Simple exploration of new procedures and information flows within the simulation model without disturbance of the real-world system.
- New hardware designs may be tested without a real setup.
- Profound prediction of the results in the corresponding real-world system.
- Verification of analytic solutions.
- Visualization and overview of a complex process.

Disadvantages

Simulations also have disadvantages:

- Building a simulation model requires profound knowledge of the real-world system.

- Difficulty of interpretation of the simulation results, random input may result in random output.
- Creating a simulation model may require time and may be expensive.

A simulation model can be used both as an analysis tool to predict the performance of real-world systems under varying sets of circumstances. The effects of changes and their impact on the system performance can be closely studied. During the simulation data may be collected as if a corresponding real-world system were observed. Analysis of the collected data allows to estimate the performance of the real-world system.
Not all real-world systems are appropriate for being simulated. Simulation should not be used:

- Whenever common sense or analytical skills can be used to solve a problem.
- When it is easily possible to perform direct experiments.
- When the costs to create a simulation exceed the savings.
- When resources and time are not available to create a simulation.
- When no data is available to build a simulation model.
- When no manpower is available to validate and verify the simulation model.

The field of network traffic simulation is a good example of the usefulness of computers to create simulations.

5.2 SIMULATION MODELS

The term "model" may be classified

- mathematical or
- physical.

A simulation model is considered as a particular type of mathematical model of a system[2].

Classification

Simulation models are further classified static or dynamic, deterministic or stochastic, discrete or continuous.

Classification	Short Explanation
static	a particular point of time
dynamic	changes over time
deterministic	no random variables
stochastic	random variables as inputs
discrete	state variable(s) change at a discrete set of points in time
continuous	state variable(s) change continuously over time

Table 1: Overview of Simulation Model Types

5.3 DISCRETE-EVENT SYSTEM SIMULATION

Modelling a system in which the state variable changes at a discrete set of points in time is called a discrete-event simulation. Most of the time numerical analysis methods are used instead of analytical methods. *Numerical* methods are used to computationally "solve" mathematical models whereas *analytical* methods make use of mathematical deduction.

Running a simulation

Simulation models are rather "run" than solved. By running a simulation an artificial system history is generated from the model assumptions. During the run observations are collected. Analysing the collected data allows to estimate true corresponding real-world system performance.

5.4 SIMULATION OF THE WHITE RABBIT SWITCH

Advantages

To be able to easily test and evaluate different scheduling algorithms in the WRS a discrete-event simulation is the method of choice. The usage of a software simulation enables to mimic the WRS without real hardware. Some advantages are

- No development hardware is required.
- Independence of hardware-specific programming language VHDL, the simulation can be developed in any programming language.
- Easy exchange of scheduling algorithms.

Exchange of scheduling algorithms

In this thesis the traffic within the WRS will be further investigated and improved. Different scheduling algorithms will be hooked into the simulation setup to measure the delivery time of a packet and analyse the results. No specialized hardware is required except for a standard **P**ersonal **C**omputer (PC) with a standard operating system such as Linux[1]. Simulation software is available under an academic

1 http://en.wikipedia.org/wiki/Linux

license.
Using a simulation framework instead of a self-written solution has the advantage of being able to use a large set of classes including time measurement possibilities, standard components such as sources, messages, queues or routers. That way a simulation model can be created using a tested and well established software framework. Comprehensive documentation is available. An accompanying newsgroup offers the possibility to discuss open questions in a larger community. Active developers maintain the simulation framework distribution by further enhancing the functionality as well as fixing bugs.
For the simulation of scheduling algorithms within the WRS the open-source software framework **O**bjective **M**odular **N**etwork **T**estbed in C++ (OMNeT)++[2] is chosen. The following chapter describes several details of this simulation environment.

OMNet++

2 http://www.omnetpp.org

6 OMNET++

6.1 OMNET++

OMNeT++ is a modular framework with the focus on simulation of networks, including wired and wireless communication networks, on-chip networks, queueing networks etc. A more detailed overview is found in [5] and in the internet.

Discrete event simulation

OMNeT++ allows to simulate discrete events. This enables the use of OMNeT++ to simulate parallel and distributed systems as well as hardware architecture. The latter is the reason to choose OMNeT++ as a base for this thesis. OMNeT++ is available freely under an academic license[1].

C++

OMNeT++ itself is realized with C++ and provides several C++ classes for the simulation of network components. Those classes can be combined into a model of the switch that allows to exchange packet scheduling algorithms for testing purposes. OMNeT++ is portable and runs perfectly well on Linux as well as on Windows systems. The simulation itself can be run in a graphical mode which is apparently slower but allows to visualize the packet flow within the model. During simulation the objects can be inspected directly, e.g. how many packets were handled, how much objects a queue holds, etc. .

Portability

6.2 MODEL STRUCTURE

OMNeT++ models consist of so called simple modules that use message passing. The simple modules are written in C++ using the simulation class library. Compound modules can be built by grouping simple modules. That way modules can be easily reused. Communication between modules is realized by sending messages. Those messages may contain usual attributes like a timestamp but also arbitrary data. Typically messages are sent via gates, but sending them directly to their destination modules is also possible. Gates represent the input and output interfaces of the modules. Connections are the link between input and output gates of different modules.
Usually messages travel through a chain of connections in a model. The messages start in a source and end in a simple module called sink. During their way they pass the simple modules combined into a module.

1 http://www.omnetpp.org/home/license

6.3 NED

NED is a OMNeT++-specific declarative high-level language. In NED files the structure of a model, such as the modules and their interconnection, is defined. A NED file consists typically of simple module declarations, compound module definitions and network definitions. The interface of a simple module is described by designing gates and parameters. Compound module definitions not only contain the declaration of their external interface but also the definition of submodules and their interconnection.

6.4 GRAPHICAL EDITOR

OMNeT++ is integrated into Eclipse as a plug-in. A powerful graphical editor is included to easily design a simulation out of the available or own components. The editor uses NED as its native file format. NED files may be edited graphically or in the source view. Changes in each mode are directly reflected in the other view.

6.5 SIMPLE MODULE PROGRAMMING MODEL

The active elements in a model are the simple modules. They are programmed in C++ with the OMNeT++ simulation class library. A simple module can be implemented by subclassing the `cSimpleModule` class. Functionality may be added either by a coroutine-based programming model or a event-processed function. Coroutine-based programming implies that the modules code runs in its own (non-pre-emptively scheduled) thread. This thread is controlled from the simulation kernel every time the module receives an event (message). Using the event-processed function model implies that the simulation-kernel calls the given function of the module with the message as argument. An important difference of the two models is that coroutine-based programming needs its own CPU stack and therefore more memory is required for the simulation. A module overrides an initialization and a finish method which are called when the simulation starts and successfully ends.

Coroutine-based programming

Event-processed function model

The most frequent tasks in simple modules are sending and receiving messages via gates. Messages are sent either directly or via an output gate. When using coroutine-based programming messages are received via one of the several variations of the receive call. When using event-processing function mode messages are delivered after an invocation from the simulation kernel. The content of a message may be user-defined in a MSG file. This enables OMNeT++ to create the necessary C++ classes that allow to investigate the messages at run-time.

Messages

An OMNeT++-simulation may be modified dynamically during the execution of a simulation: modules can be created and deleted and connections can be rearranged.

6.6 COMPOUND MODULES

Simple modules may be grouped into compound modules. A compound module has no underlying C++-class because no active behaviour is associated with it. A compound module has gates and parameters. The behaviour is realized by the simple modules within the compound module.

6.7 THE SIMULATION LIBRARY

OMNeT++ comes with a comprehensive object library. The reflection functionality enables not only high-level debugging and tracing but also automated animation. The OMNeT++ framework also tracks the object ownership and detects bugs caused by aliased pointers and misuse of shared pointers to help avoiding memory leaks and pointer aliasing.

Simulation timing

The simulation timing is realized using a 64-bit integer-based fixed-point representation. This avoids precision problems which would arise if a double type was used for the timing. The simulation library contains classes to create random numbers from various distributions, queues and several container classes. Messages can be created. Routing traffic can be simulated and algorithms may be applied. Statistics are possible with a set of statistical classes. Simulation output can be written into a file and post-processed.

6.8 REAL-TIME SIMULATION

The event scheduler in the OMNeT++ simulation framework is pluggable. It is possible to define a real-time scheduler to realize real-time simulation or hardware-in-the-loop like functionality.

6.9 ANIMATION AND TRACING

Tkenv

To visualize and animate simulations OMNeT++ uses the portable graphical windowing user interface Tkenv. Tkenv allows OMNeT++ simulations to be interactively executed. Tracing and debugging is also possible. Tkenv offers three modes of operation:

- Automatic animation
- Module output windows
- Object inspectors.

Automatic animation allows OMNeT++ to animate the flow of messages across the simulated object and reflect state changes of the nodes in the display. **Module output windows** are capable to display debug output. They display the textual debug and tracing information a simple module may write. **Object inspectors** allow to directly investigate objects in the simulation model. They display the state or contents of an object in an appropriate way depending on the data. Manual modification of the objects data is also possible. For batched simulation runs the graphical output may be turned off completely, OMNeT++ can be run on the command-line. This allows a quick runs ("Express runs") of the simulation.

6.10 VISUALIZING DYNAMIC BEHAVIOUR

Logging

The interactions between modules are logged to a file by OMNeT++. During or after a simulation run this log file may be processed or used to draw interaction diagrams. To visualize the chronology of events OMNeT++ provides a sequence chart diagramming tool. It is possible to select the modules to be followed. Causes of events and their consequences can be displayed on a non-linear time axis and analyzed. A filter for modules and events can be applied.

6.11 ORGANISING AND PERFORMING EXPERIMENTS

Simulation runs can be organized in OMNeT++ around different concepts:

- Model, the executable file and NED configuration files.
- Study, consists of a number of experiments from which conclusions may be drawn.
- Experiment, investigation of a parameter space on a model.
- Measurement, numerous simulation runs on the same model with the same parameters but varying seeds.
- Replication, a repetition of a measurement.
- Run, performance of a simulation, characterized by exact time, date and host.

Experiments may be grouped to simulation batches. The measurements to be performed can be further defined. The batch progress may be monitored in OMNeT++'s **I**ntegrated **D**evelopment **E**nvironment (IDE). On multiple processors or multicores parallel simulation runs are possible which guarantees a linear speedup.

Parallel simulation

6.12 result analysis

OMNeT++ allows to automate result analysis by applying rules. The interesting data may be selected into datasets by patterns. Various processing, filtering and charting steps may be added to datasets. Content changes in the underlying data files are directly reflected in the datasets and the charts. This allows to automatically update datasets and charts after re-runs of simulations. It is possible to build a hierarchy of datasets by having datasets as input for other datasets.

7 REALIZING THE SIMULATION

7.1 PREREQUISITES

...

7.2 THE ENVIRONMENT

...

7.3 THE IMPLEMENTATION

...

8 TIME MEASUREMENT

8.1 time measurement

. . .

8.2 how to

. . .

8.3 results

. . .

9 COMPARISON

9.1 COMPARISON

. . .

9.2 THE SCHEDULING ALGORITHMS

. . .

9.2.1 *Original Algorithm*

. . .

9.2.2 *Priority Scheduling Algorithm*

. . .

9.2.3 *Feedback Scheduling Algorithm*

. . .

9.3 ADVANTAGES AND DISADVANTAGES

. . .

10 CONCLUSION

10.1 CONCLUSION

. . .

10.2 THE OUTCOME

. . .

11 SUMMARY

11.1 SUMMARY

. . .

BIBLIOGRAPHY

[1] *Alice's Adventures in Wonderland*. 1865.

[2] *Discrete-Event System Simulation*. Pearson Prentice Hall, 2005.

[3] The White Rabbit Project. 2009.

[4] White Rabbit: Sub-Nanosecond Timing Distribution over Ethernet. 2009.

[5] Rudolf Hornig András Varga. An Overview of the Omnet++ Simulation Environment. 2007.

[6] Tomasz W. Precise time and frequency transfer in a white rabbit network. Master's thesis, Warsaw University of Technology, 2011.