FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Simulation-based evaluation of a fault-tolerant IP communication system

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Technische Informatik

eingereicht von

Helene Sophie Oberhumer
Matrikelnummer 0626627

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.-Prof. Dr. Wolfgang Kastner
Mitwirkung: Univ.-Ass. Dipl.-Ing. Lukas Krammer

Wien, 17.04.2014

(Unterschrift Verfasserin) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Simulation-based evaluation of a fault-tolerant IP communication system

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Computer Engineering

by

Helene Sophie Oberhumer
Registration Number 0626627

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.-Prof. Dr. Wolfgang Kastner
Assistance: Univ.-Ass. Dipl.-Ing. Lukas Krammer

Vienna, 17.04.2014

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Helene Sophie Oberhumer
Schönbrunner Straße 192/1/10, 1120 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasserin)

Danksagung

Ich bedanke mich vielmals bei Wolfgang Kastner und Lukas Krammer für die Betreuung der Diplomarbeit und die Möglichkeit, meine Arbeit am Institut für Rechnergestützte Automation zu erstellen.

Besonderer Dank gilt meiner Familie, meinen Freunden und Studienkollegen für die Förderung und Unterstützung während meiner Studienzeit. Meiner Schwester Astrid danke ich sehr herzlich für das Korrekturlesen. Für die Ermöglichung des Studiums und vor allem die vorhergehende Ausbildung möchte ich mich bei meiner Mama Edith bedanken.

Abstract

Automation finds an ever increasing scope of application. Consequently, automation systems and the communication systems deployed within them have to meet new requirements. In safety-critical environments, reliable communication is essential in order to prevent situations with severe consequences, like life-threatening scenarios or costly damages. To increase the reliability of communication systems, the concept of fault tolerance can be applied, which enables the system to continue its operation even in the presence of a fault. This is basically achieved by hardware redundancy like redundant links in the context of data communication. Networks with redundant paths entail the necessity of a network protection and recovery protocol. Due to the increasing importance of the Internet, network technologies such as Ethernet and IP find their way even into the automation domain. However, different restrictions and additional demands, such as limited network bandwidth and special real-time requirements, have to be considered in this context. Although many different fault-tolerant communication technologies are available, they are mainly intended for use in high-performance communication systems with a limited number of network nodes.

The present work provides an overview of the state of the art of such protocols. Subsequently, an important representative, which is based on a ring topology, is investigated. Since such a system is too complex for a formal analysis and the scalability cannot be examined in a real scenario, a simulation-based approach is used. The communication system under investigation operates at Data Link Layer and is based on Ethernet, the data exchange is based on IP. It offers further Quality of Service mechanisms to categorize the data traffic. The simulation studies are based on networks which consist of links with a bandwidth of less than $1 Mbps$, operating in half duplex mode. The Media Redundancy Protocol (MRP), a protocol established in the field of industrial automation, is selected and examined under the restrictive conditions stated above. The simulation studies are performed with the discrete-event simulation environment OMNeT++ in combination with the INET framework, which is extended by an MRP model especially programmed for the purpose of this study. The investigations explore the scalability of MRP with various parameter values and the spatial and temporal distribution of the network utilization. Additionally, experiments show how different implementations of ARP influence the network performance. Beyond the results of the simulation, the analysis of the MRP discovers possible improvements for low-speed Ethernet connections.

Kurzfassung

Mit dem wachsenden Anwendungsgebiet der Automation verändern sich auch die Anforderungen, die an ein Automatisierungssystem und dessen Kommunikationssystem gestellt werden. Vor allem bei sicherheitskritischen Systemen ist eine zuverlässige Kommunikation unerlässlich, da ein Fehlverhalten des Systems schwerwiegende Folgen haben kann, wie etwa lebensbedrohliche Situationen oder immense Kosten. Die Zuverlässigkeit von Kommunikationssystemen kann durch das Anwenden von Fehlertoleranz erhöht werden. Diese stellt die Funktionsfähigkeit eines Systems auch bei Auftreten einer Fehlerursache sicher und wird durch eine redundante Ausführung von Komponenten realisiert. Im Bereich der Datenkommunikation wird diese grundsätzlich durch Hardware-Redundanz in Form von redundanten Verbindungen umgesetzt. Auf Grund redundanter Pfade in einem solchen Netzwerk, muss ein sogenanntes *network protection and recovery* Protokoll eingesetzt werden. Mit der Bedeutungszunahme des Internets halten Netzwerktechnologien wie Ethernet oder IP auch in die Automation Einzug. Allerdings müssen in diesem Bereich auch verschiedene Einschränkungen und zusätzliche Anforderungen wie eine geringe Bandbreite oder spezielle zeitliche Bedingungen berücksichtigt werden. Viele der verfügbaren fehlertoleranten Kommunikationssysteme sind für den Einsatz in Netzwerken mit sehr hohen Bandbreiten und wenigen Netzwerkteilnehmern konzipiert. Die vorliegende Arbeit bietet einen Überblick über die aktuellesten Protokolle. Basierend darauf wird ein wichtiges Protokoll, das auf einer Ringtopologie basiert, detailliert untersucht. Nachdem dieses System für eine formale Analyse zu komplex ist und keine Möglichkeit besteht, seine Skalierbarkeit in einer realen Umgebung zu untersuchen, wird eine Simulationsstudie durchgeführt. Das betrachtete System basiert auf Ethernet, wobei die Datenkommunikation über IP gehandhabt wird. Darüber hinaus bietet das System mehrere Serviceklassen, die eine Priorisierung der Daten ermöglichen. Als *protection and recovery* Protokoll wird das Media Redundancy Protocol (MRP), ein in der Industrieautomation etabliertes Protokoll, ausgewählt. Das Verhalten dieses Kommunikationssystems wird in dieser Arbeit unter speziellen Einschränkungen, die sich aus halb-duplex Verbindungen mit einer Bandbreite von kleiner als $1 Mbps$ ergeben, analysiert. Für die Simulationsstudie wird die diskrete, ereignisorientierte Simulationsumgebung OMNeT++ und das dazugehörige Framework INET verwendet. Dabei wird das Kommunikationssystem — und im Zuge dessen auch der Standard MRP — im INET Framework implementiert. Im Anschluß wird durch verschiedene Experimente die Skalierbarkeit des Systems, sowie die zeitliche und räumliche Verteilung der Netzwerkauslastung in Ringtopologien untersucht. Zusätzlich wird eruiert, wie sich verschiedene Modelle von ARP auf die Netzwerkleistung auswirken. Neben den Ergebnissen aus den Simulationsstudien liefert diese Arbeit mögliche Verbesserungen des MRPs für den Einsatz in Ethernet basierten Netzwerken mit sehr langsamen Verbindungen.

Contents

Abbreviations

ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
ATM-TN	ATM Traffic and Network simulator
BID	Bridge Identification Number
BPDU	Bridge Protocol Data Unit
CARP	Common Address Redundancy Protocol
CFI	Canonical Format Indicator
CN	communication node
CSMA	Carrier Sense, Multiple Access
DEI	Drop Eligible Indicator
DLL	Data Link Layer
DLPDU	Data Link Layer Protocol Data Unit
EAPS	Ethernet Automatic Protection Switching
EPSR	Ethernet Protection Switching Ring
FCS	Frame Check Sequence
FDB	Filtering Database
FEL	Future Event List
FES	Future Event Set
FIFO	First In First Out
GD	General Device
GloMoSim	Global Mobile System Simulator
GTNetS	Georgia Tech Network Simulator
GWD	Gateway Device
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
ID	Identification Number
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IPX	Internetwork Packet eXchange
ISO	International Organization for Standardization
LAN	Local Area Network
LIFO	Last In First Out
LNM	Line Network Manager

LOS	Loss Of Signal
LRU	Least Recently Used
MAC	Media Access Control
MaRS	Maryland Routing Simulator
MCU	Microcontroller Unit
MGMT	Management
MPI	Message Passing Interface
MRC	Media Redundancy Client
MRM	Media Redundancy Manager
MRP	Media Redundancy Protocol
MSTP	Multiple Spanning Tree Protocol
MTU	Maximum Transmission Unit
NCMT	Network Control Message Type
NED	Network Description
NMIB	Network Management Information Base
NWL	Network Layer
OPNET	Optimised Network Engineering tool
OSI	Open Systems Interconnection
PCP	Priority Code Point
pdns	Parallel/Distributed ns
QoS	Quality of Service
QUIPS	Queen's University IP simulation
REP	Resilient Ethernet Protocol
RFC	Request For Comments
RNM	Ring Network Manager
RNMP	Primary Ring Network Manager
RNMS	Secondary Ring Network Manager
RR	Round-Robin
RRP	Ring-based Redundancy Protocol
RRPP	Rapid Ring Protection Protocol
RRPP	Rapid Ring Protection Protocol
RRPPDU	Rapid Ring Protection Protocol Data Unit
RSTP	Rapid Spanning Tree Protocol
SRPT	Sub Ring Packet Tunnel in Major Ring
STCN	Spanning Tree Topology Changes Notification
STP	Spanning Tree Protocol
TCI	Tag Control Information
Tcl	Tool command language
TCN	Topology Change Notification
TCP	Transmission Control Protocol
TOS	Type of Service
TPID	Tag Protocol Identifier
UDP	User Datagram Protocol

UID	Unique Identifier
ULC	upper-layer reachability confirmation
VID	VLAN Identifier
VLAN	Virtual Local Area Network
VMPS	VLAN Membership Policy Server
WIPSIM	Wireless IP Simulator
WRR	Weighted Round-Robin
XML	Extensible Markup Language

CHAPTER 1

Introduction

1.1 Motivation and problem statement

Communication technologies are key elements of automation systems and are present on every level of the well-known automation pyramid depicted in Figure 1.1. On the one hand, sensors and actuators, which act as interfaces to the real, physical world, are connected to the controllers at the field level to exchange process data. On the other hand, multiple controllers share information at the automation level. At the third tier, the management level, all system data is gathered and used for monitoring, optimization as well as visualization tasks. The communication systems of these levels have to fulfill different requirements. Accordingly, the amount of data managed by a single function (or device implementing the function) increases from the field level over the automation level toward the management level. In contrast, the number of devices implementing the function of the corresponding level decreases. Generally speaking, at higher tiers, higher bandwidths are needed, but the requirements on timeliness are lower [1].

Nowadays, the application field of automation becomes broader and hence the requirements on automation systems and their communication systems change. Especially when automation finds its way into safety-critical environments, communication systems have to become more reliable in order to prevent scenarios with severe consequences, like life-threatening situations or costly damages. To increase the reliability of communication systems and accomplish safety of mission critical systems, fault tolerance is deployed. Fault-tolerant systems perform their function according to their specification in spite of faults. Fault-tolerance can be achieved by hardware redundancy, i.e. additional components are integrated which are not needed in the absence of faults. In communication systems, this redundancy is accomplished by redundant links.

Due to the increasing importance of the Internet, network technologies such as Ethernet and the Internet Protocol (IP) are even applied in the automation domain. Besides that, there is a trend toward IP-based approaches for communication on the automation level and even on the field level, as the intelligence of components on the field level increases. Often, networks are divided into segments when the buildings, where the automation system is installed, reach

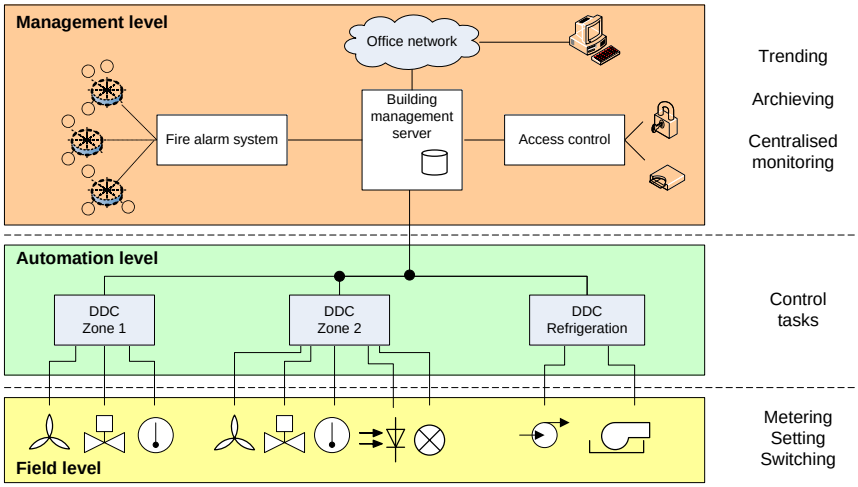


Figure 1.1: The automation pyramid [2].

considerable spatial dimensions. Therefore, an appropriate support by a protocol such as IP is needed. In addition, multicast capable protocols contribute to the efficiency of the communication system [1]. Using Ethernet, a deterministic Carrier Sense, Multiple Access (CSMA) based variant for media access control which supports frame prioritization, allows to efficiently use the "raw" throughput capacity and to fulfill time constraints [1]. In contrast, networks with redundant paths based on Ethernet require special network protection and recovery protocols to ensure a loop-free topology.

However, automation networks often need to be constructed under restrictive conditions fulfilling special real-time requirements. Such limiting circumstances can be network links which only offer low bandwidths or large networks spanning over long distances. Most of the available fault-tolerant communication technologies are designed for the application in high-performance networks though. Besides, these systems usually restrict the number of communication nodes. It is unknown, how these systems operate in networks with a much smaller performance compared to the specified high-performance and how they behave with an increasing network size. There is no knowledge about the scalability of such systems and consequently the timeliness. In addition, such systems are often too complex for formal analysis and the performing of real-world experiments deploying them too costly.

1.2 Aim of this thesis

The main goal of this thesis is the investigation of one representative of the available fault-tolerant communication technologies with regard to bandwidth restrictions and scalability. To achieve this, an initial aim is to get an overview about existing mechanisms for providing reliable communication. Another objective is the successful implementation of a selected mechanism in an open simulation framework. Therefore, an appropriate simulation environment shall be selected. The implementation shall result in a practical and complete toolchain, which allows to

easily set up different experiments via command line interface and to automatically produce statistical analysis. The latter process shall be supported by a suitable program for statistical computing. Then, multiple milestones to achieve result from various experiments with the simulation model, which are all aiming at the investigation of the network performance and the scalability of the communication system. These investigations shall examine the feasibility of a fault-tolerant communication system under the aforementioned requirements. The gained results shall be visualized for a better understanding after a thorough analysis. The final goal of this study is to provide a basis for the calculation of the network convergence time and suggestions how the investigated protocol can be improved.

1.3 Methodology

In the first step, a literature study about simulation in general was conducted. Thus, knowledge about the methodology of a simulation study was gained to improve the quality of the model implementation. Additionally, different simulation tools were discussed and an appropriate tool was chosen. Among the considered simulators were OMNeT++, ns-2, OPNET, GloMoSim and NetSim. Moreover, an understanding of the fundamental functioning of the simulation environment was obtained which helped during the implemention process. Then, state-of-the-art network protection and recovery protocols were examined. Finishing this process, one protocol was selected for the communication system. Next, the conceptualized model of the communication system was implemented in the simulation environment. In the experimental phase, various alternatives were designed and simulated. Concluding from the simulation results, the communication system was analyzed and suggestions for improvements were given.

1.4 Structure of this thesis

The present work is structured as follows. The next chapter provides theoretical aspects of simulation, involving terminology, advantages and disadvantages as well as a process flow model for carrying out simulation studies. Moreover, a very technical insight is given into the fundamental concept of discrete-event simulation and modern simulation tools are listed. Subsequently, the concept of Virtual Local Area Networks (VLANs) is described, as they are applied in various protection and recovery protocols. The second chapter is closed with a detailed overview of the state of the art of such protocols. Chapter 3 deals with the model conceptualization of the communication system and its implementation using the INET framework. Furthermore, two enhancements of the framework are proposed in this section. Chapter 4 includes a description of the conducted experiments and a thorough analysis. In Chapter 5, the thesis offers a discussion of lessions learned and draws a final conclusion.

CHAPTER 2

State of the Art

This chapter deals with the state-of-the-art of simulation and network technologies. More precisely, the first part of this chapter provides a detailed insight into the area of simulation. The second section outlines the concept of VLANs, which are used in the proposed approaches for building resilient networks. Subsequently, candidate technologies that can be used for resilient communication are discussed in a further section.

2.1 Simulation

The following section introduces the terminology used in the field of simulation. Additionally, purposes for carrying out simulation studies and their benefits and drawbacks are explained. Furthermore, possible classification schemes are described. Then, a more technical insight is provided into the functioning of discrete-event simulation environments. The last section deals with network simulation tools.

2.1.1 Terms & Definitions

A *simulation* is defined by [3] as

> *"the imitation of the operation of a real-world process or system over time. Whether done by hand or on a computer, simulation involves the generation of an artificial history of a system and the observation of that artificial history to draw inferences concerning the operating characteristics of the real system."*

In [4] it is stated that simulation is a very general term as it has many fields of applications and is seen as

> *"a broad collection of methods and applications to mimic the behavior of real systems, usually on a computer with appropriate software."*

or as

> *"the process of designing and creating a computerized model of a real or proposed system for the purpose of conducting numerical experiments to give us a better understanding of the behavior of that system for a given set of conditions."*

and refers to *computer simulation* when the imitation is done by software.
A more general definition is provided by [5], which describes a simulation in the context of computable systems as

> *"the execution of calculations using a model where input data is transformed into output data."*

In the area of computer simulations, this transformation is done by a *simulation system* or *simulator* using an appropriate algorithm, which stores or outputs the results [5].
Summarizing these definitions, a simulation is an analysis method which is used to study the behavior of a system. [6] classifies simulation among other possibilities for studying real systems, like experiments or analytical approaches, according to Figure 2.1.

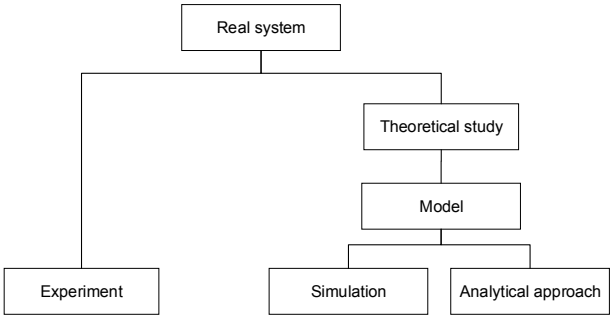


Figure 2.1: Possibilities to study the behavior of a real system [6].

[5] cites that, a *system* is a set of the system's elements — its subsystems — which cooperate in a requested manner. It can be a physical object or a hypothetical construct and is characterized by its environment, structure and behavior. Since the elements of a system are again subsystems, it depends on the intention of the investigator at which abstraction level the system is observed. According to [4], a system is a facility or process which is either actual or planned, for example:

- "A computer network with servers, clients, disk drives, tape drives, printers, networking capabilities, and operators."
- "A freeway system of road segments, interchanges, controls, and traffic."
- "A bank with different kinds of customers, servers, and facilities like teller windows, automated teller machines, loan desks, and safety deposit boxes."
- "A distribution network of plants, warehouses, and transportation links."

[5] states that the *model*, as a imitation of the real system, has to be sufficiently similar to the system in order to satisfy its purpose of creation. A simulation model is built with regard to a set of assumptions concerning the operation of the system [3]. These assumptions are mathematical, logical or symbolic expressions of the relationships between the objects of interest of the system, also called *entities*. However, the collection of entities considered for investigation can vary by the purpose of the simulation study. Numerical, *computer-based simulation* can be used to imitate the system's behavior over time, when real-world systems are too complex for being solved mathematically (e.g., by differential calculus, probability theory or algebraic methods) [3], or the mathematical solutions would not be valid anymore as they use a strong simplified model [4].

[3] introduces the following terms. The *state* of a system is a set of variables which describes the system at any time (considering the purpose of the study). An *event* is an occurrence, which might changes the state. *Attributes* are properties of entities. An *activity* is a time period of specified length. *Endogenous* activities or events happen within the system, *exogenous* in an environment that affects the system. For example, having a communication system, entities can be messages with their length or destination as attributes. The message transmission can be seen as an activity and the arrival at the destination as an event. A state variable of the communication system can be for instance the number of messages waiting to be transmitted.

2.1.2 Purposes, advantages & disadvantages

Simulation studies are carried out for several purposes bringing along advantages and disadvantages. By simulation, a system is investigated to gain information about it, e.g. for theory construction [5]. "What if" questions about the system can be explored and insights into the interactions of variables can be derived by varying the input data and studying the resulting output [3]. The goal of a simulation study can even be the understanding of how the system works, not necessarily the results of the study. Defining how the system works often opens insights into what needs to be changed to improve its operation [4]. System parameters, which lead to increased performance, can be determined by means of simulation. Furthermore, the future behavior or development of a system can be predicted [5]. For example, how potential changes on the system would impact the system's performance [3]. Simulation also supports *planning*, as information for the construction and dimensioning of a system or its subsystems can be gathered [5]. A system still in the design state can be investigated or new designs or policies can be tested before they become implemented. Possible improvements might be already identified during the design process of the simulation model [3]. Moreover, simulation can be used for pedagogical purposes and allows learning without the cost of on-the-job instruction [3]. Besides, simulation allows to carry out experiments which can not be done on the real system. For instance, when the (real) system does not exist yet or is not available or influenceable for or by the experimenter (e.g., another star system). Other reasons are that an experiment would be too dangerous or not ethically correct (e.g., nuclear tests), too expensive or even destroy the system. Experiments on the real system are also impossible when it is not practicable to observe the desired process in the real system or the experiment aims to predict the state of the system in the future (e.g., weather forecast) [5]. Beyond that, analytical solutions can be verified using simulation [3].

However, it is not always appropriate to instrument simulation. [3] claims that simulation is not suitable when:

- The problem can be solved by common sense or analytically.
- It is less expensive to do the experiments on the real system or the costs of a simulation study exceed the savings.
- The resources or time needed for a simulation are not available. For example, neither input data nor estimations are obtainable.
- There is not enough time or personnel available to conduct verification and validation of the model.
- The system is too complex or can not be defined.

Besides that improving performance-price ratios of computer hardware increases the popularity of simulation [4], investigators can benefit from the following advantages:

- Simulation allows to imitate also complicated or stochastic structures as a model. The simplification of those for analytical methods could alter the system too much and thus lead to falsified results [5]. [4] defines simulation as a versatile and powerful tool, as it is able to deal with very complicated models of correspondingly complicated systems.
- Simulations can be used in stationary and non-stationary cases, whereas analytical solutions can often only be deduced for the steady state [5].
- With a simulation, it is easier to apply different system configurations or change the environmental conditions [5].
- A simulation allows to investigate the system over a longer period [5], or time can be compressed or expanded to enable the observation of a speeded-up or slowed-down process [3].
- Results of a simulation can be presented very realistically (e.g., flight simulator), which leads to higher user acceptance [5].
- By building the model, more insights in and knowledge of the real system are gained [5]. Bottlenecks can be discovered [3].
- The experimentation with the model could reveal attractive alternatives, which could not have been tried out with the real system [4].
- Mistakes on the simulation do not effect the real system [4]. Using simulation, new features can be explored without interfering ongoing operations of the real system [3].
- In some cases, it is possible to perform quick and valid decision making [4].

- Hypotheses about the occurrence of phenomena can be tested for feasibility [3].

On the contrary, simulation entails the following disadvantages:

- Results of stochastical simulations disperse per se. For exact deductions, several simulation runs have to be carried out which then have to be analyzed with statistical methods [5]. It is hard to determine how long is "long enough" or how often is "often enough" [4]. Furthermore, it is sometimes difficult to determine whether an observation results from system interrelationships or from the randomness [3].

- The development of simulation models can be very time consuming and expensive. In some cases, special software is needed [5].

- With analytical solutions, a similar system can be deduced easily, whereas with simulation all steps have to be repeated with the modified parameters [5]. In some cases, simulation is used although analytical solutions would have been possible and preferable [3].

- Easy comprehensible results of simulations might bring along certainty which is not necessarily justified [5].

- When the experiments are performed with the real system, investigations are unquestionably conducted on the "right thing". There have to be no worries whether a model is a valid representation of the system [4]. [4] refers to a *Type III Errors* when right, simply and clear answers are found to the wrong questions, for example because of an over-simplified model of a system.

2.1.3 Classification of systems, models and simulations

The following paragraphs describe how systems, models and simulations can be classified.

Physical or logical models

[4] distinguishes between *physical* and *logical or mathematical* models. A physical model is a physical replication — either scaled or real size — of a real system. For instance, future pilots can be trained in flight simulators. The enlargement of an atom or the scaled-down version of the solar system are scaled models [3]. A logical or mathematical model, which is usually represented in a computer program, is a set of structural and quantitative approximations and assumptions about the functionality of the (future) system [4]. Those are either traditional mathematical tools like queueing theory, differential-equation systems or linear programming, or an algorithm to produce numerical answers when it is not possible to derive a simple closed-form formula. [3] defines a *simulation model* as a particular type of a mathematical model. [5] even lists the description of the relationship of "ego" and "super-ego" as an example for a *verbal model* in the psychological theory.

Static versus dynamic systems and models

[4] states that in *dynamic models*, time plays a natural role and most operational models are dynamic, e.g. a manufacturing model. Tighter definitions are given by [3], which specify a *static simulation model* as the representation of a system at a particular point in time and a dynamic model as the representation of a system as it changes over time.
[5] states that there exist many definitions of static and dynamic models of different authors. One of them claims that a system is dynamic when its state changes or can change during the considered period of time. Another one defines a model as dynamic, when model variables change over time. In economics or social science, a model is dynamic when the state of a model depends on a former state, for example, when the gross national product is determined by the previous year's value.

Continuous versus discrete systems, models and simulations

In a *discrete system*, the state changes only at a discrete set of points in time (e.g., the number of customers in a bank). On the other hand, [3] refers to a *continuous system*, when the state changes continuously over time (e.g., the water level behind a dam). The definitions of a *discrete* and *continuous model* are analogous, whereby a discrete model is not always used to model a discrete system and a continuous model does not always represent a continuous system. When a model consists of entities which are subject to both continuous and discrete changes, the model is called *mixed continuous-discrete model* [4]. For example, a model of a refinery where the pressure in the vessels changes continuously and shutdowns occur discretely, is a mixed model. However, the choice of using a discrete, continuous or mixed simulation model depends on the characteristics of the system and the goal of the study [3]. According to [3], a *discrete-event system simulation* is

> *"the modeling of systems in which the state variable changes only at a discrete set of points in time."*

[5] claims, that for classification, a distinction between *quantities* and *processes* in a system and model is necessary, respectively. A continuous quantity (e.g., time intervals) can be defined with an arbitrary value out of *infinitely* many values of a finite interval, whereas a discrete quantity accepts only values out of a *finite* set in a finite interval (e.g., number of persons waiting in a queue). However, sometimes continuous quantities in a real system become discrete in the model because of measuring accuracy, or discrete quantities become *quasi-continuously*. For instance, a quantity in the scope of trillions is quasi-continuously changing when only the unit position is increased.

According to [5], a *discontinuous process* is an event without duration, which causes a sudden state change ("shift") at a certain point in time. Furthermore, continuous processes presume continuous quantities and discontinuous (discrete) quantities lead to discontinuous processes. According to [5], in the context of simulation a combination of discontinuous processes and continuous quantities can be classified as *discrete-event simulation*.

[5] argues, that within a simulation, there are just countably many shifts on the quantities and thus countably many manifestations of a quantity. Table 2.1 shows the proposed classification by the kinds of process and quantity.

		Quantities	
		continuous	discrete
Process	continuous	continuous simulation	—
	discontinuous	discrete simulation	

Table 2.1: Classification of simulations by processes and quantities according to [5].

Deterministic versus stochastic models and simulations

In [3], a *deterministic model* is defined as a model where known inputs result in a unique set of outputs, i.e. a model which does not contain random variables. On the contrary, a *stochastic model* has at least one random variable as input and thus random outputs, which are just statistical estimates of the true characteristics of the system. A model can have both stochastic and deterministic inputs [4]. In [5], *stochastic simulations* include stochastic processes, realized as stochastic variables which influence the simulation runs and cause the outputs to disperse.

Models and systems with or without feedback

In systems without feedback, there is just one direction of the causality, i.e. *quantity A* affects *quantity B* but not vice versa [5]. Well known examples in engineering area are control engineering for systems with feedback and systems with open-loop controllers for systems without feedback.

The following examples illustrate the application of models with and without feedback in the field of queuing theory [5]. For instance, in a supermarket, the length of a queue depends on the number of customers who enqueue and those who dequeue after they were served at the desk. Input data are the time between arrivals and the service time which are usually meant to be invariant. Alternatively, the number of new arrivals at the queue of a sidewalk ice cream shop may depend on the current length of the queue. Next, the customers can decide if they want to queue up and are not forced to do it because they want to leave the shop (as in the supermarket). The sidewalk shop would be represented in a model with feedback.

Terminating or non-terminating systems

A system is *terminating*, when there is a natural end of the system's processes during the investigated period of time. More precisely, the system *can* run permanently, but the considered processes terminate. In other words, the processes can be repeated, but the end state of a terminated process will not be the initial state of the next repetition [5].
For instance, the line in a shop forms after the shop opening, does probably disappear during the day and is eventually gone when the shop has closed. Then, all processes of the system are terminated.

Of course, the line can form again at the next shop opening, but the line state of the last opening has no affect on the new opening [5].
A system is defined as *non-terminating*, when it or its processes run permanently [5]. In contrast to terminating systems, the state is not reset after an interruption. For example, the stock size of a warehouse is daily reduced when goods leave the warehouse and increases on a weekly basis by deliveries. A longer interruption like the end of the work day does not affect the process of the stock size, as the stock is as large in the evening as on the next morning.
Usually, the whole period of a terminating system and just a finite time span of the infinite duration of a non-terminating system is investigated by simulation [5].

Stationary or non-stationary systems

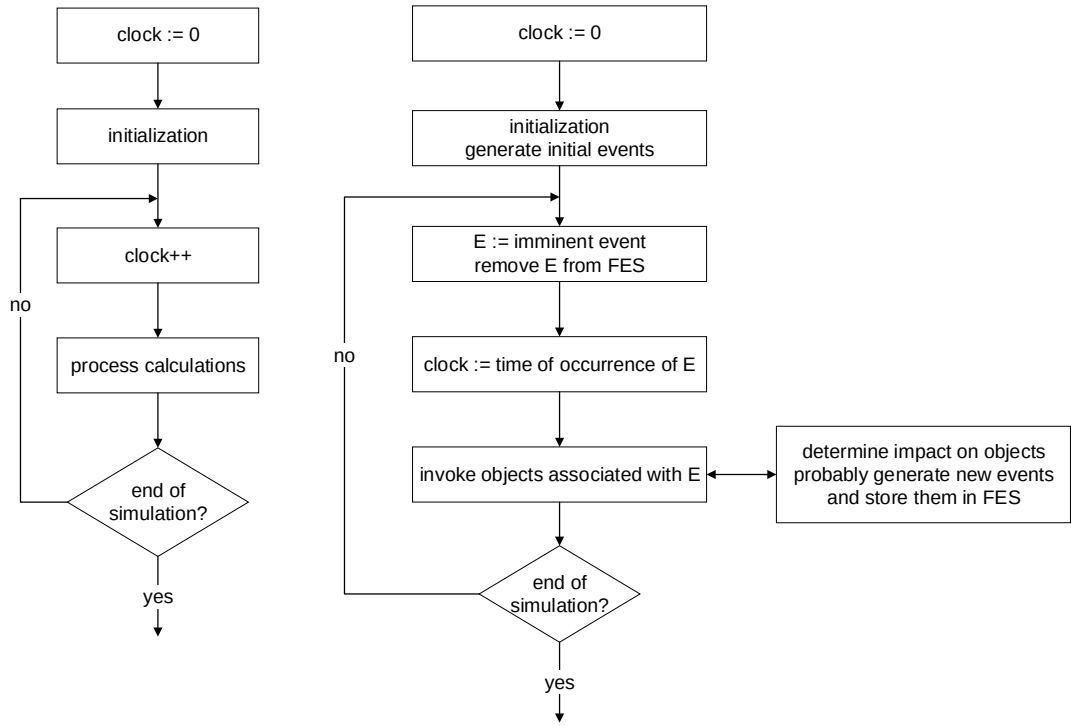
[5] cites that (in discrete systems) the probability of a certain state remains the same in the stationary phase. The dispersion of simulation quantities increases during the transient state and has a steady maximum in the stationary phase. Systems which stay in non-stationary state have for instance input data varying over time or the system itself has a permanent trend (e.g., the earth's population).

Real-time versus non-real-time simulation

Real-time simulations execute like the modeled system, for example a flight simulator, or have real data as input. The *non-real-time simulation's* duration depends on the computing capacity and does not correlate with the real time. When the purpose of the simulation is to provide a prediction (e.g., of climate or geological changes), the simulation has to run faster than in real-time. Sometimes, it is also necessary that simulations slower processes down, for instance when effects in nuclear physics are investigated [5].

Time-, event- or process-oriented simulation

The basic principles of a *time-* and *event-oriented simulation* are depicted in the flow-charts in Figure 2.2a and 2.2b, respectively. In a time-oriented simulation, the state changes with every incrementation of the time. Thus, the granularity of the time specifies the duration of a state transition. The simulation stops when the simulation clock reaches a certain value or another termination condition is fulfilled [5].
In the *event-oriented simulation* approach, the computational work depends on the occurrence of events and their impact [4]. These events are stored in an *event queue* and thus processed according to their scheduling time. The event triggers a state change on the associated object and new events are probably generated and stored in the event queue. The execution of the event itself is not time consuming. This kind of simulation terminates when the event queue is empty, a certain event is triggered or a specified point in simulation time is reached [5]. In this approach, one *master controller* manages all events, entities, attributes, variables, and statistical accumulators [4].



(a) time-oriented simulation (b) event-oriented simulation

Figure 2.2: Time- and event-oriented progress in simulations [5].

In a *process-oriented simulation*, the simulation is seen from the point of view of an entity "as it works its way through the model", which is comparable to flowcharting [4]. Using this approach, the simulation time progresses while an object's method is executed where the state of the object is changed. This is usually done with a *waiting* function. The object is deactivated until the waiting period is over or another event occurs. For instance, when a customer enters the cash desk, the desk changes its state to "busy", waits as long as the service requires and changes then back to the state "available". Furthermore, other processes may run in parallel when a process is waiting. Thus, the waiting process has to store its state and to recover, when it is activated again [5].

Other simulation approaches

Additionally, there exist other types of systems and simulations, like *chaotic systems*, *finite-element-simulation* and *Monte-Carlo-methods* [5]. In [7], a backbone network was simulated using the OPNET Modeler. Therefore, a *hybrid simulation* (real-time according to [5]) was carried out as previously measured real network traffic was imported as "background traffic"

into the simulation model [7]. According to [8], *emulation* is performed when the simulation exchanges data with real network devices. In [9], an interface was implemented which enables communication between simulated and real, IP-based hosts in a network. This module was extended to use it for an emulation environment for Message Passing Interface (MPI) programs [10, 11].

2.1.4 Simulation methodology

In [5], a *basic model* is provided, which shows the required steps to be processed for deriving a simulator from a real system. This basic model is depicted in Figure 2.3 and shows the dependencies of the real or hypothetical system, the formal (mathematical) model and the computer model (simulator). The initial step is the *model building* process, where the real system is transformed into a mathematical or formal model. Therefore, a system analysis is carried out to study the structure of the real system including interactions and behavior of its components. In the phase of the *model suitability test*, the formal model is tested whether it represents the system sufficiently to answer the research questions of the simulation study. During the *model implementation*, the formal model is built into an executable simulation program (according to usual software engineering paradigms). *Model verification* examines if the computer model is an accurate representation of the formal model. To complement the basic model, *model validation* is performed to check if the computer model is an adequate representation of the real system, which is done by comparison of the simulation results with expected or results of the real system. If necessary, the formal model or implementation has to be reviewed and modified and the process starts again (not indicated in the graph).

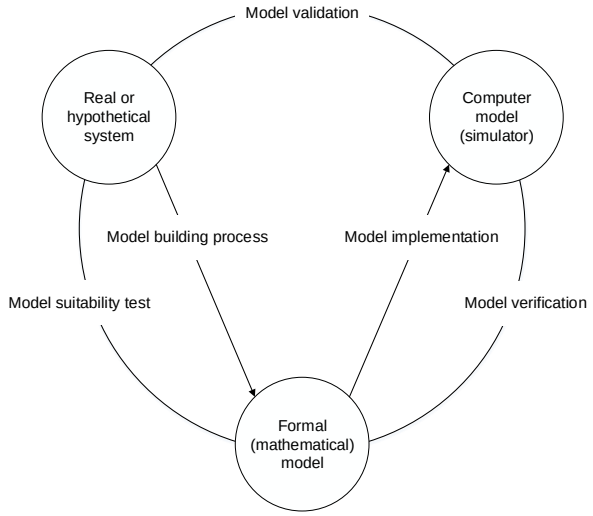


Figure 2.3: Basic model of a simulation study according to [5].

The steps processed for a thorough and sound simulation study according to [3] are depicted in Figure 2.4. The first step of the process flow is the *Problem formulation*, where the problem is stated by analysts or the owners of the problem. Occasionally, the problem formulation has to be repeated during the study (not indicated in the graph), when a problem is encountered. In [5], this step also includes the delimitation of the real system and a definition of the model's level of detail.

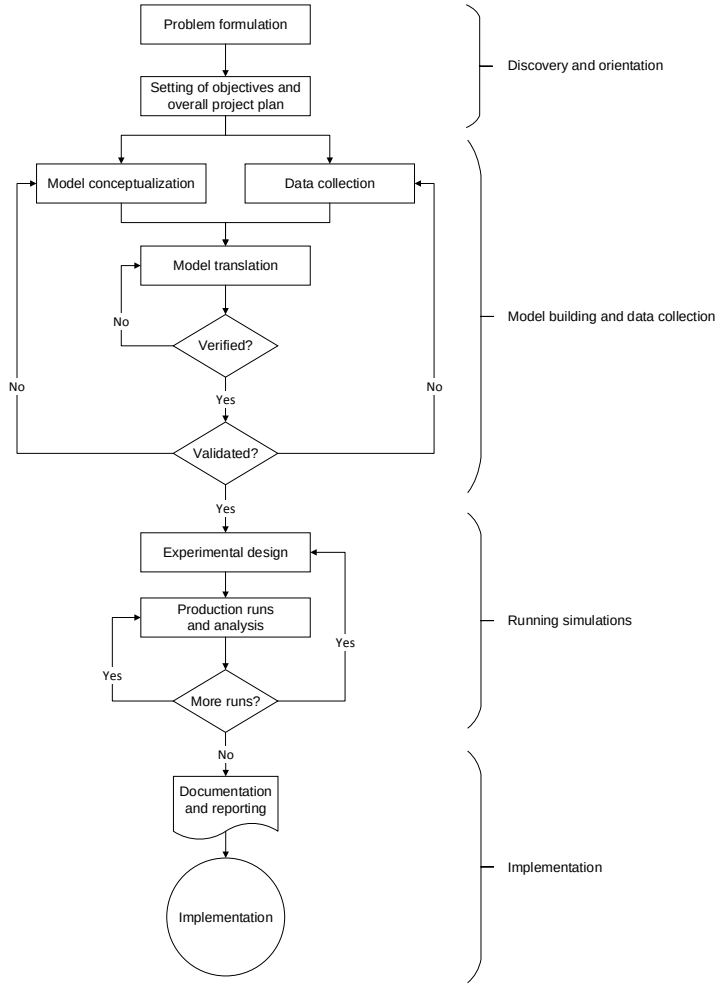


Figure 2.4: Process flow model of a simulation study according to [3].

In the second step called *Setting of objectives and overall project plan*, the questions to be answered are declared as aims of the study. Additionally it is determined, whether simulation is the appropriate tool to solve the problem and accomplish the objectives. When simulation is appropriate, a project plan is worked out. This plan includes a statement of alternative system designs to be considered and how they can be evaluated with respect to their effectiveness.

Additionally, the plan contains common points of project planning, like expected costs, the amount of working-days required to accomplish different work packages and also results expected at the end of various milestones [3].

The steps *Model conceptualization* and *Data collection* take place in parallel as there is a constant interaction between them. [3] describes the construction of a system's model to be as much art as science. The model improves with the designer's ability to abstract crucial features of a problem and "to select and modify basic assumptions that characterize the system" [3]. Using these essences, the model is elaborated until it is a useful approximation of the system. [3] recommends to start with a simple model and successively add more complexity. This complexity should not exceed the desired complexity needed for the purpose of the study, as it would just cost unnecessary resources. There is no need of an one-to-one mapping between the model and the real system [3]. In [5], the kind of the simulation, either deterministic or stochastic, or time-, event- or process-oriented, is specified in this phase. Along with the changing of the model's complexity, the required data elements can vary. The kind of data to be collected depends on the objectives of the study. For instance, when the study is about the dependency of lengths of waiting lines and the number of cashiers, distributions of interarrival times and service-times for the cashiers are needed. Historic distributions on the lengths of queues will be used to validate the model [3]. In the fifth state, the *Model translation*, the conceptual model is transformed into a *computer-recognizable* format. This can be either done by programming the model in a simulation language or using special simulation software. Simulation languages are more powerful and flexible, but development effort can be reduced when it is possible to embed the problem into a simulation software [3]. The (repeatable) step *Verified?* checks if the computerized representation (operational model) of the conceptual model is correct, i.e., if the conceptual model is reflected accurately in the operational model. The phase *Validated?* is carried out to examine, if the conceptual model is a sufficiently accurate representation of the real system. [3] describes that this is usually done by calibration, where the behavior of the computerized model is compared with the real system's behavior and the model is successively improved through gained insights and revealed discrepancies. As indicated in Figure 2.4, the phase of *model building and data collection* can be repeated many times and a continuing interplay might be necessary among its steps until an acceptable representation of the real system is achieved [3].

For each system design that is investigated, the nature of the experiment (e.g., the length of the initialization period and run, the number of replications) is specified in the phase *Experimental design*. Which alternatives are simulated might depend on results of already completed and analyzed runs, as indicated by the feedback arrow originating from step the *More runs?*. In *Production runs and analysis*, the defined experiments are carried out. Subsequently, estimations of measures of performance for the simulated system designs are obtained through analysis. Depending on the results, the analyst decides if more runs or even experiments with different system designs are needed [3].

When all runs are completed, the step *Documentation and reporting* follows. On the one hand, *program documentation* is created to support the understanding of how the simulation environment works, for example when it is used again or modified later by the same or another analyst, or when model users want to learn about the relationships between input parameters and output measures by changing parameters. On the other hand, frequent deliveries of *progress*

documentation during the project provide important information about work done and made decisions during the simulation project. A final report will allow the model users to review the final problem formulation, the investigated alternative systems and how they were compared, the results of the experiments and proposed solutions to the stated problem. Additionally, the final report should add to the credibility of the model and its building process and provides a foundation of certification for higher level decisions [3].

The last step *Implementation*, indicates the completion of the study. According to [3], a successful implementation "depends on continual involvement of the model user and on the successful completion of every step in the process". [3] claims that the validation step is perhaps the most crucial one, as it prevents from building an invalid model and thus Type III Errors (see Section 2.1.2).

2.1.5 Fundamental concept of discrete-event simulation environments

This section describes the fundamental manner of functioning of discrete-event simulations. Additionally to some basic blocks defined in Section 2.1.1, the following components are introduced or repeated [3]:

- A *list* is a "collection of (permanently or temporarily) associated entities". It is ordered in some logical fashion.
- An *event* is an "instantaneous occurrence that changes the state of a system".
- An *event notice* is a "record of an event to occur at the current or some future time, along with any associated data necessary to execute the event".
- An *event list* is a "list of event notices for future events, ordered by time of occurrence". The list is always sorted by the event time indicated in the event notice. It is also known as *Future Event List (FEL)* or *Future Event Set (FES)* [12].
- An *activity* is "a duration of time of specified length, which is known when it begins". The duration is characterized and defined by the modeler, either deterministically (e.g., exactly 3 minutes), stochastically (e.g., drawn at random from a set with equal probability) or by a function depending on system variables and/or entity attributes.
- A *delay* is a "duration of time of unspecified indefinite length, which is not known until it ends". For example, the customer's delay in a Last In First Out (LIFO) waiting queue depends on future arrivals of customers.
- *Clock* is a variable which represents the simulated time.

The duration of an activity is computable at the instant it begins and is not affected by other events, unless this is enabled by the simulation package. In such a case, the simulation environment allows canceling or postponing of an activity in progress. Thus, to keep track of an activity and its completion time, an event notice for an end-of-activity event is created at the simulated begin of the activity's duration.

This event, often called *primary event*, occurs at the completion time of the activity [3]. For example, if the current value of *clock* is 50 and an activity A with duration 5 is just beginning, an event notice for an event "end-of-activity-A" with occurrence time 55 is created [3].
On the contrary, the duration of delays is not predefined a priori and is rather determined by system conditions, e.g., when some logical conditions become true or other events occur [3]. However, the completion of delays, also named *conditional* or *secondary events*, are not managed by event notices stored in the FEL. The associated entities are placed in another list until the system conditions admit their processing [3]. A delay is also referred to as *conditional wait* and an activity as *unconditional wait* [3].

As dynamic systems change over time, the simulation system's state, number of active entities, attributes, the contents of sets, as well as activities and delays currently in progress are changing over time. These components are functions of the simulation time *clock* [3]. As already stated in Section 2.1.3, a discrete-event simulation models a system which changes at certain points in time. Those are the points when an event occurs. The evolution of the system over time is represented by a sequence of system snapshots (images) [3]. A snapshot at $clock = t$ includes the current system state, FEL, sets, statistical accumulators and other values at time *t* [3].

Event-scheduling/time-advance algorithm

The FEL is the key element in the event scheduling and time advance algorithm. It holds a list of event notices for events which are scheduled in the future time. Scheduling a future event means that, at the time an activity begins, an end-of-activity event notice is placed into to the FEL, whose event time is the activity's begin time plus its duration [3]. At $clock = t$, the FEL contains all events scheduled before t which will occur in the future. The FEL is always sorted in chronological order by the event times $t_1, t_2, \ldots$. Thus, the event times in a FEL satisfy

$$t < t_1 \leq t_2 \leq t_3 \leq \ldots \leq t_n.$$

The *imminent event* is the next event which will occur, in the above case the event associated with time t_1. After the system image has been updated at $clock = t$, the simulation time *advances* to $clock = t_1$, the event notice associated with t_1 is removed from the FEL and its event is processed. Thereby, the event changes the snapshot of time t, which results in the new snapshot for time t_1. When new future events are generated at t_1, they are scheduled by placing them into the right (temporal) position on the FEL. Then, the simulation time is advanced to the event time of the new imminent event and the procedure starts again. This is repeated until the end of a simulation, which is either triggered by scheduling of a "stop simulation" event at a specified future time or by waiting for a certain event to occur. The time of this event is not known apriori, it might even be an interest of the simulation study [3]. The *event-scheduling/time-advance algorithm* refers to the mechanism a simulator has to perform to advance the clock and build a new system snapshot [3]. As described, unlike real time, the simulated clock does not flow continuously but rather jumps from the time of one event to the time of the next event [4].

The management of the FEL has a major impact on the model's computer runtime, as the FEL is subject to permanent changes during the simulation progress. The main operations are the removal of the imminent event and the insertion of a new event somewhere in the list with respect to the logical organization of the ordering [3]. In the Objective Modular Network Testbed in C++ (OMNeT++), the FEL is organized as a binary heap which is stored in an array. A circular buffer is realized to store the events which are scheduled at current simulation time[1] [12].

Furthermore, the question of what happens if two events have the same event time may arise. In some simulation systems like e.g., OMNeT++, events are realized as messages which are sent from one module to another. These messages are stored in the FES. The time an event occurs is the *arrival time* of the message [12]. As stated in [12], events are consumed from the FES applying the following rules to meet causality:

1. Given two messages, the message with the earlier arrival time is executed first. If the arrival times are the same,
2. the message with the smaller *scheduling priority* value is consumed first. If the priorities are the same,
3. the message which was sent earlier is processed first.

The scheduling priority is a message attribute which can be assigned by the user.

2.1.6 Network simulation tools

[13] lists different modeling and simulation tools used in the field of communication networks. According to [13], the tools can be classified into four groups:

- *Analytical tools* are used to design a network model and calculate various factors, e.g., reliability or utilization.
- *Simulation tools* are, besides modeling, able to simulate dynamic behavior of networks, e.g. the Transmission Control Protocol (TCP) protocol, packet flow or link failures. Mostly, it can visualize the dynamic behavior of a system whereas analytical tools can not.
- *Topology discovery tools* are used to extract an actual network topology and present it in a graphical or textual manner.
- *Topology generation tools* allow to generate network topologies based on different algorithms.

For the purpose of this thesis, a hybrid of an analytical and simulation tool is needed. [13] divides simulation tools by their field of usage, either educational, commercial or specialized. Some of the educational simulation tools mentioned are *ns-2*, *pdns*, *MaRS*, *Network Workbench*,

[1] see `cmessageheap.h` and `cmessageheap.c` of OMNeT++ 4.2.2

WIPSIM, *GTNetS* and *NetSim*. *OPNET* is listed among the commercial tools, all other tools provided in [13] are currently not available. *ATM-TN*, *GloMoSim* and *QUIPS-II* are stated as examples for software in specialized fields.

The last version of ns-2[2] was released in November 2011 and is developed in C++. Simulation models are written in OTcl (an objected oriented version of Tool command language (Tcl)) scripts and the behavior of a model can be visualized with *nam* (*Network Animator*) [13]. New object classes have to be programmed in a OTcl class and added to the original source code of ns-2 [14]. Hereby, re-compiling is necessary, but inefficient [15]. pdns[3] is an extension of the ns-2 simulator and enables the user to run ns simulations in a distributed environment. ns-3[4] is a new release of ns-2, which is not backward compatible to ns-2. Both versions are available for free though [15]. ns-3 is mostly used in the field of wireless IP simulations. Network Workbench[5] contains a protocol stack abstracted from the TCP/IP stack including Ethernet and TCP/IP [13]. Apparently, it does not implement a full version of IP but rather a protocol with limited functions [16]. Ethernet is only implemented as an abstracted version, too. MaRS[6] only supports network routing protocols. The support of protocols on the transport and application layer is very limited [13]. The WIPSIM[7] project started in 2000 and its focus is put on the Media Access Control (MAC) and IP layer. Due to the small user and developer base of WIPSIM, there exists some uncertainty about the correctness of the implemented protocols [13]. The network simulation environment GTNetS[8] is based on C++ and supports a number of protocols including Ethernet and Internet Protocol version 4 (IPv4). Building a network simulator is done by developing a C++ main program that instantiates the network elements, applications and protocols. After compiling and linking this program with GTNetS libraries, the simulation executable can be run. This can be a major drawback, when the program has to be re-compiled everytime the setup of a simulation experiment changes. GloMoSim is a simulation environment for wireless mobile networks and thus cannot be used in the context of this thesis. NetSim[9] is available under a proprietary license. QUIPS-II is specialized on the design and evaluation of IP networks using differentiated service (*DiffServ*) for Quality of Service (QoS) mechanisms [17]. ATM-TN[10] is used to simulate Asynchronous Transfer Mode (ATM) networks. The commercial tool OPNET[11] is not considered as it is not available for free. [15] compares a set of different network simulators including ns-2 and OMNeT++[12]. According to this comparison, OMNeT++ is the most efficient open-source environment among this selection for this work.

[2] http://www.isi.edu/nsnam/ns/

[3] http://www.cc.gatech.edu/computing/compass/pdns/

[4] https://www.nsnam.org/

[5] http://netlab.gmu.edu/networkbench/

[6] http://www.cs.umd.edu/projects/netcalliper/software.html

[7] http://ostatic.com/wipsim

[8] http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/

[9] http://tetcos.com/netsim_gen.html

[10] http://warp.cpsc.ucalgary.ca/Software/ATM-TN/atm-tn.php

[11] http://www.riverbed.com/products-solutions/products/network-performance-management/network-planning-simulation/Network-Simulation.html

[12] http://www.omnetpp.org/

OMNeT++, is an open-source library and framework for building network simulations based on C++. According to [12], it has a broad field of application and can be used e.g., for modeling of wired and wireless communication networks, queuing networks, multiprocessors and other distributed hardware systems or even for validating hardware architectures. Generally, any system where the discrete event approach is applicable, can be modeled and simulated using OMNeT++ [12]. The simulations can be run on a graphical, animated user interface or command-line based which is very practicable for batch execution. OMNeT++ is free for academic and non-profit use, for commercial purposes a license can be obtained [12]. In [8] is stated, that OMNeT++ itself is not a network simulator, it rather includes the machinery and tools to write simulations. The components needed for various applications are rather provided by simulation models and frameworks[13] [8].
INET[14] is a framework, which is based on a modular approach and provides models for protocols of the International Organization for Standardization (ISO) Open Systems Interconnection (OSI) layers, such as Ethernet, Address Resolution Protocol (ARP), Internet Control Message Protocol (ICMP), IPv4, Internet Protocol version 6 (IPv6), User Datagram Protocol (UDP), TCP or Hypertext Transfer Protocol (HTTP). New mechanisms can be added and the already implemented protocols can be modified and extended easily. *INETMANET*, developed in parallel to INET, is an extension which adds protocols mainly used for mobile ad-hoc networks. *MiXiM*[15] is a framework which contains protocols for mobile and fixed wireless networks. According to its webpage, it offers detailed models of radio wave propagation, interference estimation, radio transceiver power consumption and wireless MAC protocols such as *Zigbee*. *TTE4INET* is an extension of INET for the simulation of the time-triggered Ethernet (TTEthernet).

From the INET framework, the models for Ethernet, ARP and IPv4 can be used for the simulations in the context of this thesis. The combination of OMNeT++ and INET, along with their flexibility (even the FES can be accessed, see Section 3.3.1), makes a very powerful tool and is thus chosen for this work.

2.2 Virtual Local Area Network

VLANs provide the possibility to create independent logical networks within a physical network [18]. Using VLANs, one physical Local Area Network (LAN) can be treated as many logical LANs [19]. These logical LANs are grouped by end-station or switch characteristics, or frame protocols — there is no need to change the cabling. The stations do not have to be on the same LAN, it is sufficient when there exist physical connections (like backbones and bridges) among them [18]. A *trunk link*, incident to a switch's *trunk port*, is a connection between two switches which is able to carry traffic of multiple VLANs at a time [20]. Therefore, VLANs can be used to realize advanced switching functions such as redundancy mechanisms (see Section 2.3).

[13] `http://omnetpp.org/models/catalog`
[14] `http://inet.omnetpp.org/`
[15] MiXiM, `http://mixim.sourceforge.net/`

VLANs can be classified by the way the VLAN membership is determined. [18] distinguishes between *attribute-* and *protocol-based* VLANs. With the attribute-based approach, each switch possesses an *access list* which holds attribute/VLAN associations. When it relays a frame, a switch can deduce whether ports are members of VLANs based on these mappings. In [18], the attributes include the switch port number, the station MAC and IP address. With the protocol-based method, the VLANs are determined by characteristics of the frame, not the station characteristics or switch port setting [18]. The most commonly used technique is *frame tagging*, which is also employed in the IEEE 802.1Q standard [21].
[22] refers to *static* VLAN assignment — also known as *port-based* VLANs — when a port is assigned to one particular VLAN. Hereby, the drawback is that the configuration has to be changed after a workstation has moved. Thus, with *dynamic* VLAN membership determination the switch retrieves information about the port's VLAN from a VLAN Membership Policy Server (VMPS). The VMPS maps the MAC address of the host, which is connected to the requested port, to a VLAN [22].
The following *VLAN association* rules for determining the VLAN membership are described in [23]. Based on these rules, a *VLAN-aware* device (station or a switch) decides about the belonging of a frame. A VLAN-aware switch makes its forwarding decision not only on the destination MAC address, but primarily on VLAN information associated with the frame. [23] underlines, that from the device's perspective, a *frame*, not the station, protocol or application is unambiguously associated with one VLAN.
Implicit tagging is applied for membership determination, when a frame is parsed and a set of association rules is applied on the gained information [23]. This is typically done on switches which connect VLAN and non-VLAN environments. For example, an implicit tag could be the source MAC address, the protocol type, higher-layer network identifiers or application specific fields. *Explicit tagging* or simply *tagging* is done by adding a particular tag (*VLAN tag*) to a frame [23]. For instance, a VLAN-aware switch receives an untagged frame, applies the association rules, tags the frame and forwards it to a VLAN-aware environment [23]. Accordingly, the switches also have to remove a tag when the frame is relayed from a VLAN environment to VLAN-unaware domains. Both, the tagging and untagging requires a recalculation of the Frame Check Sequence (FCS) [23]. As the VLAN tags are transparent to VLAN-unaware switches because of the frame format definition, VLAN-unaware switches are able to process tagged frames according to their MAC addresses (i.e. like usual frames) [23].
In *port-based* VLANs, the membership determination entirely depends on the port where the frame arrived [23]. Thus, no frame parsing is necessary. *MAC address-based* mapping relies on the source MAC address of the frame. It facilitates *user mobility*, as the user is associated with its address and not the port it is connected to. In case of unkown source addresses, some switches stick to the rules and do not forward the frame, saving the integrity of the VLANs and providing security. Other switches just forward the frame based on the destination MAC address, i.e. broadcast frames with unknown destination addresses [23]. However, port-based and address-based mappings can be combined. *Protocol-based* VLAN association is based on the used protocols, e.g. IP, Internetwork Packet eXchange (IPX) or AppleTalk. Combining this with a mapping based on source MAC addresses, a VLAN for each set of protocol-specific application can be created [23].

In the example depicted in Figure 2.5, station 4 is a server which provides IP- and IPX-based application services. Station 3 cannot access the IP-based service as it is on a different IP subnet, but is able to use the IPX-based as it is on the same IPX network. Furthermore, all stations are able to use the printer (station 5), as they are all members of the VLAN defined by the AppleTalk protocol.

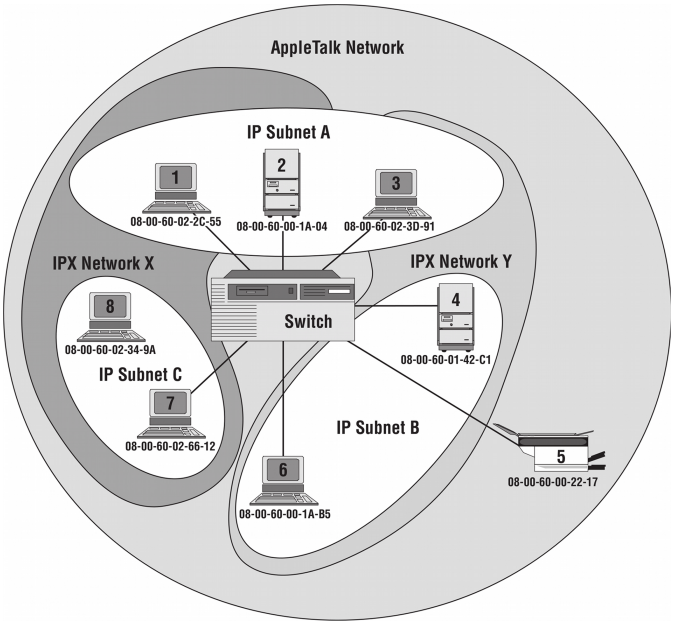


Figure 2.5: VLANs are defined for each set of protocol-specific application [23].

Using the *IP subnet-based* mapping, the membership is based on the subnet of the source IP address contained in the encapsulated IP datagram. The switch extracts the IP subnet part from the address using a configured subnet mask. *Application-based* VLAN mapping requires the switch to examine the type of application from an untagged frame, which can be a complex process and should not be a switch's task. Thus, frame tagging by VLAN-aware end stations provides a better solution. However, the resulting VLANs may be very transient as the stations and applications running on the end stations can change frequently [23].

The standard IEEE 802.1Q implements frame tagging. It specifies a tag (of length 4 bytes) which is inserted into Ethernet frames. The first two bytes after the source MAC address is the Tag Protocol Identifier (TPID) field, which stores the specified value of $0x8100$. Usually, in untagged Ethernet frames, this is the position of the *Type/Length* field. When the value in this field is greater than or equal to 1536 ($0x0600$), it is not interpreted as length but indicates the protocol encapsulated in the payload (MAC client protocol) [24]. Since $0x8100$ is larger than $0x0600$, the field denotes a *Q-tagged* frame. The Tag Control Information (TCI) field following the TPID comprises the fields *priority*, *Canonical Format Indicator (CFI)* and *VLAN*

Identifier (VID)[16] [18]. Using the three bit priority field, an eight level frame classification can be performed. The CFI flag is set to one if the physical layer is based on Ethernet. If the flag is zero, the frame is destined for a token ring and thus not forwarded as token ring cannot handle tagged frames [18]. The 12-bit VID defines the VLAN the frame belongs to and thus enables separation of frames belonging to different VLANs [21]. The null VID indicates that the frame only contains priority information [21] and *one* is used as default VID when frames ingress a port [21]. According to [24], the maximum length of Q-tagged frames is extended to 1522 bytes. VLAN tags are not processed by hosts, so they just need to be available between switches [19]. As there exist switches or hosts which are not VLAN enabled, VLAN-compliant switches have to be able to attach and detach the tag [19]. Devices which cannot handle Q-tagged frames, do reject them [18].

The use of VLANs entails the following benefits. For example in a company, VLANs can be used when the physical network structure does not fit the structure of the company. For instance, the finance and R&D departments are located in the same building and use the same LAN infrastructure [19], or the logistics department is split up into two buildings. Then, the users can be grouped by VLANs rather than by the physical network [25]. Being in the same VLAN, they have access to the same servers and applications as if they were on the same physical LAN [25]. VLANs introduce *flexibility*, as it is possible to alter, e.g. add or remove stations [18] to or from the logical topology without changing address assignments, the cabling [26] or physically move stations and people [18]. This also allows to *save costs* of both money and time [18]. According to [18], stations can be members of multiple VLANs and VLANs are easy to configure. Although the VLAN approach is cheaper for providing *user mobility*, it also involves more performance requirements on the VLAN-aware switches [23] [18]. Furthermore, a workstation should not necessarily be added to the VLAN for just occasional communication [18].
Inter-VLAN communication has to be controlled by a router and thus better control of network *security* is reached by distributing users across different VLANs [25]. Then, stations can for instance only eavesdrop on the multicast and unknown unicast traffic within their own VLAN [23], unless one of the vulnerabilities mentioned below is exploited. Also, injected malicious traffic is restricted to the own VLAN and does not disrupt the whole network [23]. For instance, when using VLANs it is possible to separate the guest host from the LAN of hosts or servers with confidential data. Notwithstanding, the concept of VLANs is vulnerable to security threats like *Tagging Attacks*, *Double-Encapsulated 802.1Q/Nested VLAN Attack*, *ARP Attack*, *Multicast Brute Force Attack*, *Spanning-Tree Attack* or *Random Frame Stress Attack* [27].
With VLANs, broadcast and multicast *domains* are *restricted* to a single VLAN and thus the network utilization is minimized and the performance is increased [18,19]. However, using VLANs *bandwith preservation* comes along without any more effort required [23]. For instance, the enormous traffic produced by the R&D department during experiments can be separated from the management in order to not disturb the management's video conference [19].
VLANs can be created *independent* of the installed physical layer (e.g., Ethernet or Token Ring) of the LAN [25], even though token rings cannot accept tagged frames [18].

[16]The priority field is called *user_priority* in IEEE 802.1Q-2003 and Priority Code Point (PCP) from IEEE 802.1Q-2005 on. Furthermore, the IEEE 802.1Q-2011 renames CFI to Drop Eligible Indicator (DEI).

Some of the protocols described in Section 2.3 are based on VLANs or use VLANs to achieve load balancing. Also, the Multiple Spanning Tree Protocol (MSTP) explained in Section 2.3.2 enables defining separate spanning tree instances within a grouping of VLANs [18].

2.3 Network protection and recovery protocols

Network protection protocols can be located on different layers of the ISO OSI reference model. Building resilient networks at the Network Layer (NWL) implies the support of the resilient mechanisms by the end-stations. In contrast, at the Data Link Layer (DLL), resilient networks can be constructed without end-stations involved efficiency issues. Thus, only DLL protocols based on Ethernet are considered in the following.

At the DLL, redundant switched networks are used to build resilient networks. Having multiple paths between two switches, bridging loops and thus broadcast storms may emerge which excessively consume network resources. Therefore, a mechanism is needed to prevent such loops in a topology. On the other hand, it is desirable that the topology is fault-tolerant, i.e. that a redundant path is used in case of a failure.
The following sections describe such protection and recovery algorithms. In general, these protocols change the particular topology by blocking and unblocking certain ports. The period the switches need to agree on the new topology after a change is called *convergence time* [28].

2.3.1 Proprietary protocols

In the following sections, the proprietary protocols Resilient Ethernet Protocol (REP), Ethernet Protection Switching Ring (EPSR), Rapid Ring Protection Protocol (RRPP) and Turbo Chain are described.

Resilient Ethernet Protocol

The REP is a proprietary technology implemented on *Cisco*® Carrier Ethernet switches [29]. Figure 2.6 shows the basic element named *REP segment*, which consists of multiple switches connected through REP enabled ports and is identified with a unique *segment Identification Number (ID)*. The ports at each end of the segment are called *edge ports* and are part of the *edge switches*. In each segment, at least one port — the *alternate port* — is always blocked.

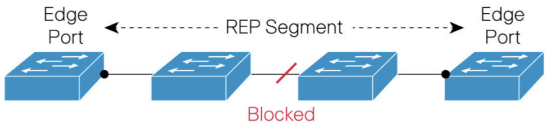


Figure 2.6: A REP segment [29].

A segment can be wrapped into a ring by installing the two edge ports on the same edge switch or connecting the two edge switches. More complex ring topologies are shown in Figure 2.7. The links connecting the edge switches could be running Spanning Tree [29].

However, a segment can be added to various network topologies by “plug and play” which makes REP a very flexible protocol [29]. But, a switch has at most two REP ports with the same segment ID.

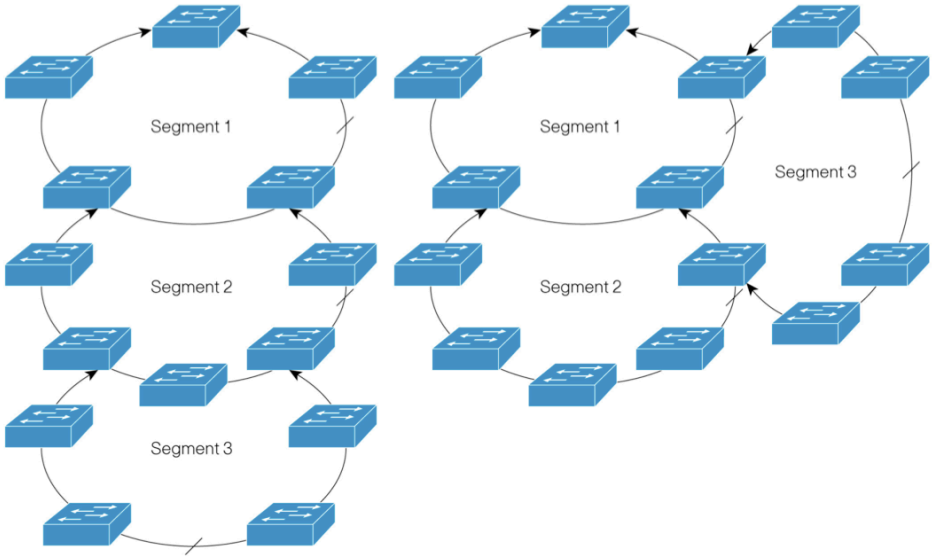
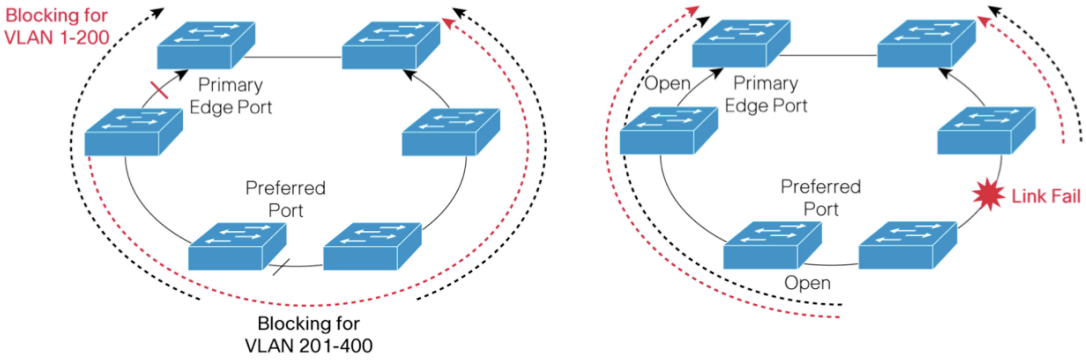


Figure 2.7: More complex network topologies with three coupled rings at a time. The crossed lines indicate the alternate ports and the arrowheads the edge ports [29].

Usually, on an intact segment the alternate port is blocked to establish a loop-free environment. If a failure is detected by one of the following to techniques, REP introduces a switch-over. Basically, the integrity of a link is monitored via *Loss Of Signal (LOS) testing* and additionally two adjacent REP nodes exchange *hello packets* to check the connectivity. After the detection of a failure, the corresponding port sends notifications to unblock the alternate port and initiate the other nodes to flush their Filtering Database (FDB). To speed up the convergence, these notifications are sent with a Cisco multicast MAC address. The *REP Adjacency Protocol*, based on sequence numbering and packet acknowledgement, allows a reliable dispersal. By default, REP blocks one of the restored ports after a recovery. Using the so-called *preemption* feature, a preferred alternate port can be blocked instead after a certain delay or anytime manually. Thereby, the segment remains in a well-known state. Additionally, REP is able to inform other segments about topology changes by sending Topology Change Notification (TCN) messages.

Furthermore, REP supports security by using a key. Just REP nodes possessing the alternate port key are able to unblock the REP alternate port and thus introduce a switch-over. The alternate port automatically generates and distributes its key within the corresponding segment only. So, every segment owns a unique key.

Moreover, it is possible to perform load balancing based on two ranges of VLANs. Then, the primary edge port and a configurable alternate port is defined for the two VLAN ranges. In the example illustrated in Figure 2.8a, two VLAN ranges $1 - 200$ and $201 - 400$ are defined for load balancing.



(a) Intact ring

(b) Ring with a link failure

Figure 2.8: REP load balancing based on VLANs [29].

The alternate port of VLANs $1 - 200$ is the primary edge port and the one of VLANs $201 - 400$ a predefined port "Preferred Port". The dashed arrows indicate the transmission direction of frames per VLAN range. When a link failure happens, both blocked ports are opened and the load balancing is deactivated per se (see Figure 2.8b). After a link recovery, one of the ports incident to the repaired link comes up blocked. If preemption is enabled, the original configuration is restored after the preemption delay.

Although REP does not aim to replace Spanning Tree Protocol (STP), it is able to interoperate with STP in case of topology changes [29]. As exemplified in Figure 2.9, it is possible that a domain controlled by REP is adjacent to a domain controlled by STP, i.e. a switch can offer both REP and STP functionality, but not at once on the same port. REP is able to inform STP about topology changes by creating and sending *Spanning Tree Topology Changes Notification (STCN)* messages, but does not forward STP Bridge Protocol Data Unit (BPDU) messages into the REP part of the network. In the above example, the link failure is propagated with REP TCN messages within the segment, and to the STP domain by the "Edge Switch B" using STCN messages.

Ethernet Protection Switching Ring

EPSR is a proprietary protocol of *Allied Telesis*^{TM} [17] [30] which follows a similar approach as Ethernet Automatic Protection Switching (EAPS). The protocol specifies a set of *data VLANs*, a *control VLAN* and the associated switch ports of switches in a ring as EAPS domain [30]. One node in the ring is referred to as *Master node* whose ports are defined as *primary port* and *secondary port*. In the absence of failures, the Master node blocks its secondary port only for the data VLANs to avoid a loop. The control frames do not circulate as they are consumed by the Master. EAPS implements two mechanisms for failure detection. The Master node either becomes aware of a failure when periodically sent *Healthcheck* frames do not reach the secondary

[17] `www.alliedtelesis.com`

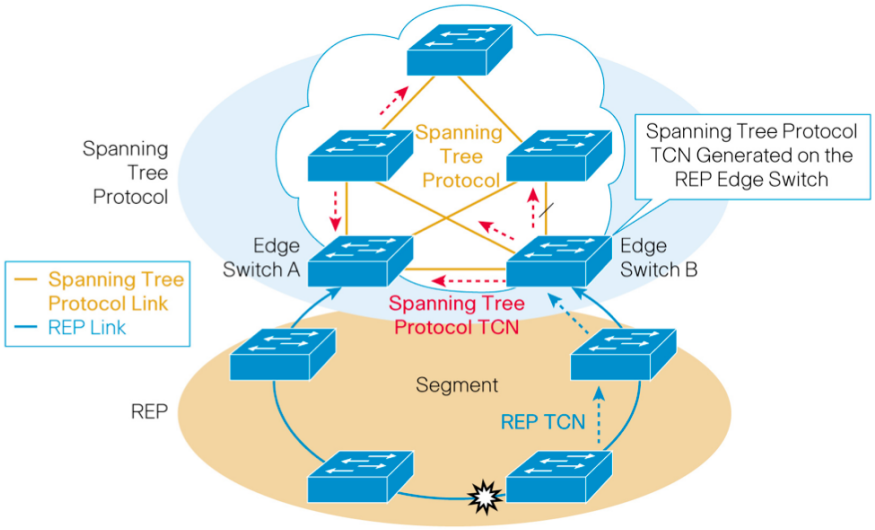
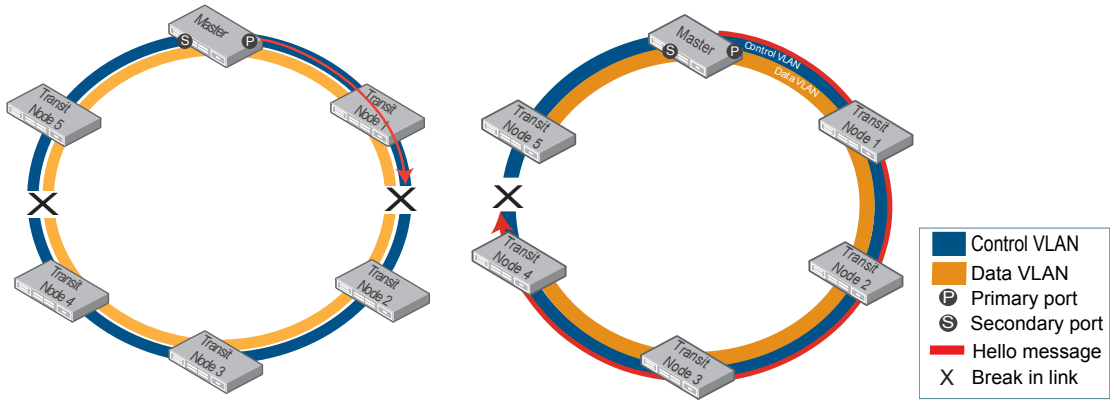


Figure 2.9: Interoperability of REP and STP in case of a link failure [29].

port anymore, or when it receives an unsolicited *Link-down* frame from a *Transit node*. When a link or node failure happens, the Master acts similar to EAPS (see Section 2.3.2). EPSR implements *enhanced recovery* to support the recovery of a link in presence of two failures. Usually, when a link recovers, the adjacent Transit node transitions into the *Pre-forwarding state*, i.e. it blocks the repaired port until it receives a *Ring-up-flush* message from the Master node. When there exists another failure on the ring, the restored port would remain in the Pre-forwarding state without the enhanced recovery.

As shown in Figure 2.10a, two link failures divide the ring in two broadcast domains, *domain1* (Node1, Master-Node, Node5) and *domain2* (Node2, Node3, Node4). Since the Master node is in the *fault-state*, it continues sending Healthcheck messages without receiving them on its secondary port. When only one link recovers and the enhanced recovery is disabled, the Master node would still not receive the Healthcheck messages and thus would not send a Ring-Up-Flush frame. Hence, the newly restored ports would stay in *Pre-forwarding* state and for data traffic, the network would still look as depicted in Figure 2.10a.

When the link between $Node1$ and $Node2$ comes up again having the enhanced recovery enabled, both Transit nodes send a *Link-Forward-Request* to the Master node. Opon reception, the Master sends a special Healthcheck message. If the message does not arrive on the other port within a certain period, the Master node sends a *Permission-Link-Forward*. Either when the Transit nodes receive a Permission-Link-Forward or do not receive it within a given interval, they open the repaired ports for the data VLANs. In the latter case, the Transit nodes deduce that the Master node is not reachable at all. After this enhanced recovery, all nodes are again reachable on the data VLANs, i.e. *domain1* and *domain2* merge to one broadcast domain (see Figure 2.10b).



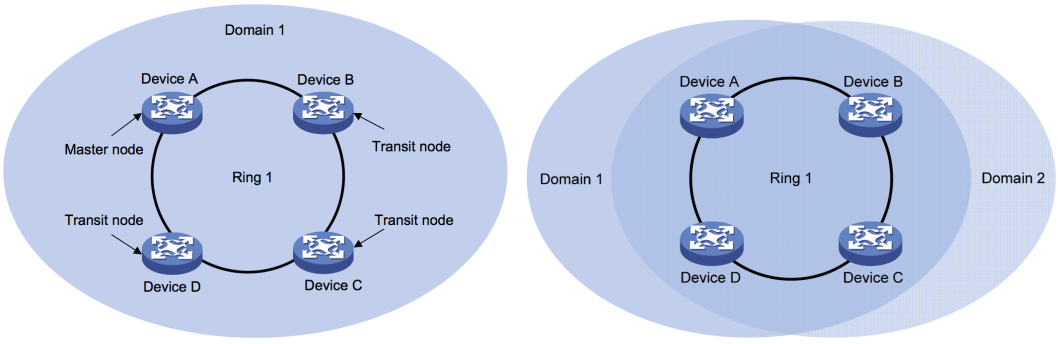
(a) Ring divided into two broadcast domains. (b) One link recovered and enhanced recovery was applied.

Figure 2.10: A ring recovers from one of two link failures by “enhanced recovery” [30].

Like EAPS, the EPSR addresses the problem of possibly emerging “super-loops” as well. Therefore, EPSR assigns a priority to every EPSR ring [30]. In case of a link failure of a common link, the incident Transit nodes send Link down messages only to the Master of the highest priority ring and thus only one secondary port will be opened [30].

Rapid Ring Protection Protocol

RRPP was developed by *H3C* an follows a similar approach than EAPS and EPSR. An *RRPP ring* is defined on an Ethernet switched ring topology and owns a unique ID. An *RRPP domain* can be configured on a single RRPP ring or multiple connected RRPP rings, whereas the configuration depends on the kind of topology [31]. On a single ring, all devices are defined in one domain (see Figure 2.11).



(a) Single-domain (b) Multi-domain

Figure 2.11: One or multiple domains can be configured on a single ring [31]. All devices of the ring are within the domain.

On intersecting rings, all devices are configured in one domain as well and one ring is dedicated as *primary ring* and the others as *subrings* (see Figures 2.12 and 2.14). In this case, the whole primary ring acts like a logical node on the subrings [31]. However, when rings are tangent as depicted in Figure 2.13, each of them needs to be in an own domain [31].

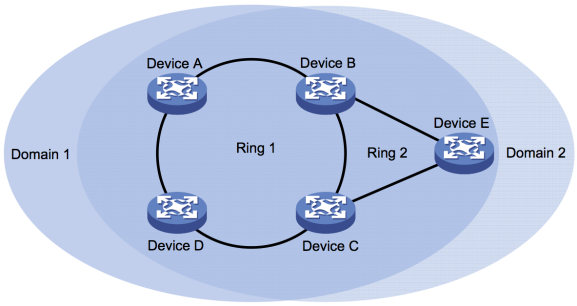


Figure 2.12: Two domains defined on two intersecting rings [31].

Each RRPP ring contains exactly one *master node*. The rest of the nodes are called *transit nodes*. In an RRPP domain, the transit nodes *edge node* and *assistant edge node* are located at the intersection points of the primary ring and subrings. The ports of the edge nodes incident to the primary ring are named *common ports* and those incident to the subring *edge ports*. The *common link* lies between the edge and assistant edge node (see Figure 2.14). The *Sub Ring Packet Tunnel in Major Rings (SRPTs)* of a subring are the two paths on the primary ring connecting the edge and assistant edge node. In the "dual homed" topology example provided in Figure 2.15a, the common link is $S2 - S3$ and the SRPTs of $Ring\ 2$ and $Ring\ 3$ are "S3-S4-S1-S2" and "S3-S2".

Each RRPP domain possesses a *primary control VLAN*, a *secondary control VLAN* and a group of VLANs, which is protected by the domain. Data traffic is handled on the *protected VLANs*. RRPP control frames, called *Rapid Ring Protection Protocol Data Unit (RRPPDU)*, are sent on the control VLANs.

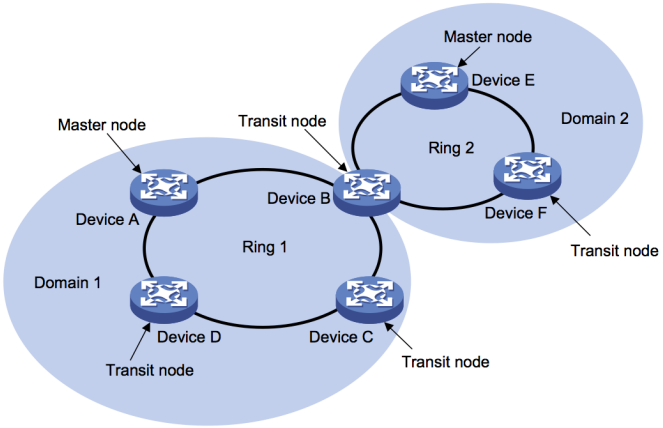


Figure 2.13: A domain is configured for each RRPP ring in a topology of tangent rings [31].

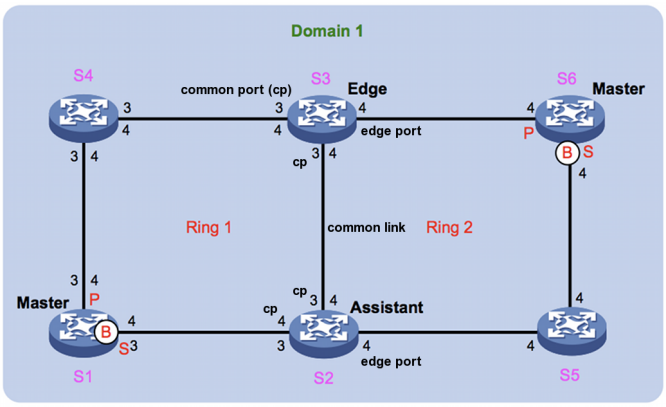
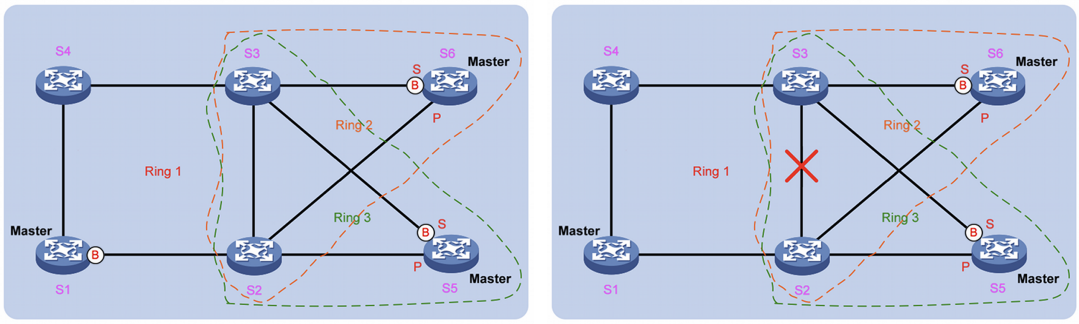


Figure 2.14: One domain defined on two intersecting RRPP rings [31].

An RRPPDU contains, amongst others, a field with the *domain* and *ring ID* it belongs to. Primary ring control frames (RRPPDUs) and *EDGE-Hello* messages of subrings are transported in the primary control VLAN. The ports on the primary ring are parts of the primary and secondary control VLAN, whereas ports on a subring are members of the secondary control VLAN only. In the example depicted in Figure 2.14, both the primary control VLAN 3 and the secondary control VLAN 4 are defined on the ports of the primary ring $Ring$ 1. On the other hand, the ports of subring $Ring$ 2 are only part of the secondary control VLAN.
As in EAPS and EPSR, the integrity of the rings is monitored by a polling and reporting mechanism. When a ring is intact, the master node of the ring blocks its secondary port for all VLANs except its control VLAN. When a link fails, the master opens its secondary port to recovery the ring. However, this mechanism can lead to loops in subrings of "dual homed" topologies and to overcome this problem, the *SRPT State Detection Mechanism* was introduced.
As mentioned above, multiple domains can be configured on single or intersecting rings (see Figure 2.11b respectively 2.12). Thus, load sharing can be achieved since the traffic of both domains chooses different paths when different master nodes are configured [31].

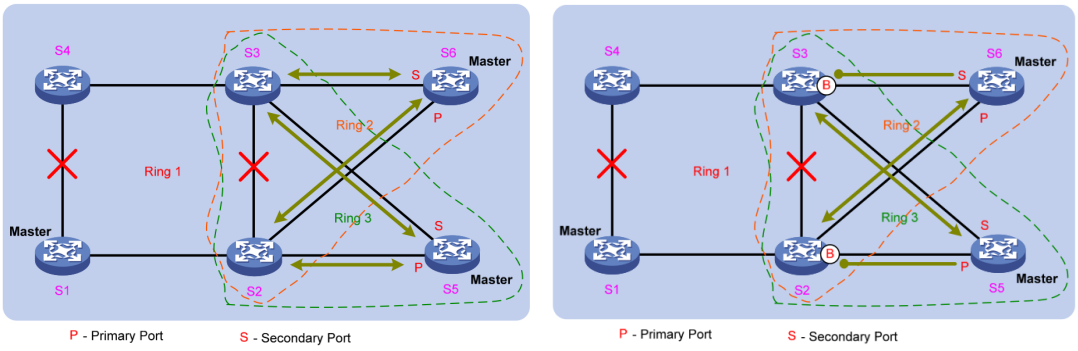
SRPT State Detection Mechanism
As stated previously, all control frames of subrings except EDGE-Hello messages are sent in the secondary control VLAN. In a topology with intersecting rings, protocol packets of a subring are transparently tunneled over one SRPT on the primary ring. Thus, no loops form on the primary ring as those RRPPDUs are sent like data traffic.
In the example illustrated in Figure 2.15a, $Ring$ 1 is the primary ring and $Ring$ 2 and $Ring$ 3 are subrings. When there is no failure present, the primary ring's master blocks its secondary port and thus no loops exist for the subrings' data and RRPPDU messages on the primary ring. But, the subrings' masters receive their *Hello-messages* at their secondary port via the SRPT $S3 - S2$. When there is a failure on the common link, the master $S1$ unblocks its secondary port and the subring Hello-messages reach their master nodes via the second SRPT "S3-S4-S1-S2" (see Figure 2.15b). Hence, the subring masters keep their secondary port blocked in both scenarios.



(a) no failure, both SRPTs are available (b) one failure, SRPT "S3-S4-S1-S2" is available

Figure 2.15: An RRPP domain with three intersecting rings [31].

When both SRPT tunnels go down, both subring masters cannot receive their Hello-messages anymore and unblock their secondary port[18]. This results in a loop as depicted in Figure 2.16a.



(a) without SRPT (b) with SRPT

Figure 2.16: An RRPP domain with three intersecting rings and both SRPTs disabled [31].

Applying the SRPT State Detection Mechanism in this situation, the edge node of a subring blocks the edge port before the secondary port of the master is unblocked [31]. Therefore, the edge node monitors the state of the SRPTs by sending EDGE-Hello messages out of both common ports. The assistant node will receive them on at least one port, if at least one SRPT for the subring is healthy. Otherwise, if the assistant node does not receive EDGE-Hello messages for a given period of time, it detects two failed SRPTs and transmits periodically *MAJOR-Fault* messages through its edge port over the subring to the edge node.

[18]This is only the case, when a master of one subring ignores Hello messages of another subring's master. As both masters send their Hello messages in the same secondary control VLAN, one master could receive the Hello messages of the other one, also when both SRPTs are down. A master could verify the belonging of a Hello message via the domain or ring ID field. However, no further details are provided in [31] regarding this topic.

When the subring is intact, these MAJOR-Fault messages reach the edge node. As long as the edge node receives them, it blocks its edge port to prevent loops in subrings (see Figure 2.16b). As soon as the edge node does not receive MAJOR-Fault messages anymore for a specified interval, it unblocks the edge port (e.g., because of a link failure in the subring). When at least one SRPT recovers, the assistant node receives EDGE-Hello messages again, hence stops sending MAJOR-Fault messages. Assuming that all links on a subring are working, the subring's master receives Hello messages again and thus blocks its secondary port. Additionally, it sends a *Complete-Flush-FDB* out of the primary port. Upon reception of this message, the edge node unblocks its edge port if not done yet. The edge port could be already opened in the case the edge node has not received a MAJOR-Fault message for a given period of time. When a subring link is down when a SRPT recovers, the edge port is unblocked as soon as the edge node does not receive MAJOR-Fault messages for a specified interval. Also, the secondary port of the subring master is kept opened, as the master does not receive its Hello messages. Concluding, after the recovery of one SRPT, either the secondary port of one subring master is blocked and a subring link is broken, or the secondary port of both subring masters are blocked when the subrings are intact.
RRPP defines two different control messages for directing FDB flushes and state transitions. On the one hand, a Complete-Flush-FDB is sent by a master when it switches to the *complete* state, i.e. the ring has recovered. Upon reception, a node flushes its FDB even when it is located on the primary ring and the sender is a subring's master. Additionally, a transit node in the *pre-forwarding* state transitions to the *link-up* state, unless it is member of the primary ring and the sender was a subring's master. To progress from the pre-forwarding to the link-up state, temporarily blocked ports are opened. On the other hand, a *Common-Flush-FDB* is sent by masters when they transition to the failed state [31]. Nodes just flush their FDB when they receive this control message, even when they are located on the primary ring and the control messages origins from a subring's master.
Usually, the edge node of every subring sends EDGE-Hello messages on the primary ring. To reduce the network utilization, subrings can be grouped to a *ring group* and then only the active subring with the smallest ID transmits EDGE-Hello messages.
Furthermore, RRPP can only be combined with STP in coupled ring topologies, where no common ports exist (*tangent mode*), as the port state calculations could come into conflict [31].

Turbo Chain

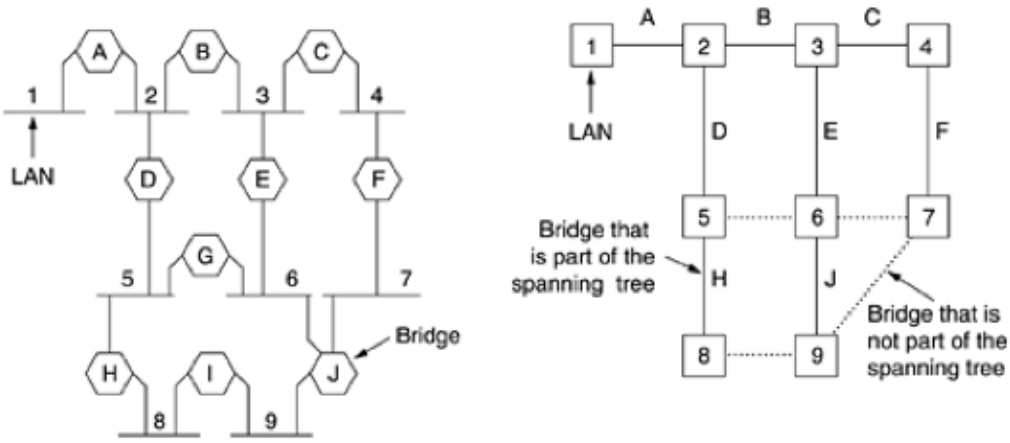
Turbo Chain is a technology implemented by *Moxa*® on their industrial managed Ethernet switches [32]. The system is based on a *chain*, which is a line of connected switches where the first switch is the *Head switch* and the last the *Tail switch*. According to [32], any redundant topologies can be built by simply connecting the chain to an Ethernet network. To avoid loops, the Tail switch blocks the port toward the connected network. In case of a failure in the chain, the Tail switch recovers the network within $20ms$ by opening the blocked port [32]. Turbo chain is compatible with other redundant protocols, i.e. the chain can be linked to networks using for instance STP or Rapid Spanning Tree Protocol (RSTP).

2.3.2 Standardized protocols

The following protocols are either standardized or publicated in a Request For Comments (RFC). The selection of protocols includes the STP, the Rapid Spanning Tree Protocol (RSTP), the Multiple Spanning Tree Protocol (MSTP), the Ring-based Redundancy Protocol (RRP), Ethernet Automatic Protection Switching (EAPS) and the Media Redundancy Protocol (MRP).

Spanning Tree Protocol

STP, specified in IEEE 802.1D, transforms a redundant switched network topology into a spanning tree by blocking distinct ports (see Figure 2.17). Thus, bridging loops are prevented.



(a) Interconnected LANs (b) Spanning tree

Figure 2.17: Interconnected LANs are transformed into a spanning tree by STP [33].

The STP processes several steps to reach this cycle free topology. First of all, the *root bridge* is defined as reference point in the network. Then, all paths from every switch to the root bridge are calculated. When redundant paths are found, STP selects the most efficient path and blocks forwarding on the other paths [34]. However, when a forwarding port goes down, the STP recalculates the spanning tree topology and reactivates a redundant path by opening a blocked port [35]. To perform the STP, switches exchange information about their status using BPDU frames, which are either *Configuration* or *TCN* BPDU frames [34] and always sent to the STP multicast destination address “01:80:C2:00:00:00”. A *Configuration BPDU* is sent for spanning tree computation and the TCN BPDU for notification of topology changes [35]. A BPDU frame contains the field *root BID*, which is the Bridge Identification Number (BID) of the root bridge and comprises of a *Bridge Priority* and the MAC address of the bridge.
The switch with the lowest BID is elected as root bridge. Every switch powering up immediately assumes that it is the root bridge and propagates configuration BPDUs with its own BID as root BID, i.e. the field *sender BID* equals the field root BID. When a switch receives a BPDU with a lower root BID, it stops advertising itself as root and instead forwards the BPDUs of

the superior switch [36]. When the election has converged, all switches agree on one switch as root bridge and henceforth, Configuration BPDUs are only sent by the root bridge every two seconds [35]. The other switches simply update the sender BID and *Root Path Cost* field in the BPDUs and forward the frame. Whenever a new switch powers up, it tries to advertise itself as root bridge by having its own BID in the root BID field in transmitted BPDU frames. The election procedure starts again and thus, root bridge election is an ongoing process [35].
After the selection of the root bridge, every non-root switch in the network determines exactly one *root port*. The root port of a switch is the starting point of the path with the lowest costs from the switch to the root bridge. The costs of a path are the sum of costs of the traversed links. The costs of a link depend on its bandwidth. IEEE defines the default values for costs such that higher bandwidth links have lower costs. The Configuration BPDU contains a value Root Path Cost which stores the cumulative costs along a path [34]. When the root bridge sends out a BPDU, this values is zero. Then, every switch that receives the BPDU adds the cost of the traversed link to the Root Path Cost to get the Root Path Cost of the incoming port. The root port of a switch is the port with the lowest Root Path Cost. This Root Path Cost is also used when the switch sends out the BPDU [35].
The third step for building the spanning tree is to determine the *designated switches* for each network segment. The designated switch is the switch with the lowest Root Path Cost and the only one, which can forward traffic from and to the segment via its *designated port* [37]. Otherwise, bridging loops can arise. Switches know about the Root Path Cost of neighbors on a shared segment from received BPDUs. When a neighbor switch has a lower Root Path Cost value, it owns the designated port [35]. When multiple switches on a segment have the same Root Path Cost, the switch with the lowest BID is selected [37]. All switch ports, which are not root ports, designated ports or ports of the root switch, are in blocking state.
The described steps show how the root bridge, the root and designated ports are determined. However, to prevent the forming of loops, each port in the network progresses through different port states [35]:

- *Disabled*: A port in this state is shut-down due to administration or a failure.
- *Blocking*: After a port comes up, it is in blocking state and thus does not receive and transmit data or learns source MAC addresses. However, blocked ports do receive BPDU frames in order to allow the switch gaining information about its neighbors. Ports are set in blocking state to avoid bridging loops.
- *Listening*: In this state, the port still does not receive and transmit data, but is able to receive, send and process BPDU frames. Thus, it can actively take part in the STP process and is allowed to become a root or designated port. When the port loses its root or designated port status, it transitions back to the blocking state.
- *Learning*: After a period of time known as *Forward Delay*, the port progresses from the listening to the learning state. In addition to receive and send BPDUs, the port is able to learn source MAC addresses. The port is allowed to gather information about the network topology for another period of Forward Delay. Then, it transitions into the forwarding state.

- *Forwarding*: In this state, the port can receive and transmit both BPDU and data frames and learn source MAC addresses.

Furthermore, the following three timers are used by the STP to let the network converge properly [35]:

- *Hello Time*: The root bridge sends Configuration BPDUs in periods of the Hello Time, which defaults to $2s$ in the IEEE standard. Additionally, every switch has a Hello Time stored which is used for transmitting TCN BPDUs.
- *Forward Delay*: A port remains in both the listening and learning states for the period of Forward Delay. IEEE 802.1D specifies a Forward Delay of $15s$.
- *Max (maximum) Age*: This time defines how long a BPDU is saved. When a stored BPDU ages out, the switch notices that the sender of the BPDU has lost connection and a topology change happened. The default value is $20s$.

The root bridge continues sending Configuration BPDUs periodically every $Hello\ Time$ interval and all other switches receive them on their root ports. A switch notices a failed link or device when it does not receive these BPDUs anymore and after the $Max\ Age$ time, it reacts on the topology change [36]. In case of a link failure or recovery, the adjacent switches send TCN BPDUs out of the root port to the root bridge. The TCN frames are sent periodically with the Hello Time interval until the switch receives an acknowledgement from an upstream neighbor [35]. When the root bridge receives the TCN it also sends an acknowledgement and a Configuration BPDU with the *Topology Change Flag* enabled, indicating that a topology change happened. This BPDU instructs all other switches to shorten their FDB aging time to the Forward Delay. Thus, the period of time the switches have FDB entries differing from the new topology can be minimized.

According to [35], the STP timer values are derived from a reference model of a network. This network has a diameter of seven switches, i.e. seven switches are connected in a line from the root bridge to the end of any branch [35]. Furthermore, STP does not necessarily compute the minimum spanning tree. It is possible that the slowest switch is elected as the root bridge [28].

Rapid Spanning Tree Protocol

The RSTP, as specified in IEEE 802.1D-2004 [38], is an advancement of STP which allows faster convergence times of maximum $6s$ compared to $30 - 50s$ in STP [28]. Furthermore, it can react on link failures within a few milliseconds [28].

Multiple Spanning Tree Protocol

MSTP, which is based on RSTP, was published in IEEE 802.1s [39] and integrated into the standard IEEE 802.1Q [21] in 2005 [40]. MSTP allows to define spanning tree instances per VLAN or per group of VLANs. Thus, more links in a LAN are used and load balancing can be achieved [41]. As stated in [42], MSTP improves the fault tolerance of a network since a failure in one instance does not affect the forwarding paths of other instances.

Ring-based Redundancy Protocol

The RRP, described in [43], is a recovery protocol for ring networks built of *RRP devices*. Basically, a RRP device contains an internal hardware full-duplex switch and two *R-ports* which are connected to a line or ring topology. The device is able to control the frame forwarding process. Every device is assigned a *Unique Identifier (UID)* which consists of a two-octet device and a six-octet MAC address. Depending on the position of the device in the network and the network structure, the forwarding possibilities are configured differently. The network topology is either a ring or a line, depending on the state of the RRP network establishment or the presence of link or device failures.
In a ring, the infinite circulation of frames is prevented by two adjacent *Ring Network Managers (RNMs)*. Both RNMs are elected automatically by a RRP procedure, whereas the *Primary Ring Network Manager (RNMP)* is the device with the highest UID and the *Secondary Ring Network Manager (RNMS)* one of its direct neighbors. The dedicated RNMs are configured in a way, that the RNMP does not forward frames to the port toward RNMS and vice versa. Figure 2.18 exemplifies an RRP ring with six nodes and *Device5* and *Device6* being the RNMs. As indicated with two red arrows, one RNM cannot forward frames to the other one.

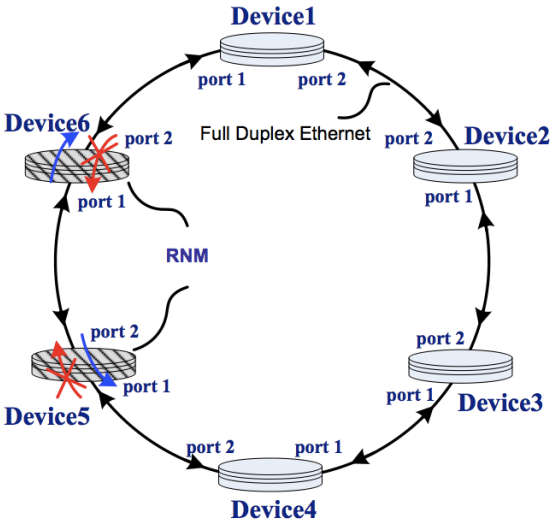


Figure 2.18: RRP ring with six devices and *Device5* and *Device6* defined as RNMs [43].

The device at each end of a line structure is called *Line Network Manager (LNM)* and is disabled to forward frames in both directions. All other devices in a ring or line topology, named *General Devices (GDs)*, relay frames depending on the frames being multicast or broadcast frames or being designated for the device itself or another device. A *Gateway Device (GWD)* is a RRP device with more than three ports. The ports other than the R-ports can be attached to an external Ethernet network. The GWD is able to switch frames between the RRP and the external network using a *dynamic table* in the application layer, which maps RRP to general MAC addresses.

Each RRP device stores a *Network Management Information Base (NMIB)* which contains network information and a path table. Basically, the path table provides the preferred R-port for sending a frame to a destination device. The *preferred R-port* is the port which points to the cheapest path in means of the lowest hop count, or *R-port1* when the hop counts are equal. Is the path starting at the preferred R-port blocked by an RNM (in case of a ring topology), the other R-port is chosen. The devices share link information by sending *Network Control Message Type (NCMT)* messages and so are able to keep their NMIBs up to date. Whenever a link or device failure happens or a new RRP device is added to the network, all devices are informed.

Table 2.2 lists the path table of *Device3* of the ring depicted in Figure 2.18. For example, although both the clockwise and counter clockwise path to *Device6* have hop count two, *Device3* chooses port *R-port2* since the clockwise path is blocked by the RNM node *Device5*. For instance, when *Device1* detects a fault on the link to *Device2* it initiates a topology change from a ring to a line and thus an update of the NMIBs by informing the other devices. Additionally, *Device1* and *Device2* become LNM devices. Also the RNMs recognize the change and transform to GDs opening the link connecting them for frame forwarding.

R-port	**Destination**				
	Device1	**Device2**	**Device4**	**Device5**	**Device6**
R-port1	3 hops	4 hops	0 hop	1 hop	2 hops
R-port2	1 hop	0 hop	4 hops	3 hops	2 hops
Preferred Port	port 2	port 2	port 1	port 1	Don't care
Destination Port	port 2	port 2	port 1	port 1	port 2

Table 2.2: Path table of *Device3* of the example depicted in Figure 2.18 [43].

Ethernet Automatic Protection Switching

EAPS, a proprietary protocol of *Extreme networks*^{TM} [19], was published under RFC 3619 [44]. An EAPS domain is defined on a single ring and is constructed to protect a set of VLANs (*protected VLANs*) [45]. This group of VLANs including a unique *control-VLAN* is specified by EAPS and has to be configured an all ring ports. The protected VLANs carry data traffic and the control-VLAN is used for transmitting EAPS control frames. Additionally, one node is defined as *master-node* and the others are referred to as *transit-nodes*. An EAPS enabled switch can be member of multiple EAPS domains when it implements multiple instances of the EAPS protocol machine.

Furthermore, multiple domains, having unique control-VLANs and own sets of protected VLANs, can coexist on one ring, which makes spatial reuse of the network's bandwidth possible [45]. The master-node blocks traffic of the protected VLANs on one of its ports, the *secondary port*, to ensure a cycle free environment on an intact ring.

[19] `www.extremenetworks.com`

On the other hand, this port is left opened for the control-VLAN and thus all control frames of the corresponding EAPS domain can be switched over the whole ring. To prevent the control frames from circulation, the master-node does not forward them.

EAPS follows two mechanisms to detect ring failures. First, using the *alert mechanism*, transit-nodes send a *Link-down* EAPS control frame to the master-node whenever they detect a link failure. Second, a polling mechanism is implemented to handle a possible loss of the Link-down frame. With this "backup" facility, the master-node periodically sends *Health-check* frames out of its primary port on the control-VLAN. The master-node is invoked to transition into the *ring-fault* state when it does not receive the Health-check frames for a configurable period of time or receives a Link-down frame. Then it unblocks its secondary port for the protected VLANs, flushes its FDB and sends a *Ring-down-flush-fdb* frame to the transit-nodes directing them to flush their FDBs as well [44]. Henceforward the nodes learn about the new topology by switching frames. The master-node continues sending Health-check frames when it is in the fault-state. Once the ring is restored, it again receives a Health-check frame on the secondary port. Then it detects the restoration of the ring and switches to the *normal* state. This means that it blocks the secondary port for non-control frames, flushes its FDB and sends *Ring-up-flush-fdb* to the transit-nodes instructing them to flush their FDBs. During the interval between the link recovers and the master-node detects the operational state of the ring, the transit-nodes incident to the repaired link remain in the *preforwarding-state*. In this state, the transit-nodes block all the protected VLANs on the restored port to avoid temporary loops as the masters' secondary port is still opened. Only after the transit-node receives the order to flush, it unblocks the repaired port.

Having a common link between two EAPS protected rings and VLANs spanning over these domains, "super-loops" arise when the common link fails. To overcome this problem, the protocol *EAPS Shared-Ports* was introduced where the nodes Controller and Partner prevent the topology from becoming a loop when the common link goes down [46].

Moreover, false switch-overs to the fault-state can happen when the master-node does not receive Healthcheck frames for other reasons than link failures, e.g. due to incorrect configuration, dropping of frames or increased delays because of exceedingly high network utilization. In [46], a method to reduce the number of these "false failures" was introduced. Thereby, the master-node verifies a possible link failure by sending a *Query-link-status* frame out of both ports. The transit-nodes with a broken link will then answer with a Link-down frame and the failure is detected with the alert mechanism.

Media Redundancy Protocol

The MRP, as specified in [47], is a recovery protocol which reacts deterministically on a single failure — either a link or a switch failure — in a switched ring topology. The *redundancy domain*, i.e. a ring, is identified with a unique *domain ID*. If nodes are part of multiple MRP rings, multiple instances of the MRP protocol machine are necessary where every domain gets a unique ID and exactly two unique *MRP ports*. Otherwise, the MRP frames of one ring, sent

with specified multicast addresses[20], would interfere with other MRP rings. According to [47], MRP is located above the DLL of each MRP node. One node of the ring is configured as *Media Redundancy Manager (MRM)* and the others are defined as *Media Redundancy Clients (MRCs)*. In the *ring-closed* state, i.e. all links and switches are working properly, the MRM blocks one of its MRP ring ports (see Figure 2.19). Only Management (MGMT) frames are allowed to enter and exit a blocked port of a switch, all other frames are discarded. This prevents non MGMT frames from circulation as the ring transforms into a line topology.

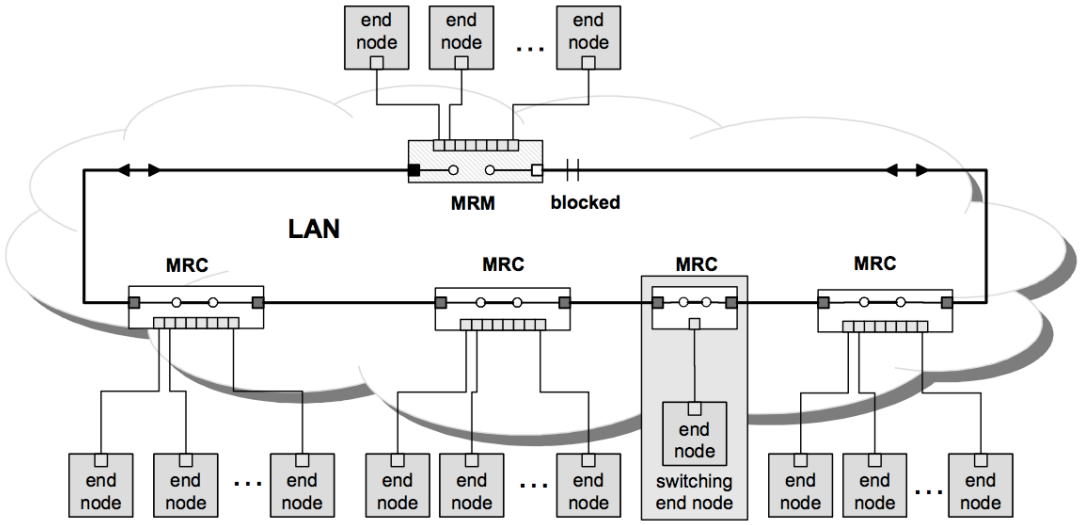


Figure 2.19: An MRP ring in ring-closed state as no failure is present. The MRM blocks one of its MRP ports [47].

In order to check the integrity of the ring, the MRM continuously sends MRP test frames as MGMT frames out of both ring ports. These *MRP test frames* are simply forwarded by the MRCs on the ring. The MRM detects a failure when it does not receive MRP test frames on either of its ports for a certain period of time. Then, the MRM switches over into the *ring-open* state by setting the blocked port to forwarding and sending multiple *MRP topology change* frames. Additionally, at the time an MRC detects a link failure, it automatically sets the corresponding port to blocking. This prevents the topology from becoming a ring for data frames immediately after the link recovers. After a particular delay, communicated by the topology change frames, all MRCs and the MRM flush their FDB. Then the convergence process is completed and all nodes successively rebuild their FDB according to the new topology through switching of data frames.

In the ring-open state, the MRM continues sending MRP test frames and as soon it again receives a test frame, the ring is recovered from the failure. Then the MRM introduces a switch-over to the *ring-closed* state by transmitting topology change frames.

[20]MRP test frames are sent with the destination MAC address “01:15:4E:00:00:01” and topology and link change frames with “01:15:4E:00:00:02”.

After the delay indicated in these frames, all MRP nodes flush their FDB, the MRM blocks one of its ports and the MRCs affected by the link failure set their ports to forwarding. To speed up the failure or ring integrity detection, involved MRCs send *Link up* and *Link down* frames when a link goes up or down, respectively. Receiving either of them, the MRM reduces the interval of sending MRP test frames. Optionally, the MRM can be configured to immediately introduce a switch-over on the reception of a Link up or Link down frame.

After discussing various protocols, MRP is chosen for the implementation of the communication system, since it is an efficient, robust and deterministic protocol.

CHAPTER 3

Implementation

This chapter deals with the model implementation. As explained in Section 2.1.4, during this process, the formal model of the system is built into an executable simulation program.

3.1 General

The aim of this work is to examine the performance of a fault-tolerant communication system connecting *communication nodes (CNs)*. Thus, the objects of interest are mainly messages exchanged by the CNs, which are influenced by the switching behavior of the CNs and the links. The fault tolerance is based on redundancy introduced by a ring topology. This ring redundancy involves the necessity of ring protection and recovery mechanisms. In this work, the MRP was chosen for this purpose.

According to Section 2.1.3, the investigated communication system is a dynamic and discrete system, as its state changes over time at discrete points in time, e.g., when data exchange or a failure happens or even when address tables are aging. Besides that, it is a non-terminating system as it runs permanently. The derived logical model and the simulations are stochastically, as failures and message transmissions can happen at arbitrary point in times. In contrast, the resulting simulations are non-real-time, as they are neither simulated in real-time, nor connected to a real system and do not use real data. For building the simulation program, the phase *model building and data collection* of the process flow model described in Section 2.1.4 was iterated multiple times. Model verification and validation was performed by means of *exploratory testing* [48].

The simulation model was built using OMNeT++[1], an open-source environment for discrete event simulation, in combination with the communication networks framework INET. Although INET provides different models for protocols of the ISO OSI layers (see Section 2.1.6), several modules, which are necessary for this thesis, had to be implemented.

[1] http://www.omnetpp.org/

OMNeT++ is an extensible and modular simulation library and framework [8]. In OMNeT++, *simple modules* can be encapsulated to *compound modules* using the Network Description (NED) language, whereas the number of hierarchy levels is not limited [8]. Also, virtual networks are built in this way (see Figure 3.1).
The behavior of the simple modules is implemented in C++ and each module has *parameters*, which are used to configure the module. The parameters can be assigned with default values or volatile with a random number drawn from a statistical distribution [8].
The modules communicate by message passing, either via *gates* or by sending messages directly to the destination module. Two gates can be linked with a *connection*, which may model properties like propagation delay, data rate and bit error rate. Protocol stacks are usually implemented as modules which exchange messages (e.g., the TCP module with the IP module) [8].
Different simulation experiments can be created by overriding the default values of the module parameters with settings in `*.ini` files. Additionally, the number of runs, the seed set and desired output can be configured via this file.

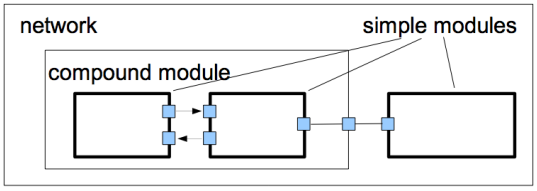


Figure 3.1: OMNeT++ follows a modular approach [49].

3.2 Communication system modeling

Fundamentally, a CN is a network node composed of a Microcontroller Unit (MCU) and a switch. The traffic producing application is executed on the MCU and the switch is responsible for relaying network traffic. The CNs can be logically arranged in a hierarchical topology, where a *super CN* is hierarchically superior to an *inferior CN*. The ring and recovery protocol mechanism, in particular MRP, runs on the MCU, too.

The model of a CN is built of two compound modules, an MCU module and a switch module which consist of both (modified) simple modules provided by the INET framework and implemented modules (see Figure 3.2). A formal model for the MRP is provided in form of the MRP standard [47]. The switch module follows the datasheet of a real, VLAN capable switch [50].

3.2.1 The Microcontroller Unit model

The model of the compound module MCU is depicted in Figure 3.3. The functions of its modules, located at layer 2 and layer 3 of the ISO OSI reference model, are defined as follows:

- `mac`: This module on layer 2 represents an Ethernet interface according to the IEEE 802.3 standard, involving full-duplex and half-duplex transmission of Ethernet frames. It simulates the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol,

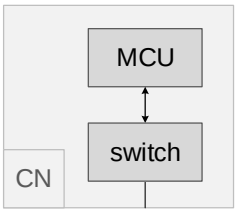
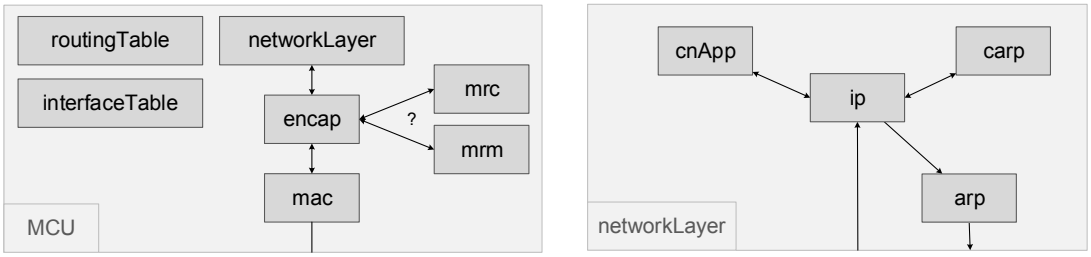


Figure 3.2: The model of a CN consists of an MCU and a switch module.



(a) Compound module MCU

(b) Compound module networkLayer

Figure 3.3: The compound module of the MCU contains the compound module *networkLayer*.

and the backoff algorithm[2] for half-duplex communication. It is also able to handle Ethernet PAUSE frames and to connect and disconnect the interface dynamically from the communication channel, for example when triggered by the `ScenarioManager` during runtime. Additionally, a redundant channel and listeners for the state of the channel (either redundant or non-redundant) and the connection state of the interface were implemented. As specified in IEEE 802.3, a frame is discarded after a certain number of retransmission attempts in half-duplex mode[3]. The modified module needs to allow to disable this function.

- The `encap` module is responsible for Ethernet frame padding, encapsulation and decapsulation. In addition, it performs VLAN-tagging and dispatches MRP frames directly to the MRP module. In case of a simulated MCU failure, it discards every frame except MRP frames.

- A CN includes either an `mrm` or an `mrc` module which are implementing the corresponding functionality according to the MRP standard [47]. In order that MRP frames can pass through a blocked switch port, they are defined as MGMT frames by locking the MRP multicast address, including a destination port vector and priority, into the switch's address table. So, incoming MRP frames are always relayed to the MCU port.

[2] IEEE 802.3, 4.2.3.2.5 Collision backoff and retransmission.

[3] According to IEEE 802.3, 4.4.2 MAC parameters, the maximum number of attempts is 16.

For outgoing frames, the MCU uses the trailer mode to supersede the switch's logic, i.e. it sets the destination port vector and a priority for the frame and disables source address learning. Moreover, the `mrm` module does not send MRP frames with its own MAC address, since the `mrc`'s switches would toggle the destination port vector of the `mrm`'s address in the address table, as MRP frames pass the ring in two directions[4]. Both the `mrm` and `mrc` module register a listener on the connection state of all `mac` interfaces.

- The `interfaceTable` holds, additionally to a loopback interface, information about all `mac` interfaces which register dynamically at the start of the simulation. Amongst others, the data include a unique identifier, the MAC address, the Maximum Transmission Unit (MTU) value, the current state (up or down) and flags indicating whether the interface supports broadcast and multicast. It also provides management functions for manipulating these data[5]. Protocol specific IPv4 data are added by the `ipv4NetworkConfigurator`, the `cnApp` and `carp` module when they are creating subinterfaces. With the modified version, it is also possible to create VLAN devices by assigning a VLAN ID and a VLAN QoS tag (the priority) to a virtual device. The actual VLAN-tagging of frames happens in the module `encap`.

- The module `routingTable` represents the IP routing table. When a device is added to or deleted from the `interfaceTable`, it automatically updates the routing table by appending or removing the netmask route ("on-connection route"). Basically, routes can be added or removed by the `ipv4NetworkConfigurator` before the start of the simulation or via `cnApp` scenario during runtime. IP forwarding can be enabled and disabled with the flag `IPForward` and has to be enabled on CNs with routing functionality.

- `arp`: The implemented version of the ARP module is a simplification of the algorithm used in the Linux kernel 2.2+ module[6]. The INET framework on the other hand implements ARP according to [52]. Basically, the number of retries per resolution attempt, the time interval between two consecutive retries and the ARP cache timeout are configurable. The INET model always sends ARP requests as broadcasts. The ARP cache is updated by received ARP packets only and entries which are not updated for the timeout period are removed from the cache. On the contrary, the implemented version sends broadcast as well as unicast ARP requests and uses a randomly chosen cache table timeout time. This timeout is chosen from the interval $[0.5 * base\_reachable\_time, 1.5 * base\_reachable\_time]$. Also, ARP table entries are not deleted immediately when they time out, but considered being stale. An ARP resolution of a timed out *(stale)* entry is done by unicast requests after a certain delay[7].

[4] According to [47], the port MAC address is used for the MRP Data Link Layer Protocol Data Unit (DLPDU)

[5] Basically, the class offers functions for manipulating the interfaceTable similar to the linux command ifconfig (`http://linux.die.net/man/8/ifconfig`).

[6] As stated on `http://linux.die.net/man/7/arp`, the kernel module uses algorithms of RFC 2461 (Neighbor Discovery for IPv6) [51] when applicable for IPv4 ARP.

[7] This delay is called DELAY_FIRST_PROBE and defaults to $5s$ in the linux module.

Additionally, it supports *upper-layer reachability confirmation*, i.e. higher level protocols can confirm an entry for a neighbor when the connection makes *forward progress*. This means, that "the packets received from a remote peer can only be arriving, if recent packets sent to that peer are actually reaching it" [51].

- The `ip` module implements the IPv4 protocol and thus is responsible for routing of datagrams using the `interfaceTable` and `routingTable`. However, IPv4 address assignment is done by the `ipv4NetworkConfigurator`. The modified module is also able to handle frames entering a subinterface by assigning the correct `InterfaceEntry` of the `InterfaceTable` to the frame.

- In the context of this work, just the network utilization caused by Common Address Redundancy Protocol (CARP) is relevant. Hence, the module `carp` only transmits CARP advertisements to the CARP multicast address[8] and does not implement the fail-over logic of the CARP masters. Instead, the switch-over from the master to the backup is carried out by providing multiple Extensible Markup Language (XML) elements for the `ScenarioManager`.
  To successfully set up CARP, it is required to add the corresponding interfaces of the selected hosts to the multicast group. This can be done by the following element:

```
<multicast-group hosts="cn3.cpu␣cn4.cpu" interfaces="mac"
address="224.0.0.18" />
```

  Then, for example, `cn3` can be selected as master by:

```
<!-- adding subinterface with virtual IP address 192.168.1.14 -->
<addDevice t="0.015s" module="cn3.cpu.networkLayer.carp" name="carp0"
    address="192.168.1.14" netmask="255.255.255.240" mtu="1500"
    multicast="true" broadcast="true" />
<setIPForward t="0.015s" module="cn3.cpu.networkLayer.cnApp"
    value="true" />
<!-- start sending CARP advertisements every 2s and send a gratuitous
    ARP message -->
<startCARPadvert t="0.016s" module="cn3.cpu.networkLayer.carp"
    advertFreq="2" sendGratARP="true" interface="carp0" />
```

  The last element starts the sending of advertisements with a period of $2s$. To fail-over from `cn3` to `cn4`, the following steps are necessary:

```
<!-- stop cn3 from being master -->
<stopCARPadvert t="7.4s" module="cn3.cpu.networkLayer.carp" />
<delDevice t="7.4s" module="cn3.cpu.networkLayer.carp" name="carp0" />
<setIPForward t="7.4s" module="cn3.cpu.networkLayer.cnApp"
    value="false" />
<!-- and switch-over to backup cn4 -->
<addDevice t="7.5s" module="cn4.cpu.networkLayer.carp" name="carp0"
    address="192.168.1.14" netmask="255.255.255.240" mtu="1500"
    multicast="true" broadcast="true" />
```

[8] UCARP 1.5.2 uses "224.0.0.18" as default, see `http://www.pureftpd.org/project/ucarp`

```
<setIPForward t="7.5s" module="cn4.cpu.networkLayer.cnApp" value="true"
    />
<startCARPadvert t="7.6s" module="cn4.cpu.networkLayer.carp"
    advertFreq="2" sendGratARP="true" interface="carp0" />
```

It is also possible to add VLAN interfaces by complementing the element `addDevice` with the attributes `vlanid` and `qos`. To use another than the local MAC address for the interface, the attribute `mac` can be set.

- `cnApp`: The three main features of this module are CN configuration, traffic generation, and recording of statistical data. If the parameters are provided by the `*.ini` file, the `cnApp` adds one or two (VLAN) device(s) to the `interfaceTable` of the MCU. This is for example necessary, when the CN provides router functionality. For traffic generation and recording, the CN first retrieves a list of its superior and inferior CNs from the module `cnAddressTable`. Then, the traffic generation is defined per set of superior and inferior CNs of the sending node.

  For data communication among CNs, the following message classes are defined:

  - `DataPeriodic`: is periodically sent with the configured priority and size
  - `DataSporadic`: is sent with the configured priority when triggered by scenarios

  The nature of the workload (sending of `DataPeriodic` messages) generated by a CN can be defined in the `cnAppBase.ned` or `*.ini` file. It is possible to separately define the priority[9], the packet size, the send interval, the start and stop time of message generation[10] for communication with all super CNs, inferior CNs or explicitly defined receivers. By defining explicit receivers, it is possible to produce higher traffic between a CN and any other CN, regardless of their logical relationship. The transmission of other types' messages is triggered by scenarios.

  Upon message reception, the `cnApp` module performs upper-layer reachability confirmation (ULC) of the source IPv4 address. Usually, this is done by a protocol at least on layer 4 (w.r.t. to the ISO/OSI reference model), such as TCP. However, this model of the MCU does only consider layer 2–3 and thus the process is simulated by the `cnApp`. Furthermore, it records the delay (arrival time - send time) for all kind of messages separately for each sender. It also counts the number of sent/received messages to/from all other CNs.

  The module `cnApp` is scriptable via the `ScenarioManager`. It is able to add or delete VLAN devices to or from the `interfaceTable` and routes to or from the `routingTable`. Also, the `IPForward` flag can be set. To trigger the sending of sporadic data or a burst of messages this module processes the following XML element (example):

```
<sporadic t="10s" module="cn1.cpu.networkLayer.cnApp"
timespan="2s" number="10" priority="3" super="1" inf="0" />
```

[9] The IPv4 Type of Service (TOS) field values 28, 92, 156 and 220 correspond to the switch priorities 0, 1, 2 and 3 respectively.

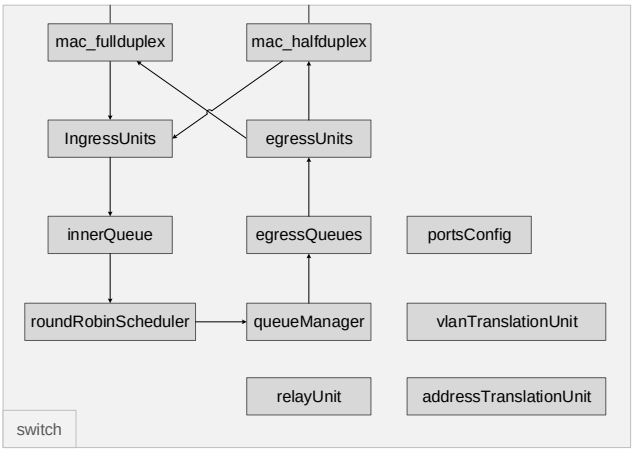
[10] The generation of messages is automatically stopped 3s before simulation end to allow the system to fade out.

This results in a burst of `DataSporadic` messages, sent uniformly distributed between time 10s and 12s to all of `cn1`'s superior CNs.

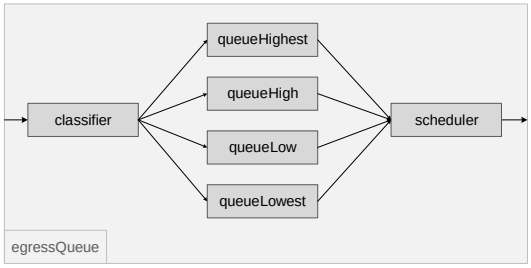
Another scenario is controlled via MAC state listeners. The changing of a switch's `mac` interface from connected to disconnected or of an adjacent channel's state from redundant to non-redundant, is notified by sending of `CNLinkFailure` messages to all superior CNs.

3.2.2 The switch model

Figure 3.4 shows the simulation model of the switch. The functionalities of the modules are specified as follows:



(a) Compound module of the switch



(b) Compound module of egressQueue

Figure 3.4: The compound module of the switch contains a compound module *egressQueue* for each port.

- `mac`: The switch offers `mac` interfaces in both half- and full-duplex mode. The RS485 links are emulated by half-duplex connections.

- There is one `IngressUnit` module per port. This modules decides if the frame is allowed to enter the switch, based on the port state and whether the frame is a MGMT frame or not. The implemented switch supports the port states *disabled*, *blocking/listening*, *learning* and *forwarding*. In this implementation, a frame is considered as MGMT frame when its destination address is locked in the switch's address table, i.e. the address is not flushed from the address table when aging is performed. Furthermore, low-priority frames are discarded when no ingress buffer is available. The switch guarantees forwarding of high-priority frames by memory reservation. It is not exactly stated in the manual which priorities are always relayed and how much memory is reserved to do so. Thus, the ingress module always forwards frames with priority three and two (if allowed by the port state). The used "reserved" memory is recorded for statistics.
  When the port state is learning, the module `IngressUnit` invokes MAC address learning, which can be disabled with a flag in a frame trailer (e.g., applied to MRP frames). Additionally, this module performs QoS classification for the `QueueManager` by mapping of one of the following values to one of four switch priorities in order by:

  1. the value provided by the frame trailer (if available),
  2. the destination address' priority in the address database (if the address is stored),
  3. the IEEE 802.1Q priority tag[11],
  4. the IPv4 TOS field or
  5. the port's default priority

  To prefer the IPv4 TOS field to the IEEE 802.1Q priority tag, the flag `tagIfBoth` can be set. Although this module extracts the VID from the frame and/or the `portsConfig`, the module `QueueManager` prevents frames from entering or leaving a port based on VLAN conditions. Using the default port VID instead of the extracted VID can be forced by setting the `forceDefaultVid` flag. The `ingressUnit` also performs untagging of VLAN (double) tagged frames.

- The `innerQueue` implements a First In First Out (FIFO) queue with unlimited buffer space. The available memory on the switch is controlled by the `QueueManager` and not restricted by this module.

- The module `roundRobinScheduler` implements a Weighted Round-Robin (WRR) scheduler which serves the queues in a cyclic manner. This modification of the provided WRR scheduler also considers messages simultaneously arriving at the input queues, as described below.

- The module `queueManager` requests frames from the `roundRobinScheduler` and processes them in a serial manner. The processing time of a real switch is simulated by this module and can be configured with the parameter `processingTime`.

[11] Called `user_priority` in IEEE 802.1Q-2003, Priority Code Point from IEEE 802.1Q-2005 on.

To process a frame, the `QueueManager` retrieves the destination port vector of the frame from the `relayUnit` and forwards it to the relevant `egressQueues` or discards it when no ports were selected, e.g. when the frame is filtered because of the VLAN configuration.
Additionally, the `queueManager` is responsible for buffer management. Hereby, it assigns ingress buffer to the `ingressUnits` and works as a multicast handler, i.e. it frees the buffer memory used by a multicast frame not before the frame has left every port. It also records statistics about memory consumption.

- The `relayUnit` module implements the switching logic of the switch. It resolves the output port(s) for an incoming frame based on its destination MAC address considering VLAN configurations, a possibly available frame trailer and the frame being a MGMT frame or not. The VLAN mode can either be port-based VLAN or a 802.1Q VLAN. The latter supports the three different options *secure*, *check* and *fallback*, which differ in level of security. Using a frame trailer, the output port(s) found by the `relayUnit` can be overruled. So, the MCU is able to supersede the switch's logic. MGMT frames are, ignoring any VLAN configurations, always forwarded to the ports indicated in the `addressTranslationUnit` address database.
  The `relayUnit` also offers switch configuration possibilities. Basically, per-port default settings are defined in `Ingress.ned` and can be modified by any `*.ini` file when starting a simulation. Per-port settings include the port state, the ingress and egress mode, the VLAN mode, the VLAN-table for port-based VLANs and so on. However, the resulting configuration can be overruled by an XML configuration file which is passed to the `relayUnit`. Furthermore, up to 64 VLANs and MAC addresses (including the entry state and priority) can be added to the `vlanTranslationUnit` respectively `addressTranslationUnit` database via the configuration file. The `relayUnit` is able to load a new switch configuration file during runtime via the following element (example):

```
<loadConfig t="6s" module="cn4.switch.relayUnit"
    file="SwitchConfig2.xml" />
```

- The switch comprises one `egressQueue` per port. This compound module is depicted in Figure 3.4b and consists of a `classifier`, one passive FIFO queue for each switch priority and a `scheduler`. The `classifier` sorts frames based on the priority into the corresponding queue. The `scheduler` selects the next frame which can egress the port based on a fixed-priority or weighted-fair scheme. With the fixed-priority scheme, all higher priority frames are scheduled before any lower-priority frames are selected. Using this mode, it is possible that lower-priority frames are never chosen for transmitting (starvation) and thus the weighted-fair scheme is enabled.

- There is one `egressUnit` per port. It receives frames from the `egressQueue` and resolves if they are allowed to exit the port. The decision is based on the port state and whether a frame is a MGMT frame or not. How a frame egresses — either unmodified, untagged or tagged — depends on the egress mode of the port (in case of port-based

VLAN) or the *membertags* in the VLAN database (in case of IEEE 802.1Q VLANs). The `egressUnit` then forwards the frame to the `mac` module for transmitting.

- Basically, the module `addressTranslationUnit` represents the address database of a switch. It contains an address table and functions for manipulating them. For each learned MAC address there exists a table entry with the corresponding destination port vector, the entry state[12] and the priority. MAC addresses can be learned from the source addresses of incoming frames or via `relayUnit` and a configuration file. The learning procedure can be disabled via `learningDisabled` flag. The address table is also subject to aging, i.e. entries with a certain age are purged from the table, if they are not locked. The maximum age of an entry is per default 304s. Of course, an address' entry is refreshed when it again appears as a frame's source address or is updated when endstations move. When the address table is full and the switch tries to learn a new address, entries are deleted according to a Least Recently Used (LRU) algorithm. Besides learning, the other main feature of this module is MAC address resolving. This function is used by the `relayUnit` to retrieve the destination port vector according to a destination MAC address.

- The module `vlanTranslationUnit` holds the database for dynamic VLANs, i.e. a table where each entry consists of a VID and a membertag for every port. A membertag describes if the port is a member of the VLAN and how frames should egress the port. Both the `relayUnit` and the `egressUnit` obtain VLAN information for VLAN filtering respectively frame modification from this module.

- The control data of every port are contained in the module `portsConfig`. The data include the port state, the ingress, egress and the VLAN mode, the default priority and VID. Additionally, this module provides functions to modify and access this information and stores a static VLAN table per port used for port-based VLAN. Moreover, other modules can register a listener on the port state of all ports, in order to be notified when a change happens.

3.2.3 Additional modules

To complete a simulation set up, the following modules have to be complemented to a network:

- `cnAddressTable`: This module represents a central site containing the address and logical relationships information of all existing CNs in a simulation. Therefore, it parses a list of CNs including their names and assigned IPv4 addresses. In addition, it retrieves the logical relationships among the CNs from a file. This input file can be generated by a Perl script (`IDIPMatching.pl`) which was developed in the context of this thesis. The output of this script has the following structure:

[12] The age of an entry can be deduced from the entry state.

```
# ADDRESSES
<CN 0 name> <ipv4 address 0>
<CN 1 name> <ipv4 address 1>
<CN 2 name> <ipv4 address 2>
...
# inf CNs (x:y means x is a inferior CN of y)
<CN 0 name>:<CN 2 name>
<CN 1 name>:<CN 2 name>
<CN 2 name>:
...
```

Each `cnApp` is able to retrieve a list of its superior and inferior CNs from this module.

- `ipv4NetworkConfigurator`: This module is responsible for the IP address assignment and the set up of routing tables. According to the provided configuration file, it allows to assign addresses to all interfaces of all nodes. This can be done either manually, automatically or even in a combined way, considering that only modules with the `node` property are treated as nodes in the topology.
  The `ipv4NetworkConfigurator` is able to set up hierarchical networks by using wildcards, e.g. by the following configuration[13]:

```
<config>
   <interface hosts="area11.lan1.*" address="10.11.1.x"
  netmask="255.255.255.x"/>
  <interface hosts="area11.lan2.*" address="10.11.2.x"
      netmask="255.255.255.x"/>
   <interface hosts="area12.lan1.*" address="10.12.1.x"
  netmask="255.255.255.x"/>
  <interface hosts="area12.lan2.*" address="10.12.2.x"
      netmask="255.255.255.x"/>
  <interface hosts="area*.router*" address="10.x.x.x"
      netmask="x.x.x.x"/>
  <interface hosts="*" address="10.x.x.x" netmask="255.x.x.0"/>
</config>
```

  On the other hand, the `ipv4NetworkConfigurator` can be forced to use an address by restricting every attribute, e.g.:

```
<interface hosts="cn0.cpu" address="192.168.1.1"
    netmask="255.255.255.240" />
```

  The second major task of this module is the configuration of IPv4 routes. Again, static routes can either be set up by providing them in the configuration file or automatically. For instance, setting up a default gateway can be done by

```
<route hosts="cn0.cpu" destination="*" netmask="*"
    gateway="192.168.1.14"
interface="mac" metric="0" />
```

  To execute all its functions, the `ipv4NetworkConfigurator` accesses information of the `interfaceTable` and `routingTable` modules of the particular nodes.

[13] see `IPv4NetworkConfigurator.ned` of INET 2.0

- `scenarioManager`: Using this module, various scenarios can be simulated by specifying them in XML. This feature makes the simulation environment very flexible, as the scenarios can be modified easily. Also, it increases the reusability, as the scenarios can be applied on various networks. Therefore, it processes XML elements and either executes them by itself or by calling the `processCommand()` function of a module implementing the `IScriptable` interface. The `scenarioManager` is able to connect or disconnect a channel to or from a module, set an arbitrary parameter of a module and an attribute of a channel. In the example below, the datarate of an existing channel, connected to the first `mac` interface of `cn0`, is set to 10Mbps at an arbitrary time in [3s, 3.7s]:

  ```
  <set-channel-attr t="3s" tend="3.7s" src-module="cn0"
      src-gate="ethBMZA$o[0]" attr="datarate" value="10Mbps" both="true"
      />
  ```

  By providing the following element, a channel will be disconnected from `cn4` at the time 5s and re-connected at 8.8s:

  ```
  <disconnect t="5s" src-module="cn4" src-gate="ethBMZA[1]" />
  <connect t="8.8s" src-module="cn4" src-gate="ethBMZA[1]"
          dest-module="cn3" dest-gate="ethBMZA[1]"
          channel-type="ned.DatarateChannel" >
          <param name="datarate" value="640kbps" />
          <param name="delay" value="0.0000005s" />
  </connect>
  ```

  The modules `CARP`, `relayUnit` and `cnApp` are implementing the `IScriptable` interface.

- `throughputMeteringChannel`: This module is an extension of the simple module `DatarateChannel` and can be used on desired links. It represents a channel which is able to measure and record *current* and *average* parameters. The current values are calculated per configured interval whereas the average values are accumulated values divided by the elapsed simulation time. Thus, the current values are recorded periodically and the average values whenever a packet is transmitted. The parameters include the current and average number of packets per second and the overall number of transmitted packets at the end of the simulation. Another parameter is the average bandwidth in bits per second, which is the average number of bits transferred until a certain point in time. A further important parameter is the *current channel utilization* in %, which describes the current number of bits per second divided by the bandwidth capacity of the link. Moreover, each indicator is separately calculated and recorded for EtherJam frames only.

In addition, every module implements functionality to collect statistical records. The measured values are either vector objects, i.e. a series of time and value pairs, or scalars and are stored in `*.vec` or `*.sca` files, respectively.

3.3 INET framework issues

During the implementation, two relevant problems of the INET framework were identified.

3.3.1 Erroneous Round-Robin scheduler

In OMNeT++, not only messages in the common sense, but also events like "a message was inserted into a queue" are represented by the class `cMessage` or a derived class. As described in [12], "the place where the event will occur is the message's destination module, and the time of occurrence the arrival time of the message". So, for instance, a timer is implemented by a message which is sent from a module to itself at a certain time. For a detailed description of event handling in discrete-event simulation environments see Section 2.1.5.

The sequential order of processing events in the simulation environment leads to the fact that the Round-Robin (RR) scheduler which is currently implemented in the INET framework is not completely fair. The ingress part of the switch model comprises seven FIFO queues (each per port) and a RR scheduler which serves the non-empty queues in a cyclic manner. At every turn, exactly one message per non-empty queue is selected. The scheduler leaves out empty queues and serves non-empty queues instead. Assuming two queues A and B where the next queue to be served is B and messages concurrently inserted into both queues, a fair RR scheduler would serve B first and A subsequently.
However, because of the sequential order of processing events, the INET RR scheduler chooses queue A before queue B. More precisely, the OMNeT++ event scheduler schedules the event "message inserted into queue A" before the event "message inserted into queue B", both stored in the FES. The scheduling sequence of the events in the FES is influenced by the ID of the module. When the event "message inserted into queue A" is scheduled, the INET RR scheduler immediately recognizes a non-empty queue A but an empty queue B and serves queue A instead of queue B, although it would have been queue B's turn. This would result in an incorrect simulation model as the RR scheduler always prefers the FIFO queue with the lower module ID. In order to avoid this favoritism, a fair RR scheduler was implemented which first checks if there are concurrent messages by looking into the FES. Based on that, it makes the scheduling decision.

3.3.2 Erroneous throughput-metering channel

In half duplex mode, the INET version of the throughput-metering channel includes bits of unsuccessfully transmitted messages and thus calculates wrong utilization values. According to IEEE 802.3 [24], a node stops the transmission of a message, when it detects a collision and sends jam instead of ensuring that other transmitting nodes on the network detect the collision as well[14]. In the provided model though, the throughput-metering channel adds the bits of a message as soon as a transmission attempt is performed, regardless of the success of the try.

[14]The content of the jam is not defined but has a specified length of jamSize, which is 32 bits in connections with datarates up to 100 Mb/s [24]. After jamming, the nodes start the retransmission of the message according to the "truncated binary exponential backoff" algorithm [24].

The throughput-metering channel was modified in such a way that only bits of a message transmitted until a collision event, successfully transmitted messages and jam are counted as workload. Additionally, the *current channel utilization* is calculated in equidistant intervals to simplify comparison of channel utilization on different links.

3.4 Simulation workflow

The simulation workflow is supported by various Perl[15] and R [53] scripts. They either assist in creating input files for the simulation environment and modules or are used to analyze output of the simulation runs.

3.4.1 Input file generation

The script `NedFileGenerator.pl` generates a `*.ned` file with a network according to channel definitions and a matrix, describing the physical connections among the CNs. For instance, an input file may look as follows:

```
channel: name = c1; datarate = 640kbps; duplex=half;
channel: name = c2; datarate = 10Mbps; duplex=full; delay = 0.0000005s;

+       cn1     cn2     cn3     cn4     cn5
cn1     0       c1      0       0       0
cn2     0       0       c2      0       0
cn3     0       0       0       c1      0
cn4     0       0       0       0       c1
cn5     c1      0       0       0       0

isMRM: cn2
```

This example results in a network with a ring of five CNs plus the necessary modules `ipv4NetworkConfigurator`, `scenarioManager`, `cnAddressTable` and `cn2` set as MRM.

The purpose of `IPConfigFileGenerator.pl` is to produce an XML file for IP address assignment and setting of default gateway routes. Therefore, it takes a list of CNs and assigns IPv4 addresses according to regular expressions, for example:

```
BMZs: cnL0 cnL1 cnL2 cnR0 cnR1 cnR2

IP: regex=/^cnR/; address=198.168.0.0; netmask=255.255.255.240
IP: regex=/L/; address=198.168.1.0; netmask=255.255.255.240
```

The resulting output serves as input for `IDIPMatching.pl` and the `ipv4NetworkConfigurator`[16].

The script `IDIPMatching.pl` is used to create an input file for the module `cnAddressTable`. Therefore, it parses a list of CNs including their IPv4 addresses from a file generated

[15] http://www.perl.org/

[16] It might need some adjustments like adding a multicast group for CARP or additional routes.

by `IPConfigFileGenerator.pl` and combines it with a matrix representing the logical relationships of the CNs.

3.4.2 Simulation data processing

The scripts `ExtractScalars.pl` and `ExtractVectors.pl` are used to extract data from the text-based output files `*.sca` and `*.vec`, respectively and export them in a syntax readable for the R scripts. An example `*.vec` file segment may look like the following:

```
version 2
run Ring64RS485_MRP_sc3-122-20131114-04:01:26-17660
attr configname Ring64RS485_MRP_sc3
attr datetime 20131114-04:01:26
attr experiment Ring64RS485_MRP_sc3
attr inifile mrp64_kw46.ini
attr network ring64.Ring64_RS485_600M
attr repetition 22
attr resultdir results
attr runnumber 122
attr scriptsc0 "\"ring64/scenarios64_3.xml\""
attr seedset 22
attr tstDefaultT 0.5
attr tstNRmax 14
...

vector 1585  Ring64_RS485.cn64.cpu.mrm  mrp_flushFDBinMRPManager  ETV
vector 1577  Ring64_RS485.cn63.cpu.mrc  mrp_flushFDBinMRPClient  ETV
vector 1267  Ring64_RS485.cn1.cpu.mrc  mrp_flushFDBinMRPClient  ETV
vector 1572  Ring64_RS485.cn62.cpu.mrc  mrp_flushFDBinMRPClient  ETV
...
1267    26335524        48.736000465    2
1572    26336060        48.737076903    2
1577    26335499        48.735990465    2
...
```

The extraction is based on applying regular expressions on the vector or scalar names (e.g. *mrp_FlushFDBinMRPClient*) and can be refined by providing an additional regular expression for the module name (e.g. *Ring64_R485.cn63.cpu.mrc*). Both scripts are able to filter input files by an attribute. For instance, it is possible to choose only simulation runs with the attribute `tstDefaultT` set to $0.5s$. For matters of traceability, each generated output file is enclosed with a log file containing the names of all used `*.vec` or `*.sca` files.

`ExtractScalars.pl` additionally calculates delivery ratios of MRP frames and data packets for each sender/receiver combination. The export of the latter can be generated in different gradations, i.e. from one output file for delivery ratios of all senders to a separate file for each communication pair. The script `ExtractVectors.pl` is able to temporally break down extracted series into up to five intervals and to output them in separate files. Furthermore, it can calculate the differences of the maximum and minimum of the time or data values per interval and input file.

For instance, this is a practical function to obtain the time span elapsing between the first and last CN flushing its FDB in case of an MRP switch-over. Another special feature of this script is the calculation of the time span elapsing between the first scenario and the first and last flush of the FDB among all CNs. Moreover, the script gathers packet transmission delays of every communication pair and is also able to output it in various gradations. Beyond that, the script can correct erroneous values by replacing them with a minimum or maximum default value.

The superscript `ExtractAndPlot.pl` calls `ExtractVectors.pl` or `ExtractScalars.pl`. As a meta-script, it automatically generates R scripts for a selection of scalars and vectors, e.g. for analyzing MRP test frame or data packet delays (with the regular expression *mrp_TestFrameDelay* respectively *AD*), the channel utilization (*CH_current_channel_utilization*) or data packet delivery ratios (*dNUM*). When the extracted data was partitioned into multiple intervals, the `ExtractAndPlot.pl` script builds an R script for each one. If enabled, it immediately executes the R scripts. The reason why R scripts are produced for every single data extraction and interval is modifiability. In this way, certain plots can be adjusted by simple modifying the script and executing it again.

The R scripts basically read data from extraction files and plot them in a suitable way to `*.pdf` files. Additionally, some of them calculate statistical values like mean, standard deviation or median and save them to a text file for subsequent manual evaluation.

CHAPTER 4

Simulation

The following sections describe the simulation studies which were carried out during this work. The last section provides a summary of the gained results and knowledge.

4.1 Experiments

As stated in Section 2.1.1, a simulation is used to analyze the behavior of a system, which is a set of cooperating subsystems. Which subsystems are observed, depends on the chosen abstraction level.
In this work, the behavior of the communication system in form of a ring topology is analyzed. The system consists of the subsystems “CNs” and “communication links”. A CN could again be segmented into its subsystems “Switch” and “MCU”, but as the entities (objects of interest) are the links, MRP test frames and data packets, respectively the network performance, the first abstraction level is sufficient.

4.1.1 Channel utilization

The main purpose of this simulation study is to obtain a suitable basis for the configuration of the workload generation for successive experiments. The required workload traffic should utilize the links at the network’s bottleneck at around 66% of their bandwidth. Once an applicable configuration is found, the temporal and spatial allocation of the bandwidth utilization on the network is examined.

The network setup consists of a ring topology of 64 CNs which are connected by half-duplex links with a bandwidth of $640kbps$ (see Figure 4.1). One node — say *cn64*, w.l.o.g. — acts as MRM and sends MRP test frames with the highest switch priority every $1s$. Additionally, it blocks the port toward *cn63* and so the ring transforms into a line topology for non MGMT frames. For workload generation, every CN periodically sends `DataPeriodic` messages with a specified size of payload and a random frequency to every other CN. More precisely, the starting time of the sending process and the periods are drawn from a uniformly distributed interval

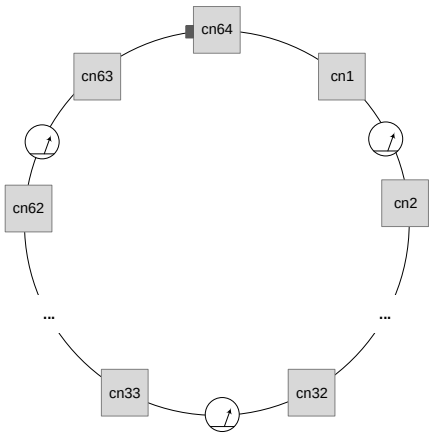


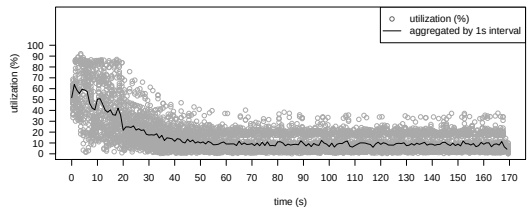
Figure 4.1: A ring of 64 CNs where *cn64* acts as MRM and blocks the port toward *cn63* at the beginning.

for each CN separately. Furthermore, throughput-metering channels (see Section 3.3.2) are installed on multiple links distributed over the ring. Then, alternative systems were derived by choosing different payload sizes and start and sending intervals. As described in Section 2.1.4, new alternatives were created by varying systems which have been already analyzed after the completion of their runs, i.e. steps 8 to 10 of the proposed process flow model were iterated multiple times. For each configuration, 40 replications with a length of $600s$ were run.

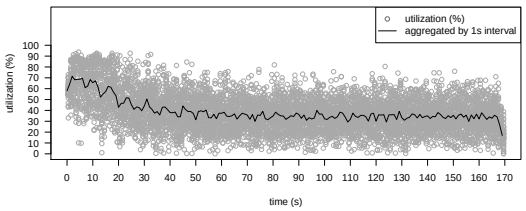
After analyzing the resulting utilizations of these experiments, the following configuration was chosen as the basic setting for successive experiments. In this configuration, every CN sends `DataPeriodic` messages with a payload of 160 bytes to every other node in periods uniformly chosen from $[6s, 9s]$, starting after an offset uniformly drawn from $[0s, 2s]$. The workload is produced with switch priority 2.

Figure 4.2 shows the resulting utilizations of this configuration on selected links. Particularly on the links cn1 – cn2, cn8 – cn9 and cn16 – cn17 the utilization is higher at the beginning and declines gradually until $50s$. After this, the mean utilization remains tolerably constant with a fluctuation of 15% on all links. Thus, only a time range of $[0s, 170s]$ was investigated further.

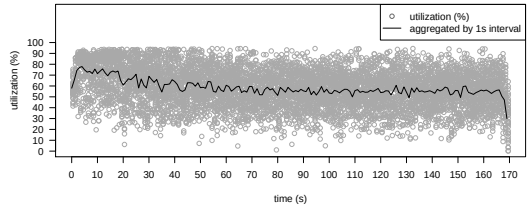
The boxplots of the channel utilizations, partitioned in two intervals, are depicted in Figure 4.3. As expected, the link utilization decreases with increasing distance of the link to the middle of the ring respectively line. This origins from the fact that traffic sent from one end of the line to the other end has to pass the links in the middle, which therefore can be seen as the bottlenecks of the network. As shown in Figure 4.3a and 4.3b, the median channel utilizations during the interval $[0s, 50s)$ are higher than the median channel utilizations during $[50s, 170s]$. The mean median utilization of all measured links is 63.2% during $[0s, 50s)$ and 48.5% for the remainder of the simulated time. It is $\approx 70\%$ on the bottleneck links for the interval $[50s, 170s]$. The high channel utilizations in the first $50s$ is caused by the extensive communication of the ARP (as described in Section 4.1.2).



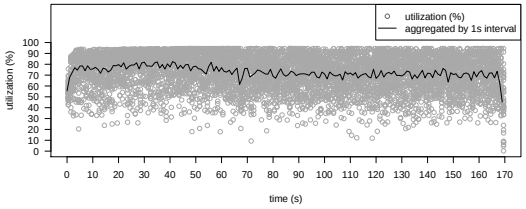
(a) Link cn1 – cn2



(b) Link cn8 – cn9

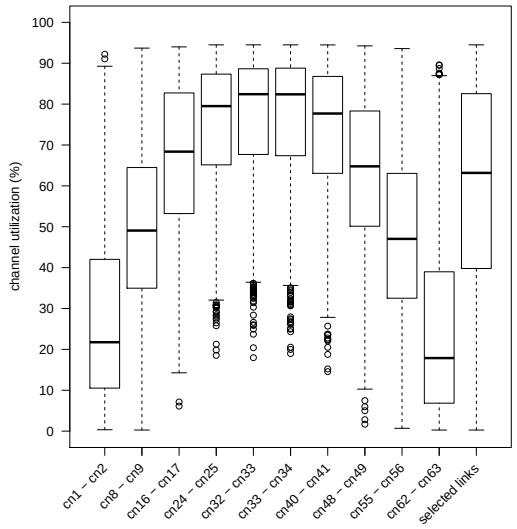


(c) Link cn16 – cn17

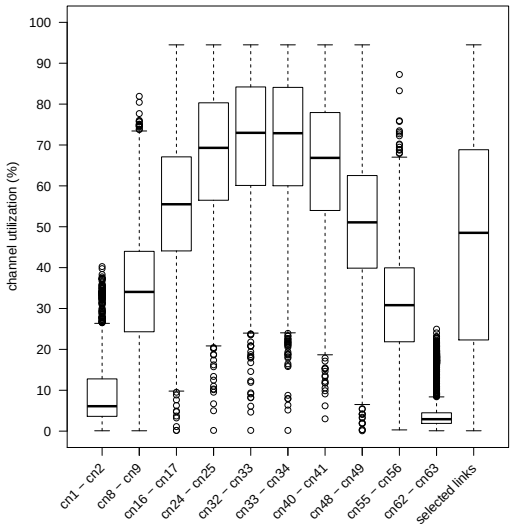


(d) Link cn32 – cn33

Figure 4.2: Channel utilizations on chosen links measured in the range $[0s, 170s]$ of the simulated time. The circles represent the utilizations of different runs and the lines show their mean value.



(a) Interval $[0s, 50s)$



(b) Interval $[50s, 170s]$

Figure 4.3: Boxplots of the channel utilizations on selected links and overall channel utilization of these links in "selected links", partitioned in two intervals.

4.1.2 Address Resolution Protocol

In this simulation study, three different approaches for implementing and simulating ARP are compared. The version provided by INET is contrasted with the implemented version with either *ULC* enabled or disabled (for a description see Section 3.2.1). Additionally, the influence of these ARP alternatives on the channel utilization and consequently transmission delays was investigated.

For conducting the study, the following three scenarios were defined:

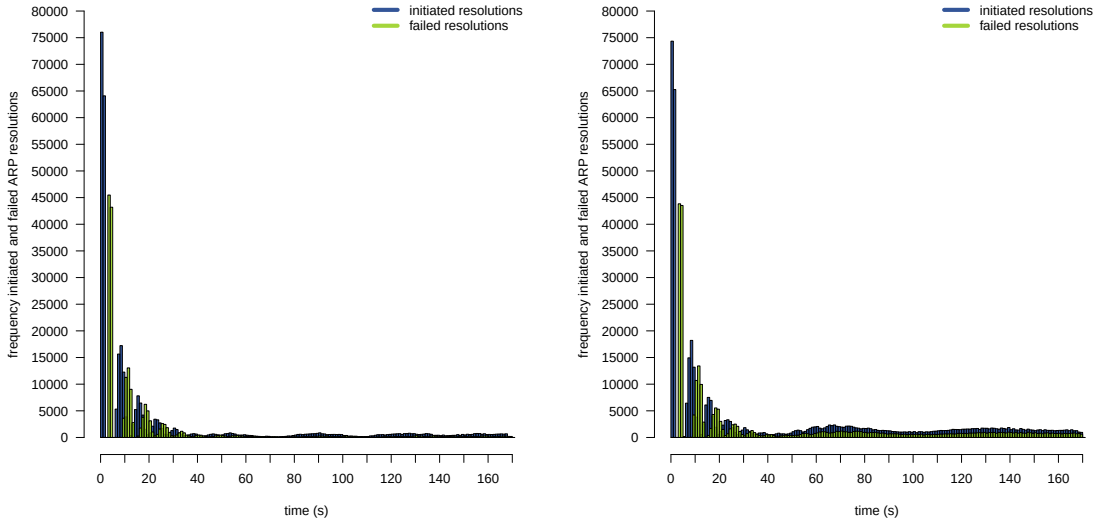
- *INET_ARP*: using the INET implementation of ARP with
  - cache table timeout time = $30s$
  - retry timeout = $1s$
  - retry count = 3
- *ULC_enabled*: using the implemented ARP with ULC enabled and[1]
  - cache table timeout time in $[15s, 45s]$ (derived from base_reachable_time = $30s$)
  - delay first probe = $5s$
  - retry timeout (retrans_time) = $1s$
  - retry count unicasts (ucast_solicit) = 3
  - retry count multicasts (mcast_solicit) = 3
- *ULC_disabled*: is the same configuration as *ULC_enabled* only with ULC disabled

The upper-layer reachability confirmation process is usually done by protocols at layer 4 (w.r.t. ISO/OSI, e.g., TCP) but since the *MCU* model only covers layers two and three of the OSI model, it is imitated by the `cnApp`. So, whenever the application receives a packet, it completes the confirmation.
The ring was utilized as described above and ARP packets were sent with the lowest switch priority. The length of a simulation run was defined with $170s$ and the number of replications with 40. This length is reasonable long enough, as it covers the initial phase with higher channel utilization and the ARP cache table timeouts for around four times.

Initiated and failed ARP resolutions

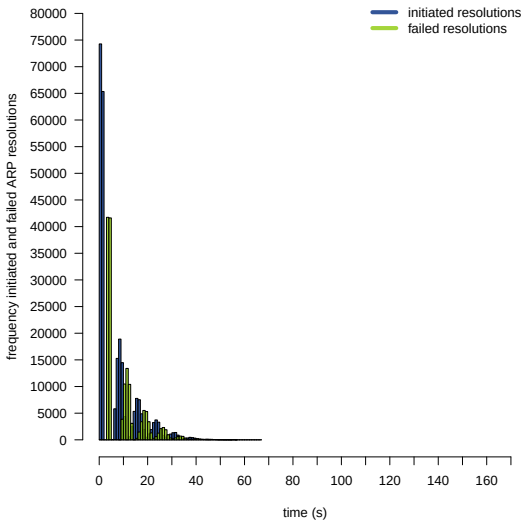
The dispersion of the initiated and failed resolutions resulting from these experiments over time are shown in Figure 4.4. Figure 4.5 summarizes these values for each scenario. In all scenarios considerably more resolutions were initiated in the interval $[0s, 50s)$. In scenario *ULC_enabled*, 99.97% of the resolutions were initiated in this interval, whereas for scenario *ULC_disabled* it were 59.57% and for *INET_ARP* 81.45%.

[1] These values are according to arp(7), see `http://linux.die.net/man/7/arp`.



(a) Scenario *INET_ARP*

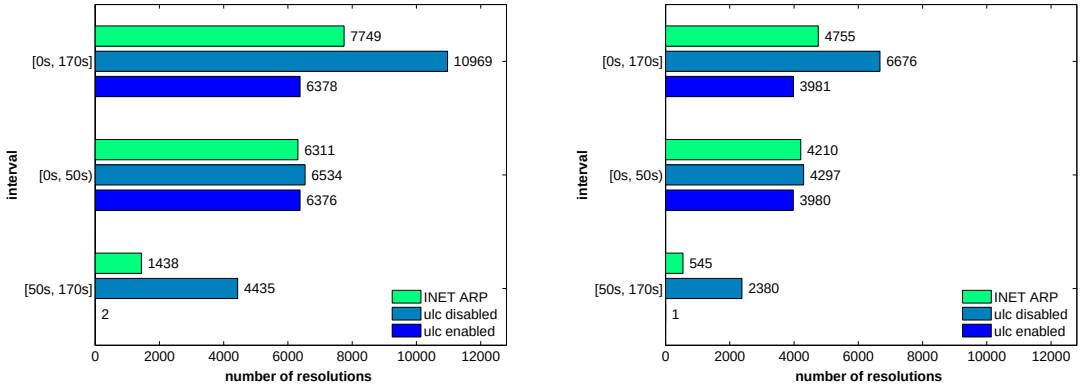
(b) Scenario *ULC_disabled*



(c) Scenario *ULC_enabled*

Figure 4.4: Frequencies of initiated and failed ARP resolutions of all simulation runs.

This is because all nodes start sending packets at the beginning having empty ARP caches. Considering the whole simulation time, scenario *INET_ARP* requires 21% and scenario *ULC_disabled* 72% more resolutions than scenario *ULC_enabled* (see Figure 4.5a). The histogram in Figure 4.4c shows that in scenario *ULC_enabled* the last ARP resolution was initiated at $62.16s$ and failed at $65.16s$.



(a) Initiated resolutions

(b) Failed resolutions

Figure 4.5: Number of the initiated and failed ARP resolutions (the values are normalized, i.e. divided by the number of simulation replications, and rounded).

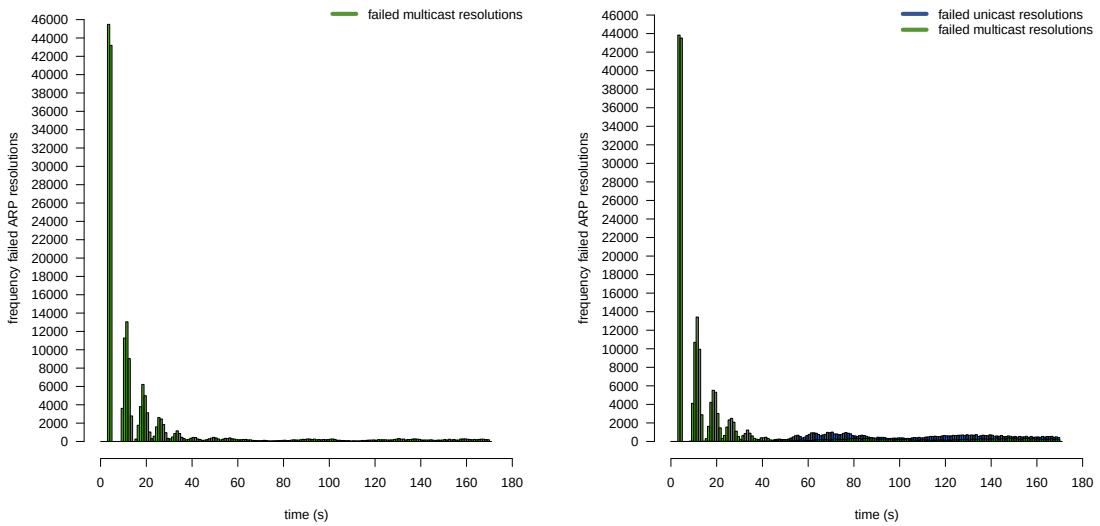
In scenario *INET_ARP*, 88% of all failed resolutions happened in the interval $[0s, 50s)$, whereas for *ULC_disabled* and *ULC_enabled* 64% and 99.97% failed in the same period, respectively (see Figure 4.5b). In each scenario, during the first $50s$ about two-third (62%–66.77%) of the initiated resolutions failed. Most of the resolutions fail in the beginning due to the higher channel utilization in this phase. Interestingly, *ULC_disabled* has about 41% more initiated and failed resolutions than *INET_ARP* considering the whole simulation time. In interval $[50s, 170s]$, it has about 208% more initiated and around four times as many failed resolutions.

According to [52], ARP just broadcasts resolutions. In the *Neighbor Discovery for IPv6* protocol [51] multicast-based resolutions are made instead of broadcast resolutions. The implemented ARP module only broadcasts. However, in this work "multicast" is used as a synonym for "broadcast" and "multicast".

In Figure 4.6, the frequencies of the failed multicast resolutions of scenario *INET_ARP* and failed uni- and multicast resolutions of *ULC_disabled* are contrasted. After the first $50s$, approximately the same number of multicast resolutions fail in *INET_ARP* and *ULC_disabled*. In the latter scenario though, additionally unicast resolutions fail.

Sent ARP requests

The frequencies of the sent ARP requests (unicast and multicast requests in case of the implemented ARP), are depicted in Figures 4.7(a)-(c) over the whole simulation time and summarized in Figure 4.8. Intuitively, the number of requests sent in the two periods $[0s, 50s)$ and $[50s, 170s]$ reflect the number of resolutions made. In scenario *ULC_enabled*, almost all requests were sent in the first $50s$ and 99.5% of them were multicast requests. As a consequence, by using upper-layer reachability confirmation all ARP resolutions are solved with ARP multicast requests at the beginning and do not have to be resolved later.



(a) Scenario *INET_ARP*

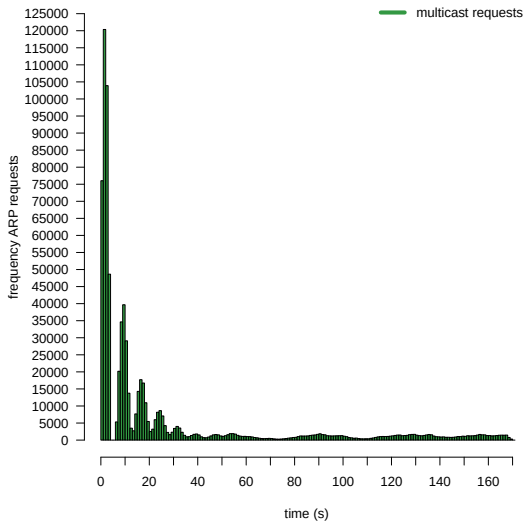
(b) Scenario *ULC_disabled*

Figure 4.6: Frequencies of failed ARP resolutions of all simulation runs. In case of scenario *ULC_disabled* unicast and multicast resolutions.

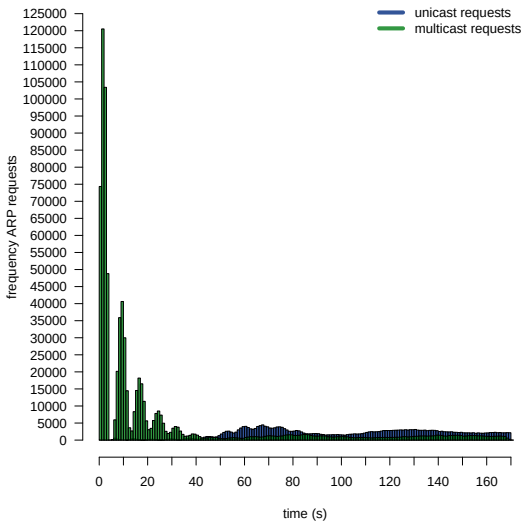
In scenario *ULC_disabled*, two-third (60.84%) of the requests are performed in the interval $[0s, 50s)$, which are again mainly multicast requests (97.6%). On the contrary, approximately two-third (70.7%) of the requests sent in $[50s, 170s]$ were unicasts.

In scenario *INET_ARP* and *ULC_disabled* almost the same number of multicast requests were necessary in both period $[0s, 50s)$ and $[50s, 170s]$ (the difference is about 88 requests per run), but in scenario *ULC_disabled* additionally unicast requests were sent.

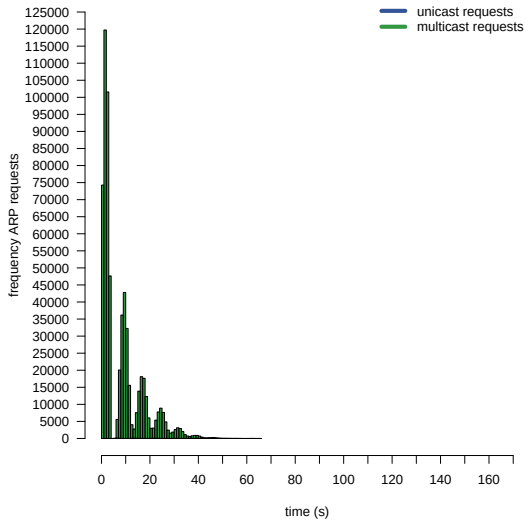
Not necessarily the random cache table timeout time is responsible for the additional unicast requests in scenario *ULC_disabled*. In both the INET and implemented ARP, a cache table entry of an address is updated when the module receives an ARP request having this address stored as source IP address. The INET ARP sends requests always as broadcasts, and so every node in the broadcast domain is updating its cache. Thus, caches of all nodes in the ring are continuously updated by some nodes' requests. On the contrary, in the implemented version of ARP a stale entry is resolved by a unicast request. Since nodes delete the unicast MAC address assigned frame at the MAC layer when it is not designated for them, they do not update their cache table. On the other hand, this results in further unicast and multicast requests, when the former are unsuccessful.



(a) Scenario *INET_ARP*

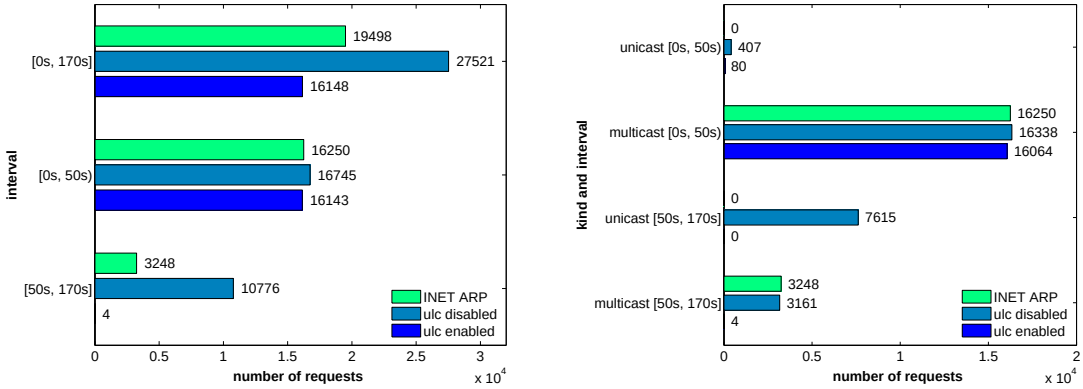


(b) Scenario *ULC_disabled*



(c) Scenario *ULC_enabled*

Figure 4.7: Frequencies of sent ARP requests of all simulation runs.



(a) All requests

(b) Separated in unicast and multicast requests

Figure 4.8: Number of the sent ARP requests (normalized and rounded values).

Effect on channel utilization, delays and delivery ratios

A slightly higher median channel utilization (on all measured links) can be identified in the first $50s$ of scenario *ULC_enabled* (63% compared to 61% for scenario *ULC_disabled* and *INET_ARP*). During the remaining simulation time though, the median utilization is lower (48% compared to 50%). This is also reflected in the delays of packets. As summarized in Table 4.1, scenario *ULC_enabled* has a larger mean delay in the first interval, whereas the variation of the MRP test frame delays ($0.58s$) of all scenarios is larger than the one of data packets ($0.37s$). In the second interval, scenario *ULC_enabled* has clearly smaller mean delays. The delta of *ULC_enabled* and *ULC_disabled* is $1s$ for MRP test frames and $0.8s$ for data packets. The difference to *INET_ARP* is $0.54s$ for MRP test frames and $0.43s$ for data packets.

	$[0s, 50s)$			$[50s, 170s]$		
Delays	***INET_ARP***	***ULC_dis.***	***ULC_en.***	***INET_ARP***	***ULC_dis.***	***ULC_en.***
MRP test f. mean	5.164s	5.092s	5.670s	2.710s	3.178s	2.173s
MRP test f. std. dev.	2.714s	2.781s	3.059s	1.445s	1.591s	1.207s
MRP test f. median	4.839s	4.760s	5.282s	2.605s	3.096s	2.058s
Data p. mean	3.451s	3.419s	3.786s	2.461s	2.821s	2.034s
Data p. std. dev.	3.090s	3.083s	3.372s	2.093s	2.401s	1.781s
Data p. median	2.803s	2.771s	3.076s	2.122s	2.443s	1.719s

Table 4.1: Delays of MRP test frames and data packets during $[0s, 50s)$ and $[50s, 170s]$.

Table 4.2 lists the means and standard deviations of the delivery ratios. The different channel utilizations do not effect the delivery ratio of high priority frames such as MRP test frames but the ratio of lower priority packets like data packets. As expected, ARP with upper-layer reachability confirmation enabled has the best delivery ratio, followed by the INET implementation.

	$[0s, 170s]$		
Delivery ratios	***INET_ARP***	***ULC_disabled***	***ULC_enabled***
MRP test frames mean	98.289%	98.026%	98.896%
MRP test frames std. dev.	0.620%	0.748%	0.456%
Data packets mean	94.486%	90.850%	95.579%
Data packets std. dev.	6.696%	10.127%	5.646%

Table 4.2: Delivery ratios of MRP test frames and data packets.

Although the delivery ratio of MRP test frames is about the same for all three scenarios, *ULC_enabled* has shorter MRP test frame delays. Furthermore, in *ULC_enabled* more data packets are delivered with a smaller delay (in $[50s, 170s]$). As a consequence, enabling the `MSG_CONFIRM` flag when using the linux `send` function[2] results in a better performance. Periodically sent "Hello" packets for monitoring point-to-point connections are keeping the ARP caches up to date, too. Also, knowing the first $50s$ to be ARP excessive, a reconfiguration of the MRP parameters (e.g., the $tstNRmax$ parameter) after this period leads to a shorter ring failure detection time.

4.1.3 Media Redundancy Protocol

The delay of an MRP test frame T_{ring}, i.e. the time the frame needs to be passed over the whole ring (cf. Section 5.3), is crucial for the configuration of MRP parameters. In the ring-closed state, the MRM periodically sends test frames every $tstDefaultT$. When it does not receive a test frame back within $T_{test} = tstDefaultT * tstNRmax$, the MRM detects that a failure has happened and changes to the ring-open state by setting the secondary port to forwarding.

Assuming an MRP test frame delivery ratio of 100%, the MRM has to be configured considering that $T_{ring} < T_{test}$ holds. Otherwise, it would introduce a switch-over without a link failure has happened and consequently a loop would emerge. Assuming that the *first* test frame, which was sent when the MRM switched to the ring-closed state in the first place, does not get lost, the loop would exist for the duration of $T_{ring} - T_{test}$. With a delivery ratio smaller than 100%, the boundary for T_{ring} is even more narrow. Under the assumption that *exactly one* test frame can be delivered during *a* test interval, the constraint for T_{ring} is reduced to $T_{ring} < tstDefaultT$. This means that the last frame sent in a test interval which has to be received before the ring would be falsely detected as being opened, has to be delivered within $tstDefaultT$.

However, the time the MRM needs to detect an open ring depends on T_{test}. The actual time elapsing from the occurrence of the failure to its detection is $T_{detect} = T_{test} + T_{ring}$ (T_{ring} as dead time) [47]. This value is even shortened on the reception of a LinkDown frame to speed up this procedure. Hence, the configuration of an MRM is aiming at keeping T_{test} minimal and accordingly, as $T_{test} > T_{ring}$ must hold in an optimal system, also T_{ring} has to be minimized.

[2] http://linux.die.net/man/2/send

Moreover, the detection of a closed ring depends on T_{ring}, too. It can be accomplished faster with shorter test frame delays.

The purpose of the following two simulation studies is to investigate how different priorities and $tstDefaultT$ parameters affect T_{ring}. Moreover, the influence on delivery ratios and data packet delays is explored.

MRP test frames with different switch priorities

This study shows how different MRP test frame $priorities$[3] and $tstDefaultT$ values affect the performance of the ring.
For the setup, a ring with 64 nodes was constructed and utilized according to Section 4.1.1. *Node64* acts as MRM and blocks the port toward *node63*. Additionally, MRP test frames were sent periodically with different $tstDefaultT$ values and $priorities$.

12 alternative systems were specified by defining scenario $sc(\$tstDefaultT, \$priority)$ as follows:

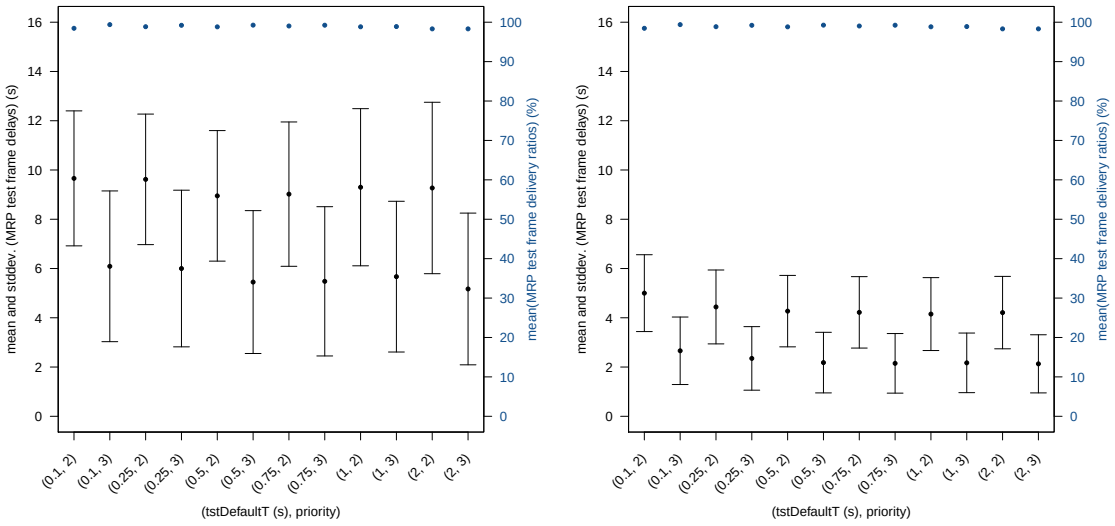
- $\$tstDefaultT \in \{0.1s, 0.25s, 0.5s, 0.75s, 1s, 2s\}$ and $\$priority \in \{2, 3\}$
- using the implemented ARP with upper-layer reachability confirmation enabled
- data packets are sent with switch priority 2

Again, the simulation length was specified with $170s$ and the number of runs was set to 20 for each scenario.

Figure 4.9 shows the MRP test frame delays of all scenarios partitioned in ARP excessive and non-excessive phase. In both phases, using different priorities has a major influence on the delays. In the ARP non-excessive phase, the mean delays of priority 3 frames are about $2.1s$ lower than priority 2 frames' delays. In the ARP excessive period, this delta is even higher with approximately $3.7s$. Considering the interval $[0s, 50s)$ only and each priority separately, the means of the scenarios with different $tstDefaultT$ values vary in a range of $1s$. The means of all scenarios with priority 3 in the ARP non-excessive phase are distributed in a range of about $250ms$. Thus, the delays of scenarios with priority 3 (scenarios *sc(*,3)*) in the non-excessive ARP phase depend at least of all scenarios on the $tstDefaultT$ parameter.

Comparing the ARP excessive and non-excessive phases, the mean delays of priority 2 frames are $4.9s$ higher in the interval $[0s, 50s)$ than in $[50s, 170s]$. For frames with priority 3, this delta is lower with $3.4s$. It holds for both priority sets, that the standard deviation of the delays is higher in the first interval than in the second interval. The highest standard deviation emerges in scenario $sc(2s, 2)$ in the ARP excessive phase and is $3.481s$, the smallest with $1.18s$ in scenario $sc(2s, 3)$ during the non-excessive phase.

[3]The switch supports four priorities. In the implementation, three corresponds to the highest priority.



(a) ARP excessive phase $[0s, 50s)$

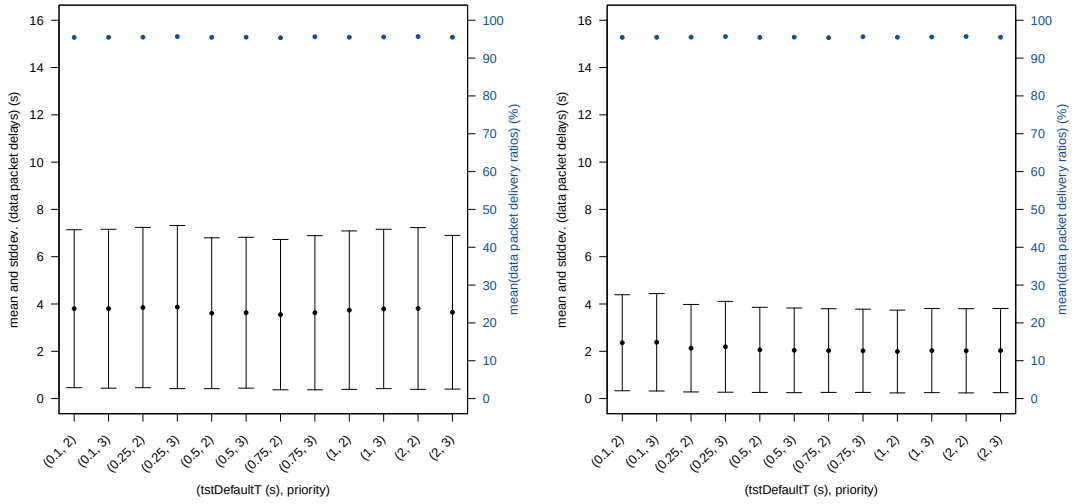
(b) ARP non-excessive phase $[50s, 170s]$

Figure 4.9: Means and standard deviations of MRP test frame delays using different $tstDefaultT$ and $priority$ values.

Out of all mean delays, the smallest mean value has scenario *sc(2s, 3)* with $2.13s$ in the non-excessive interval. Exact values are provided in the Tables A.1, A.2, A.3 and A.4 in the appendix.

Furthermore, the mean delivery ratios of MRP test frames with priority 3 are slightly larger than the corresponding ratios with priority 2 (see Figure 4.9). The same holds for delivery ratios of data packets (see Figure 4.10). Exact values are provided in Table A.5 and Table A.6. The highest mean delivery ratio can be identified in scenario *sc(0.1s, 3)* with 99.387% for MRP test frames and scenario *sc(2s, 2)* with 98.314% for data packets, respectively.
The mean data packet delays of all scenarios are depicted in Figure 4.10. The mean delay is approximately $3.8s$ during ARP excessive and $2.1s$ during ARP non-excessive phase. Within an interval, the mean delay does not depend on the $tstDefaultT$ nor on the priority of the MRP test frames. Also, the median channel utilization on all measured links does not depend on different $tstDefaultT$ values and priorities.

As a conclusion, the MRP test frames' priority (either 2 or 3) does not affect the delays and delivery ratios of data packets. In contrast, it has indeed an effect on the MRP test frame delays. In general, in both the ARP excessive and non-excessive phase, test frames sent with priority 3 have shorter delays than priority 2 frames. Also, the difference between the delays in the excessive phase and in the non-excessive phase are smaller for priority 3 frames as for priority 2 frames.



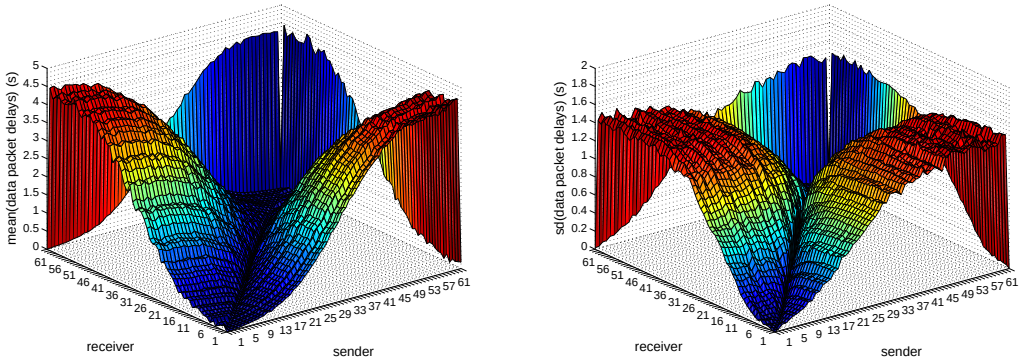
(a) ARP excessive phase $[0s, 50s)$

(b) ARP non-excessive phase $[50s, 170s]$

Figure 4.10: Means and standard deviations of data packet delays using different $tstDefaultT$ and $priority$ values.

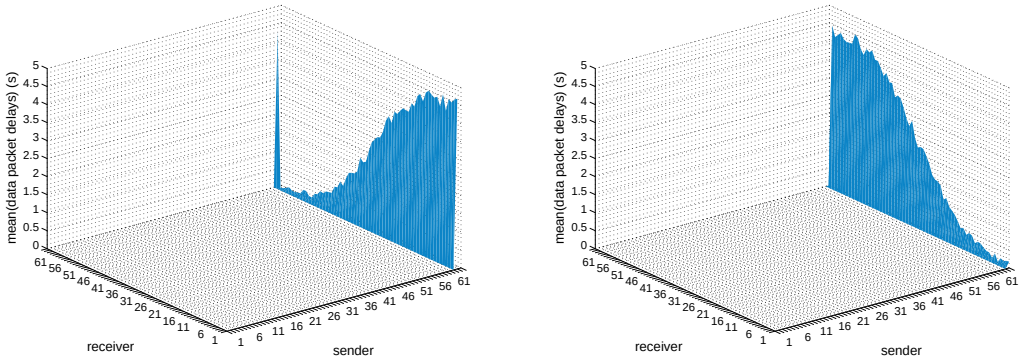
For workload generation, every node sends data packets to every other node. The mean data packet delay and standard deviation for every communication pair of scenario *sc(1s, 3)* are depicted in Figure 4.11. The values at *sender* $=$ *receiver* is 0 as a node does not send packets to itself. Intuitively, the delay increases with the distance of the receiving node on the ring. Also, the standard deviation increases proportionally to the distance between the sending and receiving node (see Figure 4.11b). From the perspective of data packets, particularly non MGMT frames, the ring topology performs like a line topology. In this setup up, *node64* blocks the port toward *node63*, thus the line topology "*node64 – node1 – node2 – . . . – node62 – node63*" is formed. For instance, from *node1* as sender, the delay increases until $4.494s$ at receiver *node63* and then suddenly falls to $0.0052s$ at receiver *node 64*. This is because *node64* is an adjacent neigbor of *node1* and *node63* is located on the opposite end of the line. The "butterfly-shaped" areas in the *sender/mean* and *receiver/mean* planes result from the ring being opened between *node63* and *node64*. For a better understanding, Figure 4.12 shows the mean delays of packets sent from sender *node63* and *node64*. The peak in the corner of Figure 4.12a indicates that packets sent from *node63* to *node64* have to be transmitted over the whole ring.

The mean data delivery ratio and standard deviation of every communication pair in scenario *sc(1s, 3)* are shown in Figure 4.13. For reasons of better illustration, the ratio of the pairs $(sender, sender)$ was set to 100% and the standard deviation to 0%. The means decrease with the increasing distance between the sender and receiver node, whereas the standard deviations increase.



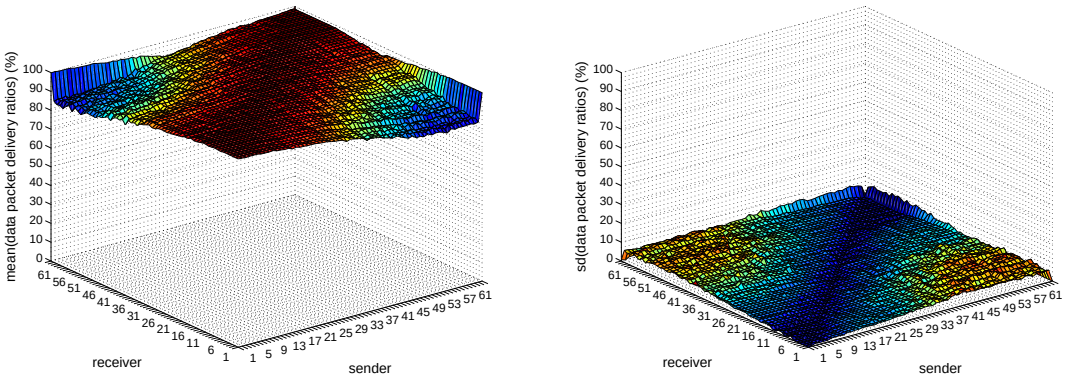
(a) Means (b) Standard deviations

Figure 4.11: Mean delay and standard deviation of data packets pairwise sent from *sender* to *receiver* in scenario *sc(1s, 3)* during the period $[50s, 170s]$.



(a) *Node63* as sender (b) *Node64* as sender

Figure 4.12: Mean delay of data packets sent from *node63* and *node64* in scenario *sc(1s, 3)* during the period $[50s, 170s]$.



(a) Means

(b) Standard deviations

Figure 4.13: Mean data packet delivery ratios of packets sent pairwise from *sender* to *receiver* in scenario *sc(1s, 3)* during the period $[0s, 170s]$.

Data packets with different switch priorities

The aim of this simulation study is to examine how different priorities for data packets influence MRP test frame and data packet delays and their delivery ratios. The set up used for this experiments is a ring of size 64 (see Figure 4.1), where *cn64* acts as MRM and sends test frames periodically every $1s$ with priority 3. The workload was generated according to Section 4.1.1.

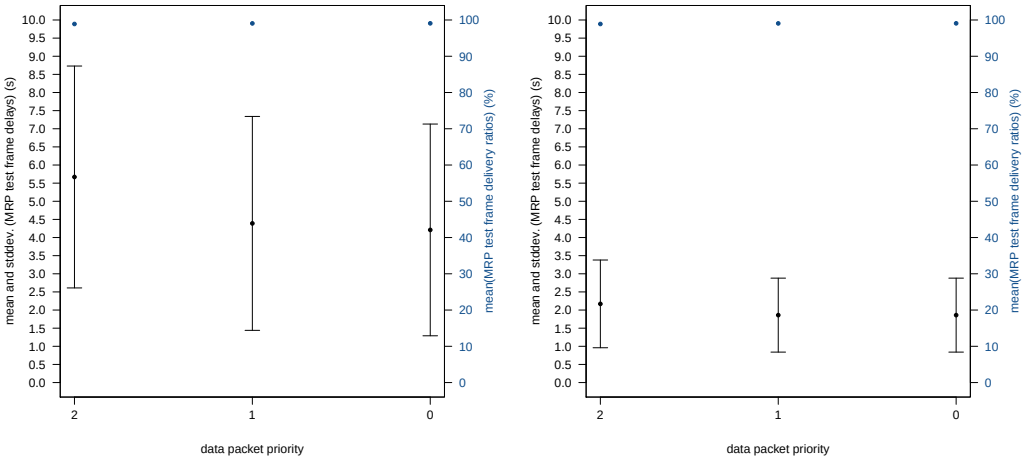
Three different system alternatives are specified, where scenario *priority($dataPriority)* is defined as follows:

- data packets are sent with switch priority $\$dataPriority \in \{0, 1, 2\}$
- $tstDefaultT$ is $1s$ and the priority of MRP frames is 3
- the implemented ARP with ULC enabled is used

For each of the alternatives 40 runs were executed, each with a simulation duration of $170s$.

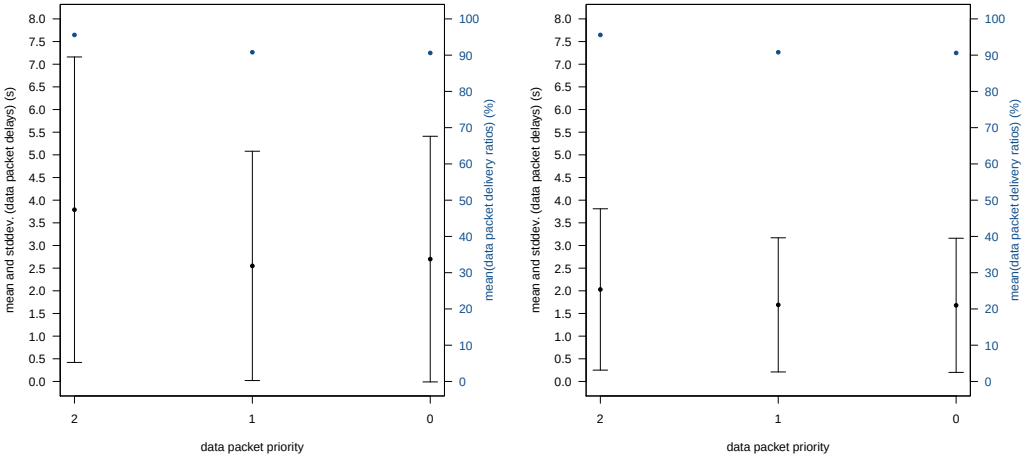
The resulting means and standard deviations of the MRP test frame delays are depicted in Figure 4.14 for the ARP excessive and non-excessive phase, respectively. As expected, a higher data packet priority of 2 leads to higher mean delays of MRP test frames. In the ARP excessive phase, the mean delay is approximately $1.4s$ and in the non-excessive phase about $0.3s$ higher than of the other two scenarios with data priority 1 and 0. Also, the mean data packet delays are larger for scenario *priority(2)*, whereas the delta is about $1.2s$ and $0.4s$ (see Figure 4.15). During the interval $[0s, 50s)$, the mean delay of scenario *priority(0)* is marginally higher than of scenario *priority(1)* due to the transmission of ARP packets with priority 0 in this period. All three scenarios result in approximately the same MRP test frame delivery ratio.

In contrast, a data packet priority of 2 leads to a much better delivery ratio of 95.6% compared to about 90.7% for data priority 1 and 0. The exact values of delays and ratios are listed in Tables A.7 and A.8.



(a) ARP excessive phase $[0s, 50s)$ (b) ARP non-excessive phase $[50s, 170s]$

Figure 4.14: Means and standard deviations of MRP test frame delays using different priorities.



(a) ARP excessive phase $[0s, 50s)$ (b) ARP non-excessive phase $[50s, 170s]$

Figure 4.15: Means and standard deviations of data packet delays using different priorities.

To sum up, a higher data packet priority leads to a better data packet delivery ratio but worse delays. Data packets with lower priority are faster if they can be switched. Furthermore, choosing priority 1 or 0 does not make any difference for MRP or data transmission performance during the ARP non-excessive phase.

4.1.4 Scalability of the Media Redundancy Protocol

According to [47], the following requirements shall be fulfilled in order to maintain a maximum recovery time of $\leq 500ms$ using the MRP:

- The link speed shall be at least $100Mbps$.
- The operation mode shall be full duplex.
- The number of nodes shall be smaller than 50, otherwise the ring may become instable.

In this work, ring topologies are needed which do not meet all of the aforementioned conditions, since:

- The link speed is $640kbps$.
- The links connecting the MRCs and the MRM operate in half duplex mode.
- More than 50 nodes are expected.

As explained in Section 4.1.3, the period an MRP test frame needs to be transmitted over the whole ring is critical for the configuration of the MRP. This simulation study examines how MRP behaves with an increasing number of nodes under the two constrictions half duplex mode and reduced link speed. In addition, the trend of the delay and delivery ratio of data packets is investigated. Furthermore, the results provide an estimator for arbitrary ring sizes by means of polynomial functions.

For the experiments, 133 alternative systems are defined, where scenario *scale($ring size, $tst-DefaultT)* is constructed as follows:

- $\$ring\ size \in \{10n \mid n \in \mathbb{N} \wedge 7 \leq n \leq 25\}$
- $\$tstDefaultT \in \{0.05s, 0.1s, 0.25s, 0.5s, 0.75s, 1s, 1.5s\}$
- using the implemented ARP with upper-layer reachability confirmation enabled
- MRP test frames are sent with switch priority 3
- data packets are sent with switch priority 1

In all resulting ring topologies, node *n0* is defined as MRM and blocks the port toward the node with the highest index (see Figure 4.16). Thus, a line topology "*n0* – *n1* – ... – $n_{\textit{ring size-2}}$ – $n_{\textit{ring size-1}}$" is formed for non MGMT frames.

For traffic generation, the first 32 nodes at the leftmost side of the line topology (i.e., {*n0*, ..., *n31*}) send data to the 32 nodes at the rightmost side (i.e., {$n_{ring\ size-32}$, ..., $n_{ring\ size-1}$}) and vice versa. The nodes start sending packets with $160byte$ payload at an arbitrary point in time in $[0s, 5s]$ and continue in periods uniformly chosen from $[4s, 7s]$. Using this configuration, the link in the middle of the ring, i.e. the bottleneck, has a median channel utilization of approximately 65%. Per configuration, a number of 10 replications with each a simulation length of $120s$ was simulated.

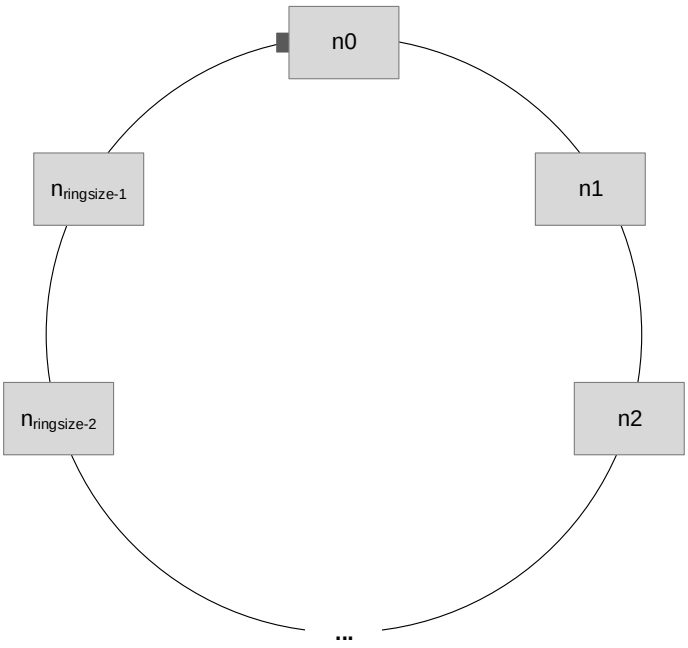
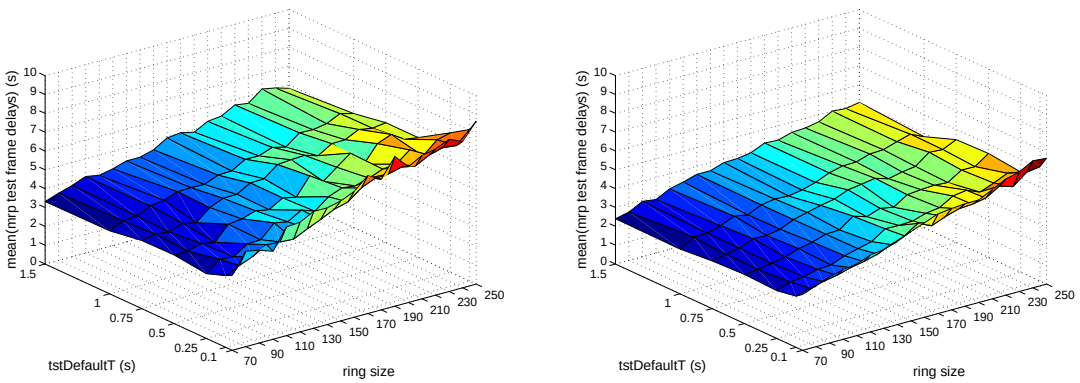


Figure 4.16: The setup of a ring topology with a size of $ring\ size$ where $n0$ acts as MRM.

In all scenarios, ARP communication utilizes the network mostly in the first $50s$. ARP resolutions performed after the first $50s$ are negligible for the network load. The means and the standard deviations of the MRP test frame delays of all scenarios, partitioned in ARP excessive and non-excessive phase, are depicted in Figure 4.17 and Figure 4.18, respectively.

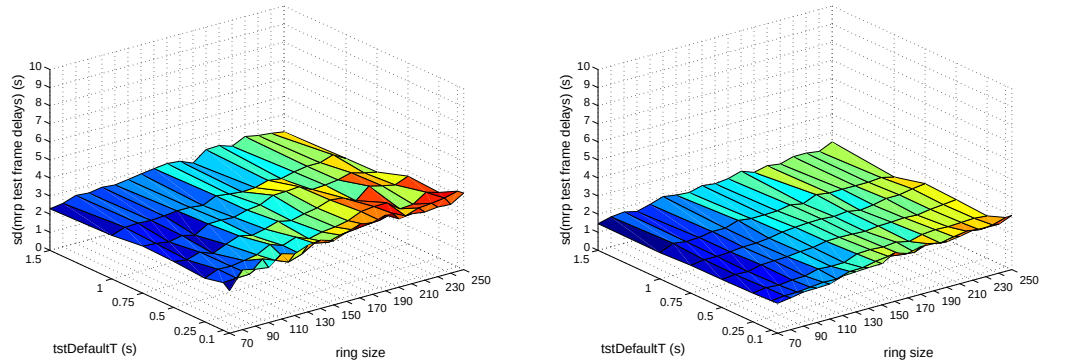
Generally speaking, the mean delays and the standard deviations increase proportionally to the $ring\ size$ and inversely proportional to $tstDefaultT$. The highest mean delays result from $tstDefaultT = 0.05s$ and large ring sizes like 250 for the excessive and 240 for the non-excessive phase. Considering the interval $[50s, 120s]$ only, the highest mean delay is $6.74s$ for scenario *scale(240, 0.05s)* and the lowest $2.38s$ for $tstDefaultT = 1s$ and a ring with size 70. The highest standard deviation has scenario *scale(250, 0.05s)* with $3s$ and the lowest scenario *scale(70, 1s)* with $1.45s$.

Examining the values in a mathematically correct way, the mean delays do just *tend* to increase proportionally according to the ring size for each $tstDefaultT$, but are not *strictly* mono-



(a) ARP excessive phase $[0s, 50s)$ (b) ARP non-excessive phase $[50s, 120s]$

Figure 4.17: Means of MRP test frame delays partitioned in ARP excessive and non-excessive phase.



(a) ARP excessive phase $[0s, 50s)$ (b) ARP non-excessive phase $[50s, 120s]$

Figure 4.18: Standard deviations of MRP test frame delays partitioned in ARP excessive and non-excessive phase.

tonically increasing. However, for $tstDefaultT = 0.05s$ the mean delay of ring size 240 (scenario *scale(240, 0.05s)*) is only $0.0032s$ higher as the delay of ring size 250 (scenario *scale(250, 0.05s)*). Considering this difference as negligible, the delays for $tstDefaultT = 0.05s$ can be seen as strictly monotonically increasing with the ring size. It holds for all $tstDefaulT$ values, that the difference of the delays at the ring sizes where the delays violate the condition for being monotonically increasing, is $0.33s$ maximum.

For providing an estimator of the mean delays for arbitrary ring sizes, a polynomial function

$$MD_{tstDefaultT}(ring\ size) = p_1 * ring\ size + p_2$$

was deduced from the simulation results for both the ARP excessive and non-excessive phase.

The coefficients for $MD_{tstDefaultT}$, fitting the results of interval $[50s, 120s]$ according to the method of least squares[4], are listed in Table 4.3. The polynomials and the corresponding results of the experiments are illustrated in Figure 4.19b. Concluding from these polynomials, the smallest mean delays are achieved using the highest $tstDefaultT$ parameter $1.5s$. The derived polynomials and simulation results for period $[0s, 50s)$ are shown in Figure 4.19a. In this phase, the polynomials have a higher y-intercept and slope, i.e. the delays increase faster proportionally with the ring size than in the non-excessive phase. The coefficients are provided in Table A.9.

The maximum delta between the mean delays of interval $[0s, 50s)$ and $[50s, 120s]$ for ring size 250 has $tstDefaultT = 0.1s$ with $1.94s$, the minimum $tstDefaultT = 1.5s$ with $0.99s$. The minimum overall difference has scenario *scale(70, 1.5s)* with $0.85s$. Thus, the difference between the mean delays of the ARP excessive and non-excessive phase is at most $1.94s$ and at least $0.85s$ for each $tstDefaultT$ and all ring sizes. Roughly speaking, the deltas between ARP excessive and non-excessive phase increase with a decreasing $tstDefaultT$ value.

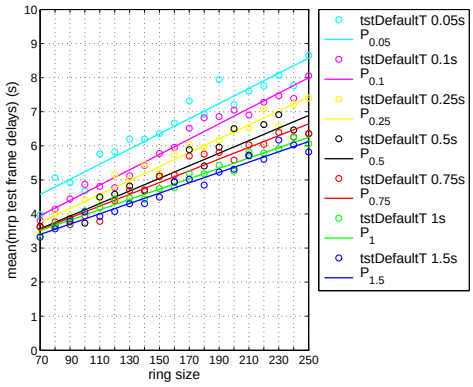
TstDefaultT	**p_1**	**p_2**
0.05s	0.0207	1.5753
0.10s	0.0184	1.4587
0.25s	0.0164	1.4824
0.50s	0.0150	1.5339
0.75s	0.0154	1.5096
1.00s	0.0143	1.5864
1.50s	0.0144	1.5466

Table 4.3: Coefficients of $MD_{tstDefaultT}$ for mean MRP test frame delays during the interval $[50s, 120s]$.

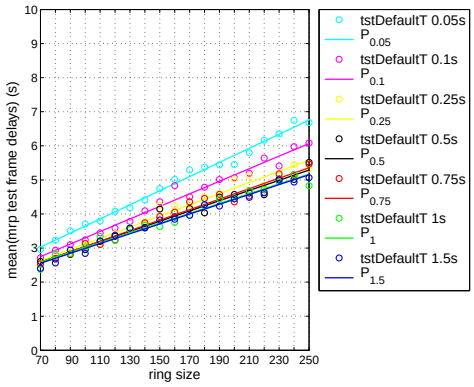
In Figure 4.20a, the mean delivery ratios of MRP test frames of all scenarios are shown. Generally speaking, the mean delivery ratio falls with increasing $ring\ size$ and faster for higher $tstDefaultT$ values. However, the delivery ratios do not strictly monotonically decrease for a certain $tstDefaultT$ over all ring sizes. Values which do not comply with the monotonicity have a mean difference of 0.5%, the maximum difference is 2.22%.

Again, a linear polynomial function $MR_{tstDefaultT}$ was derived from the simulation results of the whole duration $[0s, 120s]$. The coefficients are listed in Table 4.4 and the graphs are depicted in Figure 4.20b.

[4]For calculation, the MATLAB® function `polyfit` was used which executes "polynominal curve fitting in a least square sense", see `http://www.mathworks.de/de/help/matlab/ref/polyfit.html`

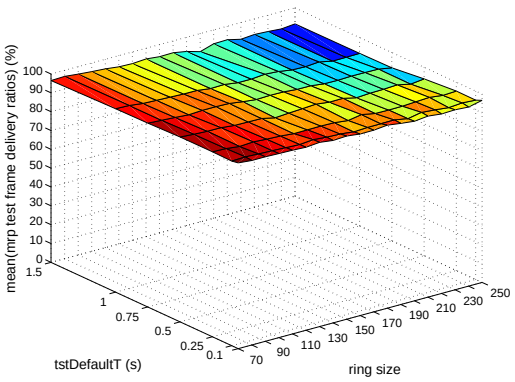


(a) ARP excessive phase $[0s, 50s)$

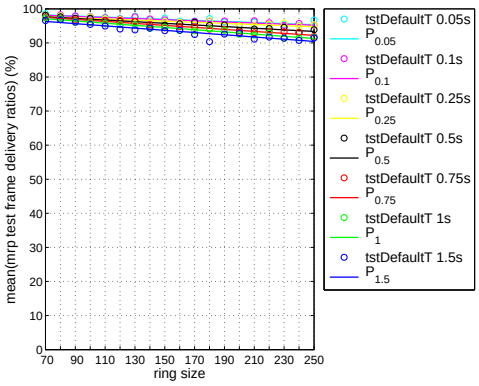


(b) ARP non-excessive phase $[50s, 120s]$

Figure 4.19: Means of MRP test frames delays and corresponding polynomial functions $MD_{tstDefaultT}$.



(a) Mean delivery ratios



(b) Mean delivery ratios and polynomials

Figure 4.20: Means of MRP test frame delivery ratios and the corresponding polynomials $MR_{tstDefaultT}$ for the whole simulation duration $[0s, 120s]$.

Considering the polynomial functions, the MRP frames delivery ratio of all scenarios is greater than 90%. Clearly, a higher frequency of MRP test frames leads to a better delivery ratio.

To sum up, the mean delays and standard deviations of MRP test frames increase proportionally with the ring size and inversely proportional to the parameter $tstDefaultT$. Thus, scenarios with smaller $tstDefaultT$ values do have higher delays with greater deviations.

Especially scenarios with $tstDefaultT$ being $0.05s$ or $0.1s$ have longer delays per ring size than the remaining scenarios. On the other hand, MRP test frames sent with a higher frequency

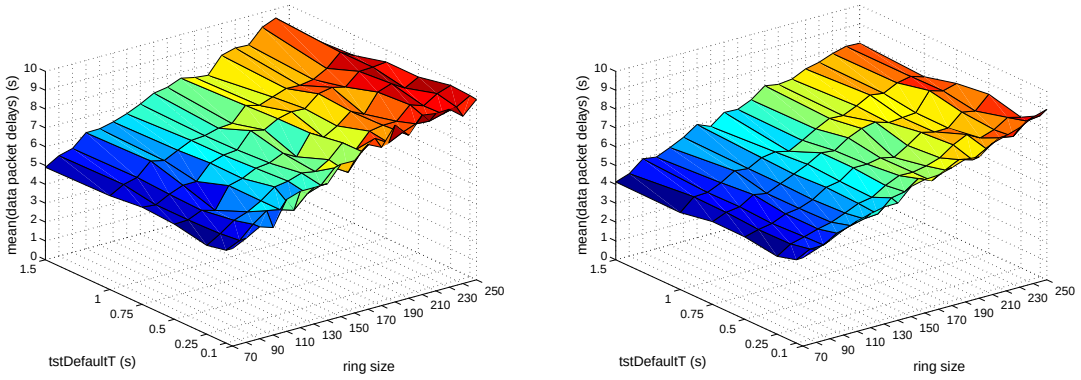
TstDefaultT	p_1	p_2
0.05s	-0.0160	99.3435
0.10s	-0.0162	99.2372
0.25s	-0.0163	98.8077
0.50s	-0.0239	99.3441
0.75s	-0.0293	99.5347
1.00s	-0.0322	99.3342
1.50s	-0.0326	98.5813

Table 4.4: Coefficients of $MR_{tstDefaultT}$ for mean MRP test frame delivery ratios.

(i.e. smaller $tstDefaultT$ value) have better delivery ratios. Thus, they are switched more slowly but with a higher delivery ratio. The ratios for all $tstDefaultT$ values and ring sizes are larger than 90%.

The mean data packet delays are depicted in Figure 4.21a and 4.21b for the ARP excessive and non-excessive phase respectively. During the excessive phase, the highest mean delay $9.698s$ results from scenario *scale(250, 0.25s)* and the lowest $4.713s$ from *scale(70, 0.25s)*.

In interval $[50s, 120s]$, the highest mean delay $9.052s$ results from scenario *scale(250, 0.05s)* and the lowest value $4.0834s$ from *scale(70, 1s)*. Thus, the delays are not strictly monotonically increasing.



(a) ARP excessive phase $[0s, 50s)$

(b) ARP non-excessive phase $[50s, 120s]$

Figure 4.21: Means of data packet delays partitioned in ARP excessive and non-excessive phase.

For a prediction of the data packet delays for arbitrary sizes of rings, a polynomial function $DD_{tstDefaultT}$ for each $tstDefaultT$ was derived from the experiments. The coefficients of these functions for the period $[50s, 120s]$ are provided in Table 4.5 and visualized in Figure 4.22b. The resulting polynomials for the phase $[0s, 50s)$ are illustrated in Figure 4.22a, their coefficients are listed in Table A.10. Intuitively, the mean delays represented by these polyno-

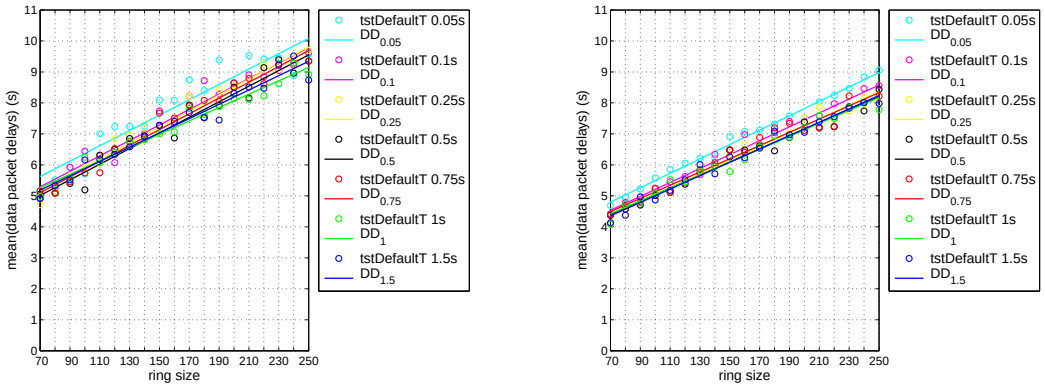
mials are shorter when MRP test frames are sent less frequently. The delays of all $tstDefaultT$ values and phases increase approximately equal with the ring size, i.e. the slope of the polynomials is about the same.

The polynomials of scenarios with $tstDefaultT \geq 0.1$ are within an envelope, whose width spans from $0.2s$ at $ring\ size = 70$ to $0.4s$ at $ring\ size = 250$ for the period $[50s, 120s]$, respectively to $0.7s$ at $ring\ size = 250$ for the interval $[0s, 50s)$.

Thus, the mean delays for $tstDefaultT \geq 0.1$ have a maximum dispersion of $0.4s$ for all ring sizes in the ARP non-excessive phase. The delays of polynomial $DD_{0.05}$ are outliers and are approximately $0.5s$ higher as the (mean of the) envelope in the phase $[0s, 50s)$, and $0.66s$ higher at maximum in the remaining simulation time.

TstDefaultT	p_1	p_2
0.05s	0.0232	3.1751
0.10s	0.0224	2.9635
0.25s	0.0223	2.8088
0.50s	0.0211	2.9249
0.75s	0.0213	2.9976
1.00s	0.0209	2.9615
1.50s	0.0216	2.8437

Table 4.5: Coefficients of $DD_{tstDefaultT}$ for mean data packet delays during $[50s, 120s]$.



(a) ARP excessive phase $[0s, 50s)$

(b) ARP non-excessive phase $[50s, 120s]$

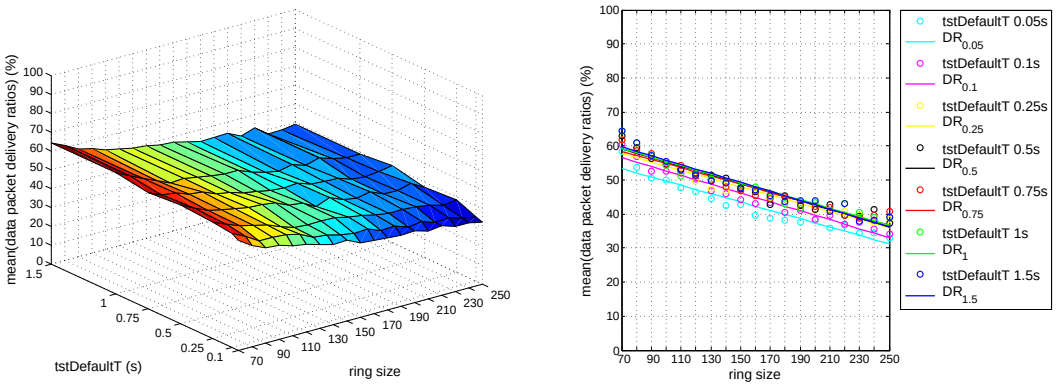
Figure 4.22: Means of data packet delays and corresponding polynomials $DD_{tstDefaultT}$.

The mean data packet delivery ratio of each scenario is shown in Figure 4.23a. Intuitively, the ratios decrease proportionally according to the ring size for each $tstDefaultT$. The ratios for large ring sizes are worse since a frame encounters more collisions while traveling over a large ring compared to a small ring. However, they do not strictly monotonically decrease

within a $tstDefaultT$ and the largest delta where they do not is 2.8%. The coefficients of the polynomials $DR_{tstDefaultT}$ fitting the results are stated in Table 4.6. The ratios derived from the polynomials of scenarios with $tstDefaultT$ equals $0.05s$ and $0.1s$ are clearly lower than of the rest, which lie in a 2% "tube" for all ring sizes (see Figure 4.23b). The smallest value of this tube for ring size 70 is $DR_{0.25}$ with 57.7% and the largest one is 59.7% from $DR_{1.5}$. The tube becomes more narrow at ring size 250, where the ratios range from 36% for $DR_{0.25}$ to 36.7% for DR_1.

TstDefaultT	p_1	p_2
0.05s	-0.1226	61.9217
0.10s	-0.1306	65.7739
0.25s	-0.1209	66.1934
0.50s	-0.1268	67.9966
0.75s	-0.1210	66.9475
1.00s	-0.1248	67.9202
1.50s	-0.1303	68.8464

Table 4.6: Coefficients of $DR_{tstDefaultT}$ for mean data packet delivery ratios.



(a) Mean delivery ratios

(b) Mean delivery ratios and polynomials

Figure 4.23: Means of data packet delivery ratios and corresponding polynomials $DR_{tstDefaultT}$ for the whole simulation duration $[0s, 120s]$.

Summarizing, different $tstDefaultT$ values do have an effect on the data packet delays and ratios. In general, the delays increase and the ratios decrease proportionally with the ring size. Using $tstDefaultT$ values of $0.05s$ or $0.1s$, the delays and ratios are obviously worse than the delays and ratios resulting from other $tstDefaultTs$. The delays of the latter ones disperse within $0.4s$ and $0.7s$ for $[50s, 120s]$ and $[0s, 50s)$ respectively, and the ratios within 2%. For all $tstDefaultT > 0.1s$ holds, that the resulting delays are approximately $0.5s$ better in the

ARP non-excessive than in the excessive phase. For the purpose of a better data transmission, a higher $tstDefaultT$ parameter like $1s$ or $1.5s$ is preferable, as more data are switched with shorter delays.

4.2 Results

In this work, several simulation studies were carried out on various ring topologies. All ring topologies consisted of CNs connected by half-duplex links with a bandwidth of $640kbps$. In every experiment, one node of the ring was configured as MRM, sending periodically MRP test frames and blocking one of its ports for non MGMT frames. Thus, the rings transformed into line topologies for non MGMT frames.
Except in the first study, the simulation results were analyzed based on a set of four metrics. This set includes the MRP test frame delay, the MRP test frame delivery ratio and the data packet delay and delivery ratio.

The first simulation study aimed to find a suitable workload generation function, which leads to a link utilization of two-third of the bandwidth at the bottleneck link. Additionally, investigations on the temporal and spatial distribution of the network utilization were performed. Therefore, throughput-metering channels were installed on multiple links of the ring. For the experiments, a ring of 64 nodes was constructed, where every node periodically sends data to every other node. Applicable periods for this traffic generation were found by an iterative simulation process (see Section 2.1.4), i.e. new alternatives were simulated after analyzing completed runs. Intuitively, the bottleneck link is the link in the middle of the line, i.e. in the opposite of the blocked port (w.r.t. the graphical representation), and the utilization decreases with increasing distance of a link to the bottleneck. The measured utilization of multiple links shows higher utilization in the first $50s$. After that, the utilization remains tolerably constant for the remaining simulation time. The higher utilization can be referred to extensive network communication produced by ARP.

In the following study, three different approaches for implementing and simulating ARP were investigated. One system alternative was the ARP version provided by INET, the other two used the implemented versions of ARP, either with upper-layer reachability confirmation (ULC) enabled or disabled. For the network setup, again a ring of 64 nodes was constructed and the traffic was produced as recommended by the first simulation study. In all three scenarios, the first $50s$ can be defined as ARP extensive. During this time, the implemented ARP with ULC enabled (scenario *ULC_enabled*) sends mainly multicast requests and after this period, it does not send requests at all. The INET version and the ARP with ULC disabled (scenario *ULC_disabled*) transmit approximately the same number of multicast requests in both the ARP excessive and non-excessive phase, but scenario *ULC_disabled* additionally sends unicast requests in the second period $[50s, 170s]$. This is caused by the fact that nodes delete unicast ARP requests with a MAC address not designated for them and hence do not update their cache table. All three scenarios have the same MRP test frame delivery ratio, but scenario *ULC_enabled* has shorter delays. In this scenario more data packets are delivered with a shorter delay.

A third simulation study was carried out to examine how different switch priorities used for

MRP test frames or data packets and $tstDefaultT$ values affect the performance of the ring. The focus was put on the MRP test frame delay (T_{ring}), i.e. the time a frame needs to be passed around the ring. Different $tstDefaultT$ values or MRP test frame priorities have no major effect on the data packet delays, neither in the ARP excessive nor non-excessive phase.

Using priority 3 instead of priority 2 for MRP test frames, results in better mean test frame delays though. In the ARP excessive phase, the improvement is about $3.7s$ and in the non-excessive period about $2.1s$ for all $tstDefaultT$. The delivery ratios of MRP test frames and data packets are similar for all scenarios (approximately 99% and 95%, respectively), thus independent of the $tstDefaultT$ value or priority. Consequently, a higher priority for MRP test frames is the better choice as it results in faster test frame delays, regardless of the chosen $tstDefaultT$ parameter or the considered point in time (at least for the chosen ring size). On the other hand, a higher data packet priority (but lower than the MRP frames' priority) leads to worse data packet delays and higher delivery ratio. Thus, data packets with lower priority are faster if they can be switched. Furthermore, data priority 1 or 0 have the same effect on MRP or data transmission performance during the interval $[50s, 170s]$.

The purpose of the last experiments was to study the scalability of the MRP under conditions, which violate the requirements provided by the standard for a correct functioning of the protocol. These restrictions concern the bandwidth and operating mode of the links, together with the size of the ring. Additionally to the scalability, various $tstDefaultT$ parameters ranging from $0.05s$ to $1.5s$ were investigated. For the experimental setup, ring topologies with sizes spanning from 70 to 250 were constructed. The connecting links performed with a bandwidth of $640kbps$ and in half duplex mode. One node acted as MRM, sent periodically test frames (MGMT frames) and blocked one of its ports. For workload generation in all scenarios, the 32 nodes on one end of the formed line periodically send data to the 32 nodes on the other end and vice versa. In this way, the bottleneck of the ring was utilized at about two-third of its bandwidth.
To simplify the analysis of the simulation metrics and to provide an estimator for arbitrary ring sizes, polynomial functions were derived from the simulation results for every metric. As in previous experiments, the simulated network utilization can be partitioned in an ARP excessive and non-excessive phase. Occasionally, ARP resolutions occurred after $50s$. As they do not utilize the network extensively, the ARP excessive phase was defined as $[50s, 120s]$. Generally, the means and standard deviations of MRP test frame and data packet delays are higher during this phase in all scenarios.
Concerning all four metrics, the scenarios with $tstDefaultT$ set to $0.05s$ or $0.1s$ (*high frequency* group) stand out from all others. The *low frequency* group contains the $tstDefaultT$ values $0.25s$, $0.5s$, $0.75s$, $1s$ and $1.5s$. The means of the test frames increase proportionally with the ring size and indirectly proportional with the $tstDefaultT$. During interval $[50s, 120s]$, the means range from about $2.6s$ at ring size 70, to about $5.3s$ at ring size 250 for low frequency $tstDefaultTs$. The means of the high frequency group rises from $2.8s$ to approximately $6.3s$. Thus, MRP test frames sent with a higher frequency result in mean delays which are about $1s$ longer for larger ring topologies. It holds for all $tstDefaultT$ parameters, that the MRP frame delivery ratio decreases with increasing ring sizes. Further, the delivery ratio falls faster with increasing $tstDefaultT$ value, i.e. the high frequency group results in better delivery ratios. Altogether, the delivery ratios of all scenarios are above 90%.

The mean data packet delay is about $4.5s$ for rings with size 70 during the second interval $[50s, 120s]$. For ring size 250, the low frequency group shows a mean delay of $8.3s$. The high frequency group has a mean delay of $8.8s$.

Hence, higher frequent test frames have a delaying effect on data packets. For all scenarios, the data packet delivery ratio is below 60%. In scenarios with $tstDefaultTs$ of the low frequency group, 58% or 36% of all data packets are switched for ring size 70 and 250, respectively. The high frequency group results in a 4% less delivery ratio. Consequently, MRP test frames sent with higher frequency lead to worse data packet delivery ratios.

To sum up, MRP test frames sent with a higher frequency lead to higher MRP test frame delays and data packet delays, higher MRP test frame delivery ratios, and a lower data packet delivery ratios. A reconfiguration of MRP after the ARP excessive phase optimizes the performance of MRP, at least for large ring sizes.

CHAPTER 5

Conclusion and Outlook

5.1 Summary

In this work, a given fault-tolerant IP communication system was analyzed regarding scalability and network performance. Additionally, the correctness of the communication scheme was investigated. For this purpose, a simulation methodology was applied. The network nodes of the examined system incorporate a switching function and an IP stack. They exchange data packets based on IP on links with bandwidths much smaller than state-of-the-art links in the scope of $Mbps$ or $Gbps$. The fault tolerance is established on the DLL by redundancy in form of a ring topology, which involves the necessity of *ring protection and recovery* mechanisms. Without such protocols, bridging loops form, resulting in message duplication and consequently in broadcast storms, which extensively consume network resources. On the other hand, redundant paths can be used in the case of a link or node failure. There exist multiple ring protection and recovery protocols to serve these purposes, which are extensively described in Chapter 2.

Some of the protocols listed in Chapter 2 are proprietary (REP, EPSR, RRPP, Turbo Chain) while others are standardized (STP, RSTP, MSTP, MRP, RRP, EAPS). Basically, they block (multiple) ports of switches for non MGMT frames to prevent bridging loops. Some of them are based on VLANs (MSTP, EAPS, EPSR, RRPP) or use it for load balancing (REP). VLANs are outlined in Chapter 2, too. Another section deals with simulation studies in general. It presents theoretical aspects of simulation like terms and definitions, purposes, advantages and disadvantages, a scheme for classification and describes how simulation studies are carried out. Besides that, a very technical insight is given into the fundamental functioning of discrete-event simulation environments. Finally, an overview of current simulation tools is provided.

A major task of this work was the model implementation of the communication system. First, different state-of-the-art simulation frameworks were examined and compared and based on this, the simulation environment OMNeT++ was selected. Then, the protocol framework INET of the simulation environment was modified and extended. During this process, an issue

of the scheduler and throughput-metering channel was identified. The ready-to-use INET version of the scheduler was not implemented according to the specification. More precisely, the scheduler was not fair and the throughput-metering channel measured incorrectly. Both modules were re-implemented in the context of this thesis. The modeled network node contains a specific switch and an Ethernet capable MCU. The switch operates at layer 2 of the ISO OSI reference model and supports QoS with four traffic classes. The MCU module models layer 2 and layer 3. On the top of the network layer, an application is based for data traffic generation and recording of statistical values. The standardized and — especially in the field of industrial automation — established protocol MRP was chosen as ring protocol. It is performed as functionality of the MCU and not as part of the switch, as it usually is the case in conventional industrial networks. Using MRP, the ring transforms to a line topology for non MGMT frames as the MRM blocks one of its port. The performance of MRP depends on T_{test}, which has to be larger than the delay of an MRP test frame, T_{ring}. Additionally, scripts for input data generation and output data processing were developed to complete the simulation workflow.

Subsequent to the implementation of the model, several simulation studies were carried out. All experiments based on ring topologies with the same bandwidth on every half duplex link and violating the requirements defined by the MRP standard concerning ring size, bandwidth and operation mode.

5.2 Conclusion

To conclude this work, the results of the simulation studies were analyzed. The aim of the first study was to figure out how an appropriate network utilization for other experiments can be produced. The required utilization was two-third of the bandwidth at the network's bottleneck link. Moreover, the temporal and spatial distribution of the network utilization in the rings were examined. As shown, the utilization decreases with increasing distance of the link to the middle of the formed line. The links at the middle are considered as bottlenecks of the line. Furthermore, in the first minute of data communication, a much higher utilization was identified. This higher utilization is caused by traffic of ARP.

In the second study, the ARP provided by INET and an enhanced version were compared. The latter one is based on RFC 2461 (*Neighbor Discovery for IP Version 6 (IPv6)*) and resolves stale ARP cache entries with unicast resolutions instead of broadcast resolutions as in the original ARP (RFC 826). Additionally, it supports upper-layer reachability confirmation. Clearly, the enhanced version of ARP (with ULC enabled) improves the performance, meaning that shorter MRP test frames were achieved. Also, more data packets with shorter delays were switched. Interestingly, the ARP according to RFC 826 leads to a better performance than the enhanced one with ULC disabled. Consequently, when a real system with a kernel module that implements the newer version of ARP, is developed, it is essential to perform ULC. For instance, this can be done by the MSG_CONFIRM flag of the Linux send function. Since the MRP test frame delays are shorter in the ARP non-excessive phase, a reconfiguration of the MRP parameters after the first minute allows improving the MRP behavior.

In a further step, it was tried to optimize the MRP test frame delays. The experiments were conducted with different $tstDefaultT$ values and switch priorities for test frames and data

packets. A higher priority for test frames reduces the test frame delay, but has no effect on the data packet delays or delivery ratios. Using a higher data packet priority (but lower than the test frames' priorities), results in a better data packet delivery ratio but worse delays, i.e. data packets with lower priority are faster if they can be switched. There was no performance difference when using the lowest and second lowest priority for data packets during the ARP non-excessive phase.

Finally, the scalability of MRP was investigated. Experiments with 19 different ring sizes and seven different $tstDefaultT$ parameters showed, that higher $tstDefaultT$ values lead to lower test frame delays (T_{ring}), lower data packet delays and higher data packet delivery ratios, which is in line with the intuition. In contrast, the resulting MRP test frame delivery ratios are lower but still above 90% for all ring sizes. Consequently, it is better to use higher $tstDefaultT$ values to gain a better performance. The resulting worse test frame delivery ratios are negligible for the ring integrity test, since it is more important that within an interval of T_{test}, *at least one* frame reaches the MRM with a *small* delay, than *all* or *many* test frames reach the MRM with a large delay. Receiving at least one frame, the MRM is already able to detect the integrity of the ring. Hence, also the convergence time of MRP, which depends on T_{test}, can be reduced. The improvement is possible as T_{test} can be shortened when T_{ring} is minimized. On the other hand, test frames sent with a higher $tstDefaultT$ extend the detection of a ring recovery. This can be bypassed with very short $tstShortT$ values upon the reception of a LinkUp frame. Besides, a recovery from an open ring forms an exception compared to the integrity test, or at least *shall* form an exception.

Furthermore, a problem regarding MRP would emerge for a real system which corresponds to the implemented model of the network node. According to that, the MRC module fails and does not forward test frames anymore, when the MCU running the MRP fails. Hence, the MRM would introduce a switch-over due to the absence of test frames although all links and switches are working, and consequently create a bridging loop.

5.3 Lessions learned

During the analysis of MRP, possible improvements for slow Ethernet connections were discovered. The MRM protocol state machine provided in [47] introduces an unnecessary *topology change* of the ring after a certain sequence of events. This can happen when the transmission of a test frame over the whole ring lasts longer as the time the MRM needs to detect that one of its ports went down. When a topology change is performed, the MRM and all MRCs having their two ring ports up, flush their FDB. A segment of the state machine is shown in Figure 5.1. The depicted states can be described as [47]:

- The MRM is in the *PRM_UP (Primary Ring Port with Link Up)* state, when only one port has a link. This port is called the *primary port*. Although the *secondary port* is down, the MRM periodically sends test frames.

- The MRM transitions to *CHK_RO (Check Ring, Ring Open State)*, when it has not received test frames for a given period of time.

- In the state *CHK_RC (Check Ring, Ring Closed State)*, the MRM shall send test frames and check the link of its ring ports to test the integrity of the ring.

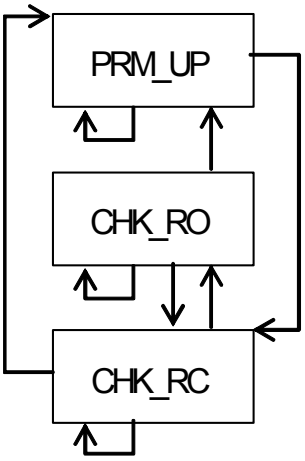


Figure 5.1: A segment of the protocol state machine of the MRM according to [47].

Furthermore, the parameter *REACT_MODE* specifies, if the MRM should react immediately on a received *LinkDown* frame and initiate a topology change. The $tstDefaultT$ value defines the period the MRM sends test frames with and T_{ring} is the time an MRP test frame needs to be transmitted over the whole ring. It is computed as follows [47]:

$$T_{ring} = N * (T_{switch} + T_{queue} + T_{bit} + T_{line})$$

where

- N is the number of switches in the ring.
- T_{switch} is the delay introduced by each switch.
- T_{queue} is the delay added due to a switch's queue.
- T_{bit} is the time needed by a switch to transmit a frame.
- T_{line} is the propagation delay of the physical medium.

Table 5.1 describes the sequence of transitions which leads to unnecessary flushes of the FDBs. The topology changes, initiated in *transition36* and *transition47*, are unnecessary as the secondary port was already in blocking state before it went down. Thus, the topology has not changed from the MRP nodes' point of view. Assuming that all links on the ring use the same bandwidth, the time window during which *transition42* has to happen depends on T_{ring} and is

$$T_{critical} \approx T_{ring} - T_{oneNodeDistance} = \frac{N-1}{N} * T_{ring}$$

The problem can be overcome by an additional condition asking about the state of the secondary link in *transition13*. When the secondary link is down, the state machine shall remain in the state PRM_UP.

State/transition	**Description**	**Link states** sec./prim.	**Port states** sec./prim.
CHK_RC	The ring is closed. Pairs of test frames are periodically sent and received as long as there is no failure. (Test frame *tf2* is sent out of the secondary port.) Assuming that $T_{ring} << tstDefaultT$, there are no other test frames except the currently sent ones on the ring.	up/up	blocking/ forwarding
transition42	Immediately after tf2 has passed the neighbor next to the secondary port, the secondary port's link goes down. The corresponding MRC starts transmitting LinkDown frames.		
PRM_UP	Only the link at the primary port is up.	down/up	blkd/fwd
transition13	The MRM receives tf2 on its primary port. *No_TopologyChange_flag* is set to false. The MRM is initiated to send test frames.		
	Two transitions are possible depending on the configuration of *REACT_MODE*.		
CHK_RC	REACT_MODE is disabled: The MRM remains in CHK_RC as long as the timer has not timed out *tstNRmax* times. It sends additional test frames after *tstShortT* on reception of a *LinkDown* frame from the MRC.	down/up	blkd/fwd
transition36	The timer timed out for the tstNRmax time. The MRM introduces a *topology change* and sets the secondary port to forwarding.		
CHK_RC	or REACT_MODE is enabled: The MRM remains in CHK_RC as long as the timer has not timed out tstNRmax times or has not received a LinkDown frame.	down/up	bldk/fwd
transition47	The MRM received a LinkDown frame. The MRM introduces a *topology change* and sets the secondary port to forwarding.		
CHK_RO	The MRM continues testing the ring integrity.	down/up	fwd/fwd

Table 5.1: Two possible scenarios leading to an unnecessary *topology change* event

However, this issue is irrelevant when the Ethernet interface is not able to detect a failed link within the time $T_{critical}$. For example, in a ring with 50 switches, $100Mbps$ links, $T_{switch} = 10\mu s$, $T_{queue} = 0s$, a frame size of 64 bytes and a propagation delay of $T_{line} = 0.5\mu s$, T_{ring} is $781\mu s$ and $T_{critical} = 765\mu s$. Having connections with $10Mbps$, $T_{critical}$ is $3.023ms$.

Nevertheless, it becomes considerable when the ring consists of very slow half duplex connections, where the transmission of MRP test frames indeed can take more than a second (as in the experiments in Section 4.1).

5.4 Future work

As the model implements the complete MRP standard, further experiments with link failures on different locations of the ring can be conducted. It can be investigated how various parameters for the topology change or the react mode of the MRM influence the convergence time of MRP in such slow environments.

Apart from simple ring topologies, more complex networks like coupled rings or mesh topologies can be built and examined deploying the implemented framework. Besides from analyzing existing networks, this framework can be used for developing new fault-tolerant communication schemes.

In order to contribute to the improvement of OMNeT++, the discovered issues will be reported to the community.

List of Figures

List of Tables

APPENDIX A

Tables

TstDefaultT	**0.1s**		**0.25s**		**0.5s**	
Priority	**2**	**3**	**2**	**3**	**2**	**3**
mean	9.663s	6.093s	9.624s	5.996s	8.954s	5.446s
std. dev.	2.743s	3.059s	2.655s	3.184s	2.653s	2.902s
median	9.389s	5.730s	9.496s	5.503s	8.815s	5.114s
maximum	20.961s	20.180s	22.392s	20.300s	19.279s	16.47s

Table A.1: MRP test frame delays during the ARP excessive phase $[0s, 50s)$ using different $tstDefaultT$ values ($0.1s, 0.25s$, $0.5s$) and MRP test frame priorities.

TstDefaultT	**0.75s**		**1s**		**2s**	
Priority	**2**	**3**	**2**	**3**	**2**	**3**
mean	9.023s	5.482s	9.302s	5.670s	9.267s	5.173s
std. dev.	2.935s	3.029s	3.186s	3.059s	3.481s	3.078s
median	8.754s	5.057s	9.026s	5.282s	9.130s	4.707s
maximum	21.657s	17.265s	20.690s	17.984s	21.131s	17.245s

Table A.2: MRP test frame delays during the ARP excessive phase $[0s, 50s)$ using different $tstDefaultT$ values ($0.75s, 1s$, $2s$) and MRP test frame priorities.

TstDefaultT	**0.1s**		**0.25s**		**0.5s**	
Priority	**2**	**3**	**2**	**3**	**2**	**3**
mean	4.996s	2.660s	4.444s	2.348s	s 4.273s	2.184s
std. dev.	1.564s	1.369s	1.503s	1.293s	1.453s	1.228s
median	4.867s	2.558s	4.3087s	2.229s	4.157s	2.073s
maximum	11.800s	8.604s	11.281s	9.397s	11.662s	10.061s

Table A.3: MRP test frame delays during the ARP non-excessive phase $[50s, 170s]$ using different $tstDefaultT$ values ($0.1s, 0.25s$, $0.5s$) and MRP test frame priorities.

TstDefaultT	**0.75s**		**1s**		**2s**	
Priority	**2**	**3**	**2**	**3**	**2**	**3**
mean	4.224s	2.149s	4.148s	2.173s	4.211s	2.127s
std. dev.	1.446s	1.211s	1.478s	1.207s	1.471s	1.178s
median	4.115s	2.047s	4.014s	2.058s	4.045s	2.015s
maximum	10.688s	7.432s	13.341s	8.225s	10.799s	7.362s

Table A.4: MRP test frame delays during the ARP non-excessive phase $[50s, 170s]$ using different $tstDefaultT$ values ($0.75s, 1s$, $2s$) and MRP test frame priorities.

TstDefaultT	**0.1s**		**0.25s**		**0.5s**	
Priority	**2**	**3**	**2**	**3**	**2**	**3**
MRP test f. mean	98.46%	99.387%	98.858%	99.221%	98.827%	99.26%
MRP test f. std. dev.	0.588%	0.384%	0.734%	0.376%	0.540	0.367%
MRP test f. median	98.442%	99.470%	99.045%	99.229%	98.900%	99.340%
Data p. mean	95.464%	95.48%	95.528%	95.685%	95.479%	95.533%
Data p. std. dev.	5.764%	5.706%	5.619%	5.573%	5.806%	5.700%
Data p. median	95.652%	95.652%	95.652%	95.652%	95.652%	95.833%

Table A.5: Delivery ratios of MRP test frames and data packets using different $tstDefaultT$ values ($0.1s, 0.25s$, $0.5s$) and MRP test frame priorities.

TstDefaultT	**0.75s**		**1s**		**2s**	
Priority	**2**	**3**	**2**	**3**	**2**	**3**
MRP test f. mean	99.04%	99.242%	98.838%	98.896%	98.314%	98.314%
MRP test f. std. dev.	0.569%	0.315%	0.556%	0.456%	0.495%	0.728%
MRP test f. median	99.231%	99.232%	98.830%	98.976%	98.256%	98.547%
Data p. mean	95.382%	95.637%	95.514%	95.579%	95.706%	95.514%
Data p. std. dev.	5.778%	5.642%	5.651%	5.646%	5.425%	0.728%
Data p. median	95.652%	95.833%	95.652%	95.833%	95.652%	98.547%

Table A.6: Delivery ratios of MRP test frames and data packets using different $tstDefaultT$ values ($0.75s, 1s$, $2s$) and MRP test frame priorities.

	[0s, 50s)			[50s, 170s)		
Delays	***priority(2)***	***priority(1)***	***priority(0)***	***priority(2)***	***priority(1)***	***priority(0)***
MRP test f. mean	5.670s	4.386s	4.206s	2.173s	1.863s	1.857s
MRP test f. std. dev.	3.059s	2.951s	2.921s	1.207s	1.020s	1.024s
MRP test f. median	5.282s	3.720s	3.549s	2.058s	1.803s	1.760s
MRP test f. maximum	17.984s	16.832s	17.674s	8.225s	6.315s	6.693s
Data p. mean	3.786s	2.553s	2.701s	2.034s	1.692s	1.683s
Data p. std. dev.	3.372s	2.528s	2.711s	1.781s	1.482s	1.485s
Data p. median	3.076s	1.902s	1.998s	1.719s	1.410s	1.389s
Data p. maximum	21.852s	19.586s	24.046s	17.498s	13.569s	12.815s

Table A.7: Delays of MRP test frames and data packets using the data priorities 2,1 and 0.

Delivery ratios	***priority(2)***	***priority(1)***	***priority(0)***
MRP test f. mean	98.896%	99.057%	99.079%
MRP test f. std. dev.	0.456%	0.360%	0.378%
MRP test f. median	98.977%	99.123%	99.123%
Data p. mean	95.579%	90.804%	90.609%
Data p. std. dev.	5.646%	8.335%	8.246%
Data p. median	95.833%	91.304%	91.304%

Table A.8: Delivery ratios of MRP test frames and data packets using the data priorities 2,1 and 0.

TstDefaultT	$\mathbf{p_1}$	$\mathbf{p_2}$
0.05s	0.0222	3.0165
0.10s	0.0225	2.3722
0.25s	0.0204	2.3232
0.50s	0.0185	2.2737
0.75s	0.0172	2.3352
1.00s	0.0152	2.4532
1.50s	0.0152	2.3372

Table A.9: Coefficients of $MD_{tstDefaultT}$ for mean MRP test frame delays during the interval $[0s, 50s)$.

tstDefaultT	**p_1**	**p_2**
0.05s	0.0248	3.8929
0.10s	0.0250	3.5461
0.25s	0.0259	3.3357
0.50s	0.0252	3.2503
0.75s	0.0255	3.3150
1.00s	0.0216	3.7549
1.50s	0.0231	3.5701

Table A.10: Coefficients of $DD_{tstDefaultT}$ for mean data packet delays during $[0s, 50s)$.

Bibliography

[1] W. Kastner and G. Neugschwandtner, *Data Communication for Distributed Building Automation*, Richard Zurawski, Ed. Boca Raton, Florida, USA: CRC Press, 2009.

[2] ——, "Datenkommunikation in der verteilten Gebäudeautomation," *Bulletin SEV/VSE*, vol. 17, pp. 9–14, Aug. 2006.

[3] J. Banks, J. S. Carson II, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*, 5th ed. Upper Saddle River, New Jersey: Prentice Hall, 2010.

[4] W. D. Kelton, R. P.Sadowski, and N. B. Swets, *Simulation with Arena*, 5th ed. New York City, USA: McGraw-Hill, 2010.

[5] T. Sauerbier, *Theorie und Praxis von Simulationssystemen*, 1st ed. Wiesbaden, Germany: Vieweg, 1999, german.

[6] B. Acker, *Simulationstechnik Grundlagen und praktische Anwendungen*. Renningen, Germany: expert verlag, 2011, german.

[7] J. Potemans, J. Theunis, B. Rodiers, B. V. den Broeck, P. Leys, E. V. Lil, and A. V. de Capelle, "Simulation of a campus backbone network, a case-study," www.esat.kuleuven.be/telemic/networking/opnetwork02_jan.pdf, Aug. 2002, accessed: 2012-06-01.

[8] A. Varga, "OMNeT++," in *Modeling and Tools for Network Simulation*, K. Wehrle, M. Günes, and J. Gross, Eds. Springer Berlin Heidelberg, 2010, pp. 35–59.

[9] M. Tüxen, I. Rüngeler, and E. P. Rathgeb, "Interface connecting the INET simulation framework with the real world," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops (Simutools)*. Brussels, Belgium: ICST, 2008, pp. 40:1–40:6.

[10] B. Penoff, A. Wagner, M. Tüxen, and I. Rüngeler, "MPI-NeTSim: A Network Simulation Module for MPI," in *Proceedings of the 2009 15th International Conference on Parallel and Distributed Systems (ICPADS)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 464–471.

[11] I. Rüngeler, M. Tüxen, B. Penoff, and A. Wagner, "A New Fast Algorithm for Connecting the INET Simulation Framework to Applications in Real-time," in *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques (Simutools)*. Brussels, Belgium: ICST, 2011, pp. 75–80.

[12] András Varga and OpenSim Ltd., "OMNet++ User Manual Version 4.2.2," 2011.

[13] M. A. Rahman, A. Pakštas, and F. Z. Wang, "Network modelling and simulation tools," *Simulation Modelling Practice and Theory*, vol. 17, no. 6, pp. 1011 – 1031, 2009.

[14] Paul Meeneghan and Declan Delaney, "An Introduction to NS, Nam and OTcl scripting," National University of Ireland, Maynooth, Tech. Rep., May 2004.

[15] D. Bunyai, "ZigBee Network Layer Simulation on top of IEEE 802.15.4," Master's thesis, Vienna University of Technology, Vienna, Austria, 2012.

[16] J. M. Pullen, "The Network Workbench: Network Simulation Software for Academic Investigation of Internet Concepts," *Computer Networks*, vol. 32, no. 3, pp. 365–378, Mar. 2000.

[17] Z. Di and H. Mouftah, "QUIPS-II: a simulation tool for the design and performance evaluation of Diffserv-based networks," *Computer Communications*, vol. 25, no. 11-12, pp. 1125 – 1131, 2002.

[18] M. B. Dumas and M. Schwartz, *Principles of Computer Networks and Communications*, 1st ed. Upper Saddle River, New Jersey: Pearson Prentice Hall, 2009.

[19] A. S. Tanenbaum and D. J. Wetherall, *Computernetzwerke*, 5th ed. Hallbergmoos, Deutschland: Pearson Deutschland GmbH, 2012.

[20] T. Lammle, *CCNA INTRO Introduction to Cisco Networking Technologies STUDY GUIDE*. Indianapolis, Indiana: Wiley Publishing, Inc., 2006.

[21] IEEE Computer Society, "Media Access Control (MAC) Bridges and Virtual Bridge Local Area Networks (IEEE 802.1Q-2011)," Aug. 2011.

[22] M. Valentine and K. Barker, *Cisco CCENT ICND1 100-101 Exam Cram*. Pearson Education, 2013.

[23] R. Seifert and J. Edwards, *The All-New Switch Book: The Complete Guide to LAN Switching Technology*, 2nd ed. Indianapolis, Indiana: Wiley Publishing, Inc., 2008.

[24] IEEE Computer Society, "Ethernet (IEEE 802.3-2012)," 2012.

[25] D. Hardy, G. Malléus, and J.-N. Méreur, *Networks Internet Telephony Multimedia*, 1st ed. De Boeck Diffusion s.a., 2002.

[26] L. L. Peterson and B. S. Davie, *Computernetze*, 1st ed. Heidelberg, Germany: dpunkt.verlag GmbH, 2008, german.

[27] Cisco, "VLAN Security White Paper," http://www.cisco.com/en/US/products/hw/switches/ps708/products_white_paper09186a008013159f.shtml#wp39042, accessed: 2014-03-22.

[28] D. K. Michael G. Solomon, *Fundamentals of communications and networking*, 1st ed. Burlington, Massachusetts, USA: Jones & Bartlett Learning, 2013.

[29] Cisco Systems, Inc., "Cisco Resilient Ethernet Protocol," http://www.cisco.com/c/dam/en/us/products/collateral/switches/me-3400-series-ethernet-access-switches/prod_white_paper0900aecd806ec6fa.pdf, accessed: 2014-02-26.

[30] Allied Telesis, "EPSRing Ethernet Protection Switched Ring," http://www.alliedtelesis.com/media/pdf/EPSR_White_Paper_REVD1.pdf, accessed: 2014-03-03.

[31] H3C, "RRPP Technology White Paper," http://www.h3c.com/portal/Products___Solutions/Technology/LAN/Technology_White_Paper/200810/618495_57_0.htm, 2008, accessed: 2014-03-04.

[32] Sean Wang, "Turbo Chain: A New Recovery System Beyond Ethernet Redundant Ring Technology," http://www.automation.com/pdf_articles/moxa/MOXA_Turbo_Chain.pdf, Oct. 2009, accessed: 2014-03-07.

[33] A. S. Tanenbaum, *Computer Networks*, 4th ed. Upper Saddle River, New Jersey: Prentice Hall, Aug. 2002.

[34] I. Javvin Technologies, *Network Protocols Handbook*. Saratoga, California, USA: Javvin Technologies, Inc, 2005.

[35] D. Hucaby, *CCNP BCMSN Exam Certification Guide*. Indianapolis, Indiana, USA: Cisco Press, 2004.

[36] W. Odom, *CCNA ICND2 Official Exam Certification Guide Second Edition*, 2nd ed. Indianapolis, Indiana, USA: Cisco Press, 2008.

[37] M. Ford, H. K. Lew, S. Spanier, and T. Stevenson, *Handbuch Netzwerk-Technologien*. München, Germany: Markt&Technik Buch- und Software-Verlag GmbH, 1998, german.

[38] IEEE Computer Society, "IEEE Standard for Local and metropolitan area networks, Media Access Control (MAC) Bridges and Virtual Bridge Local Area Networks (IEEE 802.1D-2004)," Jun. 2004.

[39] ——, "IEEE Standards for Local and Metropolitan Area Networks - Amendment to 802.1Q Virtual Bridged Local Area Networks: Multiple Spanning Trees (IEEE 802.1s-2002)," 2002.

[40] C. E. Spurgeon and J. Zimmerman, *Ethernet The Definitive Guide*, 2nd ed. Sebastopol, California, USA: O'Reilly & Associates, 2014.

[41] H. Reynolds and D. Marschke, *JUNOS Enterprise Switching*, 1st ed. Sebastopol, California, USA: O'Reilly & Associates, 2009.

[42] D. Bokotey, A. Mason, and R. Morrow, *CCIE Practical Studies Security*, 1st ed. Indianapolis, Indiana, USA: Cisco Press, 2003.

[43] European Committee for Electrotechnical Standardization, "Industrial communication networks – High availability automation networks – Part7: Ring-based Redundancy Protocol (RRP) (IEC 62439-7)," Brussels, Belgium, Mar. 2012.

[44] IETF, "Extreme Networks' Ethernet Automatic Protection Switching (EAPS) Version 1 (RFC 3619)."

[45] Extreme Networks, "Ethernet Automatic Protection Switching (EAPS)," Extreme Networks, Tech. Rep., Nov. 2006.

[46] IETF, "Extreme Networks' Ethernet Automatic Protection Switching (EAPS) Version 1.3 (RFC 3619)."

[47] European Committee for Electrotechnical Standardization, "Industrial communication networks – High availability automation networks – Part2: Media Redundancy Protocol (MRP) (IEC 62439-2)," Berlin, Germany, Sep. 2010.

[48] A. Spillner and T. Linz, *Basiswissen Softwaretest*, 4th ed. Heidelberg, Germany: dpunkt.verlag, 2010, german.

[49] Varga, András and Hornig, Rudolf, "An Overview of the OMNeT++ Simulation Environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (Simutools)*, ICST, Brussels, Belgium, 2008, pp. 60:1–60:10.

[50] Marvell, "Link Street 88E6063 Datasheet," Nov. 2004.

[51] IETF, "Neighbor Discovery for IP Version 6 (IPv6) (RFC 2461)."

[52] ——, "ARP (RFC 826)."

[53] R Development Core Team, "R: A language and environment for statistical computing," http://www.R-project.org/, R Foundation for Statistical Computing, Vienna, Austria, 2011.